NLC_2-DECOMPOSITION IN POLYNOMIAL TIME

ÖJVIND JOHANSSON*
Department of Numerical Analysis and Computing Science,
Royal Institute of Technology, SE-100 44 Stockholm, Sweden

ABSTRACT

NLC_k is a family of algebras on vertex-labeled graphs introduced by Wanke. An NLC-decomposition of a graph is a derivation of this graph from single vertices using the operations in question. The width of the decomposition is the number of labels used, and the NLC-width of the graph is the smallest width among its NLC-decompositions. Many difficult graph problems can be solved efficiently with dynamic programming if an NLC-decomposition of low width is given for the input graph. It is unknown though whether arbitrary graphs of NLC-width at most k can be decomposed with k labels in polynomial time. So far this has been possible only for $k = 1$, which corresponds to cographs. In this paper, an algorithm is presented that works for $k = 2$. It runs in $O(n^4 \log n)$ time and uses $O(n^2)$ space. Related concepts: clique-decomposition, clique-width.

Keywords: Graph algebra; graph decomposition; NLC-width; clique-width; cograph.

1 Introduction

In [15] Wanke introduces an algebra for a class of vertex-labeled graphs called NLC_k. This class consists of all graphs that can be obtained from single vertices with labels in $[k] = \{1, \ldots, k\}$ using the two operations of the algebra, union and relabeling, defined as follows: *Union* requires two disjoint graphs, and permits edges to be drawn between these. More precisely, all such edges will be added that match a set of ordered label pairs accompanying the union operator. *Relabeling* requires just one graph, and changes its labels according to a likewise specified mapping from $[k]$ to $[k]$. With $k = 1$, one obtains the class already known as *cographs*, described in [2] for example. (In such a comparison, we are referring to edge-structure only, since cographs are unlabeled.)

A similar algebra has been defined in [6]. The main difference is that in the latter, no edges can be added when two graphs are united; edge-drawing is a separate unary operator.

*ojvind@nada.kth.se

By a *decomposition* of a graph with respect to one of these algebras, we mean a derivation of this graph from single vertices using the operations in question. Specifically, the terms *NLC-decomposition* and *clique-decomposition* refer to the two algebras discussed above. The *width* of a decomposition is the number of distinct labels actually used, and the *NLC-width* and *clique-width* of a graph are the smallest widths among all its NLC-decompositions and clique-decompositions respectively.

The relationship between these algebras has been studied in [10]. It was found that a clique-decomposition can be transformed into an NLC-decomposition of the same graph and vice versa. In the first direction, no additional labels are needed. In the second direction, at most a doubling of the label set may be necessary. Thus, the NLC-width of a graph is bounded by the clique-width, which in turn is bounded by two times the NLC-width. NLC-width 1 (cographs) corresponds exactly to clique-width 1 and 2. Less clear is the relationship between the classes with NLC-width 2 and clique-width 3 for example. They may intersect properly. An indication for this is given by the fact that neither of the two bounds above can be improved by a constant multiplicative factor.

NLC-decompositions and clique-decompositions have a binary tree structure. What makes them important is that decomposing a graph can be an excellent first step in solving more particular problems on it. Many problems which are hard for arbitrary graphs can be solved with dynamic programming in polynomial or even linear time on graphs which can be decomposed using a bounded number of labels, assuming that the graph is given in such a decomposed form. For example, decision, optimization, and enumeration problems expressible in MS1 logic, such as 3-Colorability, MaxClique and #MaxClique, can be solved in linear time on graphs given as clique-decompositions of width at most k [4, 5]. And *P-recognizable* problems, such as Hamiltonian Circuit (which is not MS1-expressible [4]), can be solved in polynomial time on graphs given as NLC-decompositions of width at most k [15]. Note here that in theory it does not really matter which decomposition we have. For the transformations between NLC-decompositions and clique-decompositions mentioned above can in fact be carried out in linear time.

However, these transformations do not necessarily preserve minimality of width. Since the time complexities of the dynamic programming algorithms in question grow quickly with increasing k, it is of practical interest to use "first-hand" decompositions for that algebra which best captures a particular graph problem. It is by no means clear though that the problems of finding NLC-decompositions and clique-decompositions of minimal width are equally hard. In fact, it is unknown whether arbitrary graphs of NLC-width (clique-width) at most k can be NLC-decomposed (clique-decomposed) with k labels in polynomial time. For cographs, algorithms follow easily from [3], for example. A more recent result concerns certain families of graphs with restrictions on the number of induced P_4s [11]. (A P_4 is shown in Fig. 3.)

With the algorithm in this paper, it is now possible to NLC-decompose, using a minimum number of labels, all graphs of NLC-width at most 2 in polynomial

time. Concerning these, one can note that although cographs are equivalent with P_4-free graphs [2], a graph with NLC-width 2 — as well as one with clique-width 3 — can have an exponential number of induced P_4s. (Consider for example those fourpartite graphs whose edges we can define by letting each vertex part correspond to one of the vertices in a P_4.)

It was pointed out in [6] that clique-decomposition can refine the *modular decomposition* of a graph. This refinement idea works equally well for NLC-decomposition. In either case, a minimum-width decomposition of a graph G can be obtained from minimum-width decompositions of the *quotient* graphs in the modular decomposition of G. Accordingly, the algorithm presented in this paper uses modular decomposition as a first step. Thus, in Section 4 we define modular decomposition for labeled graphs, and we investigate the properties of the resulting quotient graphs. In Section 5 we then show how to NLC-decompose these quotient graphs, as long as their NLC-width is at most 2. We indeed exploit some observations particular to NLC-width 2, and no generalization to higher width seems readily obtainable.

2 Preliminaries

Unless stated otherwise, a *graph* G is assumed to be undirected, but it may be either labeled (see below) or unlabeled. $\mathrm{V}(G)$ and $\mathrm{E}(G)$ denote the vertex and edge sets of G, and (V, E) denotes the unlabeled graph with vertex and edge sets V and E.

With a *labeled graph* G we mean the graph $(\mathrm{V}(G), \mathrm{E}(G))$, also denoted $\mathrm{unlab}(G)$, together with a labeling function, lab_G, mapping each vertex in $\mathrm{V}(G)$ to a positive integer. G may be denoted by $(\mathrm{V}(G), \mathrm{E}(G), lab_G)$. $\mathrm{L}(G)$ denotes the set of all labels in G, that is, $\{lab_G(v) : v \in \mathrm{V}(G)\}$. Often we will require that $\mathrm{L}(G) \subseteq \{1, \ldots, k\}$. We denote this set by $[k]$. If all vertices in a set $V \subseteq \mathrm{V}(G)$ have (that is, are mapped to) the same label (by lab_G), we say that V is *uniformly labeled* (in G). If this holds for $V = \mathrm{V}(G)$, G is uniformly labeled.

Two graphs G_1 and G_2 are disjoint when $\mathrm{V}(G_1) \cap \mathrm{V}(G_2) = \emptyset$. Then, if G_1 and G_2 are both unlabeled or both labeled, their *disjoint union* G is defined as follows: In either case, $\mathrm{V}(G) = \mathrm{V}(G_1) \cup \mathrm{V}(G_2)$ and $\mathrm{E}(G) = \mathrm{E}(G_1) \cup \mathrm{E}(G_2)$. In case G_1 and G_2 are both unlabeled, so is G. In case G_1 and G_2 are both labeled, G is labeled too, with $lab_G(u) = lab_{G_1}(u)$ for all $u \in \mathrm{V}(G_1)$, and $lab_G(u) = lab_{G_2}(u)$ for all $u \in \mathrm{V}(G_2)$. The disjoint union of three or more graphs is defined analogously.

For a set of vertices V in a graph G, $G|V$ denotes the subgraph of G induced by V. The usual definition for unlabeled graphs is extended to labeled graphs in the obvious way.

3 NLC-Decomposition

In this section we give basic definitions and lemmas related to NLC-decomposition. We begin with the two fundamental graph operations.

Definition 1 (Union [15]) Let G_1 and G_2 be disjoint graphs labeled with numbers in $[k]$, and let $S \subseteq [k]^2$ (that is, S is a set of ordered label pairs). Then

$\times_S(G_1, G_2)$ is defined as the graph obtained by forming the disjoint union of G_1 and G_2, and adding to that all edges $\{u, v\}$ satisfying $u \in \mathrm{V}(G_1)$, $v \in \mathrm{V}(G_2)$, and $(lab_{G_1}(u), lab_{G_2}(v)) \in S$. See Fig. 1.

Definition 2 (Relabeling [15]) Let G be a graph labeled with numbers in $[k]$, and let R be a mapping from $[k]$ to $[k]$. Then $\circ_R(G)$ is the labeled graph G' defined by $\mathrm{V}(G') = \mathrm{V}(G)$, $\mathrm{E}(G') = \mathrm{E}(G)$, and $lab_{G'}(u) = R(lab_G(u))$ for all $u \in \mathrm{V}(G')$. See Fig. 1.

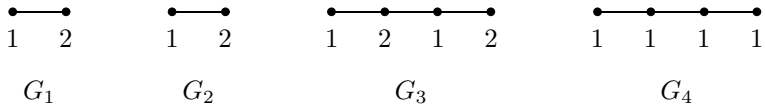

Fig. 1. Union and relabeling. $G_3 = \times_{\{(2,1)\}}(G_1, G_2)$ and $G_4 = \circ_{\{(1,1),(2,1)\}}(G_3)$.

We continue with a formal definition of graph-producing expressions based on the above operations. We call these expressions NLC_k-terms, and the graph produced by such a term D will be denoted $\mathrm{G}(D)$. We also define $\mathrm{L}(D)$ to be the set of all labels in graphs produced by subexpressions of D, including D itself.

Definition 3 (NLC_k-term) D is an NLC_k-term if it satisfies one of the following:

- D has the form $\lambda_i(x)$, where $i \in [k]$, and where x is either the name of a vertex, or a bullet symbol, $\bullet$, representing an unnamed vertex. In either case, $\mathrm{G}(D)$ is this vertex labeled with i, and $\mathrm{L}(D) = \{i\}$. (Each occurrence of a bullet symbol represents a vertex distinct from all other vertices.)
- D has the form $\times_S(D_1, D_2)$, where $S \subseteq [k]^2$, and where D_1 and D_2 are NLC_k-terms such that $\mathrm{G}(D_1)$ and $\mathrm{G}(D_2)$ are disjoint. Then $\mathrm{G}(D) = \times_S(\mathrm{G}(D_1), \mathrm{G}(D_2))$, and $\mathrm{L}(D) = \mathrm{L}(D_1) \cup \mathrm{L}(D_2)$.
- D has the form $\circ_R(D')$, where D' is an NLC_k-term, and where R is a mapping from $[k]$ to $[k]$. Then $\mathrm{G}(D) = \circ_R(\mathrm{G}(D'))$, and $\mathrm{L}(D) = \mathrm{L}(\mathrm{G}(D)) \cup \mathrm{L}(D')$.

Example 1 Let $D = \times_{\{(2,1)\}}\left(\times_{\{(1,2)\}}(\lambda_1(\bullet), \lambda_2(\bullet)), \times_{\{(1,2)\}}(\lambda_1(\bullet), \lambda_2(\bullet))\right)$. Then D is an NLC_2-term, and $\mathrm{L}(D) = \{1, 2\}$. D produces the graph G_3 in Fig. 1.

It is often convenient to view NLC_k-terms as rooted ordered binary trees. See Fig. 2.

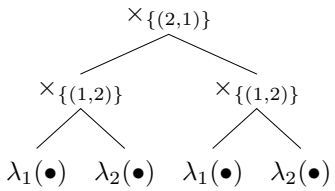

Fig. 2. The NLC_2-term in Example 1 expressed with a binary tree.

We are now prepared to define the class NLC_k, as well as what will be our most frequently used concepts.

Definition 4 (NLC$_k$ [15]) NLC$_k$ is the class of all (labeled) graphs produced by NLC$_k$-terms.

Definition 5 (NLC$_k$-decomposition, NLC-decomposition [10]) An NLC$_k$-decomposition of a graph G is an NLC$_k$-term D such that $G = \mathrm{G}(D)$ if G is labeled, and such that $G = \mathrm{unlab}(\mathrm{G}(D))$ if G is unlabeled. If D exists, G is said to be NLC$_k$-decomposable. An NLC-decomposition of G is an NLC$_k$-decomposition of G for some unspecified value of k. Note that G always is "NLC-decomposable".

Definition 6 (Width) The width of an NLC$_k$-term D is $|\mathrm{L}(D)|$.

Definition 7 (NLC-width [10]) The NLC-width of a graph G, $\mathrm{width}_{\mathrm{NLC}}(G)$, is the smallest width among all NLC-decompositions of G.

The NLC$_k$-decomposition problem. A graph G is given, unlabeled or labeled with numbers in $[k]$. The task is to find an NLC$_k$-decomposition of G, if that exists.

Example 2 The NLC$_2$-term D in Example 1 is an NLC$_2$-decomposition of G_3 in Fig. 1, as well as of its unlabeled variant called P_4, shown in Fig. 3. D has width 2. The reader is invited to show that there is no NLC-decomposition of P_4 with width 1. Thus P_4 has NLC-width 2.

Fig. 3. The graph P_4.

Finally, we reproduce some fundamental observations by Wanke.

Definition 8 (Restriction) Let D be an NLC$_k$-term, and let $V \subseteq \mathrm{V}(\mathrm{G}(D))$. Then $D|V$ denotes the restriction of D to V, the expression obtained by deleting the terms for vertices not in V, and removing superfluous operations in the obvious way. Evidently, $\mathrm{G}(D|V) = \mathrm{G}(D)|V$.

Since restricting a decomposition does not increase its width, we immediately have:

Lemma 1 *If H is an induced subgraph of G, then* $\mathrm{width}_{\mathrm{NLC}}(H) \leq \mathrm{width}_{\mathrm{NLC}}(G)$*, and H is* NLC$_k$*-decomposable if G is.*

Definition 9 (Complement) Let D be an NLC$_k$-term. Then $\overline{D}$ denotes the edge-complement of D, the expression obtained by exchanging each union operator $\times_S$ for $\times_{\overline{S}}$, where $\overline{S} = [k]^2 \setminus S$. Evidently, $\mathrm{G}(\overline{D}) = \overline{\mathrm{G}(D)}$.

Since $\overline{D}$ is an NLC$_k$-term if D is, we have:

Lemma 2 ([15]) *$G \in$ NLC$_k$ if and only if $\overline{G} \in$ NLC$_k$.*

4 Modular Decomposition

Modular decomposition has been defined a number of times for various kinds of structures. It is called substitution decomposition in [13, 14], where an abstract analysis is presented and applied to relations (such as graphs), set systems, and boolean functions. The kind of generalized graphs called 2-structures [8] are also well-suited for modular decomposition, as shown in [7, 8, 9, 12]. The reader is referred to [7, 12, 13, 14] for further references.

In this section we define modular decomposition for labeled graphs. We indicate its connection with NLC-decomposition, and we formulate some properties which we will use later. Finally, we describe how the modular decomposition of a labeled graph can be computed with an existing algorithm for modular decomposition of 2-structures.

4.1 Substitution

In general, the modular decomposition of a structure S is a derivation of S with the implicit or explicit help of a substitution operation. Let us first look at how substitution normally works for unlabeled graphs, and how we can extend the operation to produce labeled graphs as well.

Definition 10 (Substitution of graphs) Let G' be a graph and let the graphs $G_v, v \in \mathrm{V}(G')$, be unlabeled and disjoint.

- If G' is unlabeled, $G'[G_v, v \in \mathrm{V}(G')]$ is defined as the unlabeled graph obtained from the disjoint union of $G_v, v \in \mathrm{V}(G')$, by adding, for each edge $\{u, v\}$ in G', all possible edges between $\mathrm{V}(G_u)$ and $\mathrm{V}(G_v)$. See Fig. 4.
- If G' is labeled, $G'[G_v, v \in \mathrm{V}(G')]$ is defined as the labeled graph G obtained by proceeding first as in the unlabeled case, and then assigning the labels, so that for each vertex v in G', we have $lab_G(u) = lab_{G'}(v)$ for all $u \in \mathrm{V}(G_v)$. See Fig. 4.

Associated with the composition $G = G'[G_v, v \in \mathrm{V}(G')]$ is a natural mapping from $\mathrm{V}(G)$ to $\mathrm{V}(G')$: If $X \subseteq \mathrm{V}(G)$, the *image* of X in $\mathrm{V}(G')$, denoted $\mathrm{Im}(X)$, is the set $\{v : v \in \mathrm{V}(G') \text{ and } \mathrm{V}(G_v) \cap X \neq \emptyset\}$. And if $Y \subseteq \mathrm{V}(G')$, the inverse image of Y in $\mathrm{V}(G)$, denoted $\mathrm{Im}^{-1}(Y)$, is the set $\bigcup_{v \in Y} \mathrm{V}(G_v)$. See Fig. 4.

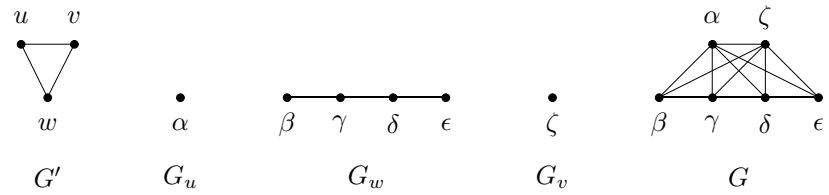

Fig. 4. Graph substitution. $G = G'[G_u, G_v, G_w]$. As G' is unlabeled, so is G. If G' were labeled, let us say with 1 for u and v, and 2 for w, then G would be labeled too, with 1 for α and ζ, and 2 for β, γ, δ, and ϵ. The image of β in G' is $\{w\}$. The inverse image of w in G is $\{\beta, \gamma, \delta, \epsilon\}$.

The importance of the substitution operation in connection with NLC-decomposition is evident from the following:

Proposition 1 *Suppose that $G = G'[G_v, v \in \mathrm{V}(G')]$. Then the NLC-width of G equals the highest NLC-width among G' and $G_v, v \in \mathrm{V}(G')$, and an NLC-decomposition of G with this width can be obtained from NLC-decompositions of G' and $G_v, v \in \mathrm{V}(G')$.*

Proof. Let k be the highest NLC-width among G' and $G_v, v \in \mathrm{V}(G')$. Without loss of generality, we assume that if G is labeled, the labels belong to $[k]$. Let

D' be an NLC_k-decomposition of G', and for each $v \in \mathrm{V}(G')$, let l_v be the initial label of v in this decomposition. Finally, for each $v \in \mathrm{V}(G')$, let D_v be an NLC_k-decomposition of G_v. Then we get an NLC_k-decomposition of $G'[G_v, v \in \mathrm{V}(G')]$ by replacing in D', for each vertex $v \in \mathrm{V}(G')$, the (innermost) term for v by $\circ_{R_v}(D_v)$, where R_v maps each element of $[k]$ to l_v. It follows that the NLC-width of G is at most k. See Fig. 5.

On the other hand, G' and $G_v, v \in \mathrm{V}(G')$, are all induced subgraphs of $G = G'[G_v, v \in \mathrm{V}(G')]$. True, this statement is a bit informal. For G' we should actually speak of an isomorphism. And $G_v, v \in \mathrm{V}(G')$, are all unlabeled, whereas the corresponding induced subgraphs of G may be uniformly labeled. Nevertheless, by Lemma 1 it is clear that the NLC-width of G cannot be less than k. □

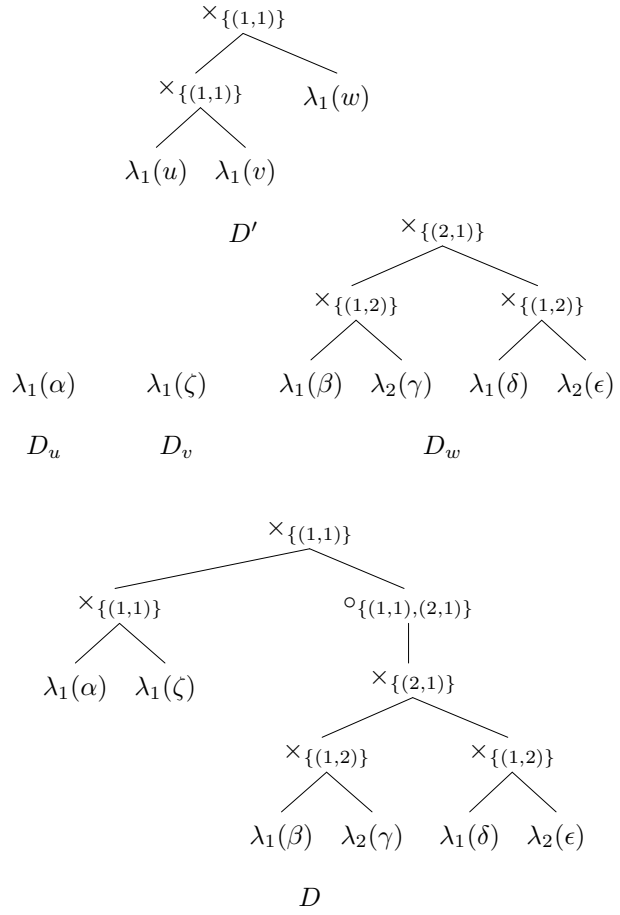

Fig. 5. Substitution of decompositions. How an NLC-decomposition D of $G = G'[G_u, G_v, G_w]$ (see Fig. 4) can be obtained from NLC-decompositions D', D_u, D_v, and D_w of G', G_u, G_v, and G_w.

4.2 Modules

Let G be a labeled or unlabeled graph. If it can be written as $G'[G_v, v \in \mathrm{V}(G')]$ the partition $\pi = \{\mathrm{V}(G_v), v \in \mathrm{V}(G')\}$ of the vertices of G is called a *congruence partition* of G. Although not unique, the graph G' is often called the *quotient* of G modulo π.

We shall now define what is meant by a *module* of G. First, if $M \subseteq \mathrm{V}(G)$ is a class of some congruence partition of G, then M is a module of G. Second, $M = \mathrm{V}(G)$ is always a module of G. Essentially, this is the approach in [13] (where modules are called autonomous sets though), adjusted for the fact that $\{\mathrm{V}(G)\}$ is not necessarily a congruence partition of G when G is labeled. It follows from Definition 10 that if G is labeled, $M \subseteq \mathrm{V}(G)$ is a module of G if and only if

(i) M is nonempty;
(ii) for each vertex $v \in \mathrm{V}(G) - M$, v has edges, either to all vertices in M, or to none of them; and
(iii) either M is uniformly labeled, or $M = \mathrm{V}(G)$;

whereas if G is unlabeled, $M \subseteq \mathrm{V}(G)$ is a module of G if and only if (i) and (ii) hold.

Example 3 The modules of the graph G in Fig. 4 are $\{\alpha\}$, $\{\beta\}$, $\{\gamma\}$, $\{\delta\}$, $\{\epsilon\}$, $\{\zeta\}$, $\{\alpha, \zeta\}$, $\{\beta, \gamma, \delta, \epsilon\}$, $\{\alpha, \beta, \gamma, \delta, \epsilon\}$, $\{\beta, \gamma, \delta, \epsilon, \zeta\}$, and $\{\alpha, \beta, \gamma, \delta, \epsilon, \zeta\}$.

We denote the set of modules of G by $\mathrm{M}(G)$. Recall that for a set of vertices V in a graph G, $G|V$ denotes the subgraph of G induced by V. Let us also define that two sets A and B *overlap* if $A \setminus B$, $A \cap B$, and $B \setminus A$ are all nonempty. It is now a straightforward task to check that module properties (A1) through (A4) in [13], restated below for a graph G, apply not only if G is unlabeled, but also if it is labeled.

(A1) $\mathrm{V}(G) \in \mathrm{M}(G)$, and $\{v\} \in \mathrm{M}(G)$ for each $v \in \mathrm{V}(G)$. (These are the *trivial modules* of G. If G has no other modules, it is called *prime*.)
(A2) If $A, B \in \mathrm{M}(G)$ overlap, then $A \setminus B$, $A \cap B$, $B \setminus A$, and $A \cup B$ are also modules of G.
(A3) If $A \in \mathrm{M}(G)$, then $\mathrm{M}(G|A) = \{V \in \mathrm{M}(G) : V \subseteq A\}$.
(A4) For $G = G'[G_v, v \in \mathrm{V}(G')]$ we have:
- If $X \in \mathrm{M}(G)$, then $\mathrm{Im}(X) \in \mathrm{M}(G')$.
- If $Y \in \mathrm{M}(G')$, then $\mathrm{Im}^{-1}(Y) \in \mathrm{M}(G)$.

4.3 Strong Modules and the Decomposition Tree

It should be no surprise that given a graph G, any partition π of its vertices into two or more modules is a congruence partition of G, that is, π corresponds to a substitution composition $G = G'[G_v, v \in \mathrm{V}(G')]$. Of course, this holds for the graphs $G_v, v \in \mathrm{V}(G')$, as well. Consequently, by recursively partitioning modules into smaller modules, we can find a derivation of G based on the substitution composition.

It was shown in [13] that for any structure S whose modules satisfy (A1) through

(A4) (in their general forms), there is one recursive partitioning — the modular decomposition of S — which has particularly nice properties. It is defined there in terms of two decomposition principles. Here we shall instead base the definition on the characterization in [12]. A module of a graph G is called *strong* if it does not overlap any other module. The *strong module tree* of G, $\mathrm{T}_{\mathrm{SM}}(G)$, is defined as follows: The nodes of $\mathrm{T}_{\mathrm{SM}}(G)$ are the strong modules of G. The root is $\mathrm{V}(G)$, and a node M_1 is a descendent of another node M_2 if and only if $M_1 \subset M_2$. Consequently, the leaves of $\mathrm{T}_{\mathrm{SM}}(G)$ are the singleton subsets of $\mathrm{V}(G)$. One can notice that every module of G is a union of siblings in $\mathrm{T}_{\mathrm{SM}}(G)$. See Fig. 6.

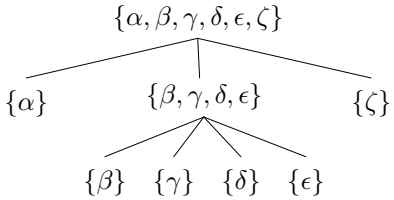

Fig. 6. The strong module tree of the graph G in Fig. 4.

The strong module tree of a graph G recursively partitions the vertices of G. By inspection of Proposition 3.1 and Theorem 3.5 in [13], it is clear that the modular decomposition, as defined in that article, exactly corresponds to the partitioning given by the strong module tree. (To see this, and to appreciate it, one must keep (A3) in mind.) Thus, with the definition below, we are actually following [13].

Definition 11 (Modular decomposition) The modular decomposition of a labeled or unlabeled graph G is that recursive derivation of G (using the substitution operation in Definition 10) which corresponds to the strong module tree of G. We denote it $\mathrm{D}_{\mathrm{M}}(G)$. See Fig. 7.

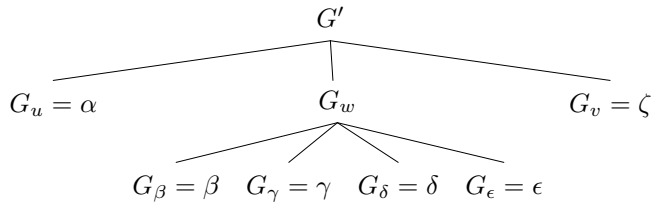

Fig. 7. The modular decomposition of the graph G in Fig. 4, described with a tree. Internal nodes correspond to substitution operations and are marked with their quotient graphs. Leaf nodes designate single-vertex graphs. Compare with Fig. 6.

It is now easy to formulate the properties of the modular decomposition of a graph G. By the results in [13], every internal node M of $\mathrm{T}_{\mathrm{SM}}(G)$ is one of the following:

- *Degenerate.* Every union of children of M is a module of M.
- *Linear.* There is a linear order on the children of M, such that the union of some children is a module of M if and only if these children are consecutive

with respect to this order.

- *Prime.* The only proper union of children of M which is a module of M is M itself.

Of course, a node with just two children will satisfy all these cases. But if M has three or more children — a situation which we call "proper" (or "properly ...") — then exactly one case will apply.

Example 4 In the strong module tree in Fig. 6, the top node, $\{\alpha, \beta, \gamma, \delta, \epsilon, \zeta\}$, is degenerate, whereas its child $\{\beta, \gamma, \delta, \epsilon\}$ is prime. Compare with Example 3.

By (A4) there is a completely analogous characterization of the quotient graph Q associated with the partition of M into its children, that is, satisfying $G|M = Q[(G|M_v), v \in \mathrm{V}(Q)]$, where $M_v, v \in \mathrm{V}(Q)$, are the children of M. Thus, Q is either degenerate, linear, or prime, and the meaning of this is given by the definitions above, when "M" is replaced by "Q", and "children" is replaced by "vertices".

Example 5 In the modular decomposition in Fig. 7, G' is degenerate, whereas G_w is prime.

The linear and degenerate cases can be characterized further. It is not hard to see that if an unlabeled undirected graph G is *semi-linear*, meaning that there is a linear order on the vertices of G such that any set of consecutive vertices form a module of G, then G is either *complete* (having all possible edges) or *discrete* (having no edges at all). Thus, G is in fact degenerate. The proper linear case appears only in directed graphs.

The introduction of vertex labels does not change any of this. For if a labeled graph is properly semi-linear, clearly it must be uniformly labeled.

So a quotient graph Q in a modular decomposition satisfies one of the following:

- Q has two vertices.
- Q is properly degenerate, implying that it is complete or discrete, and either unlabeled or uniformly labeled.
- Q is properly prime.

We finish our discussion about the properties of modular decomposition with a lemma that we will need for the analysis of the NLC_2-decomposition algorithm.

Lemma 3 *The total number of vertices in the quotient graphs of the modular decomposition of a graph G is bounded by $2|\mathrm{V}(G)|$.*

Proof. Let us view the modular decomposition of G, $\mathrm{D_M}(G)$, as a tree, T. The leaf nodes of T correspond to the vertices of G, and the nonleaf nodes correspond to the quotient graphs in $\mathrm{D_M}(G)$. Each nonleaf node has as many children as there are vertices in its quotient graph. Thus, the total number of vertices in the quotient graphs equals the number of nodes in T, minus one corresponding to the root. Clearly, this is less than twice the number of leaf nodes. □

4.4 Computing the Modular Decomposition

We are now going to show that the modular decomposition of a labeled or unlabeled graph G can be computed in $O(n^2)$ time. To avoid any lengthy discussion,

we shall simply make use of an existing algorithm for modular decomposition of 2-structures. As shown in [9], the latter can express a wide variety of graphs.

For a set V, a 2-edge over V is an ordered pair (u, v), where $u, v \in V$ and $u \neq v$. $\mathrm{E}_2(V)$ denotes the set of all 2-edges over V. A 2-structure $S = (V, R)$ is the set V (usually called the domain of S) and an equivalence relation R on $\mathrm{E}_2(V)$. It is sometimes convenient to express R with a labeling function lab_S on $\mathrm{E}_2(V)$ such that $e_1 R e_2$ if and only if $lab_S(e_1) = lab_S(e_2)$. We then write $S = (V, lab_S)$.

When relations are represented by labeling functions, one can define substitution of 2-structures in the same way that substitution is defined for graphs. And as for graphs, there is an accompanying module concept. A module of a 2-structure $S = (V, R)$ is a set $M \subseteq V$ such that for all $x, y \in M$ and all $z \in V - M$, we have $(x, z)R(y, z)$ and $(z, x)R(z, y)$. In contrast to [8, 9] (where modules are called clans) and [12], we do not consider the empty set to be a module. We denote all modules of S by $\mathrm{M}(S)$. It can be seen in [8] that analogues of (A1) through (A4) are valid for 2-structures. This implies the existence of modular decomposition, in the sense we already know it.

It is easy to express an unlabeled graph $G = (V, E)$ as a 2-structure $S = (V, lab_S)$ by defining, for each $\{u, v\}$, $lab_S(u, v) = lab_S(v, u) = 1$ if $\{u, v\} \in E$, and $lab_S(u, v) = lab_S(v, u) = 0$ otherwise. What is worth noticing is that G and S have the same modules. Of course, this means that they also have the same strong modules and strong module trees, and that we can get the modular decomposition of G by computing the modular decomposition of S.

This approach can be used for labeled graphs also. Given $G = (V, E, lab_G)$, we construct the 2-structure $S = (V, lab_S)$ by defining, for each 2-edge (u, v), $lab_S(u, v) = (1, lab_G(u), lab_G(v))$ if $\{u, v\} \in E$, and $lab_S(u, v) = (0, lab_G(u), lab_G(v))$ otherwise. It is not difficult to see that G and S have the same modules.

Thus we can get the modular decomposition of a graph G, labeled or not, by computing the modular decomposition of a derived 2-structure $S = (V, lab_S)$, where $V = \mathrm{V}(G)$. With the algorithm in [12], this takes $O(|\mathrm{V}(G)|^2)$ time and space.

5 NLC_2-Decomposition in Polynomial Time

In this section we solve the NLC_k-decomposition problem for $k = 2$. But before we restrict our choice of k, let us draw the full conclusion of our previous discussion about modular decomposition. By applying Proposition 1 to the modular decomposition of a graph, we find:

Proposition 2 *Let G be a graph with more than one vertex. Then the NLC-width of G equals the highest NLC-width among the quotient graphs in the modular decomposition of G, and an NLC-decomposition of G with this width can be obtained from NLC-decompositions of these quotient graphs.*

So let Q be a quotient graph in the modular decomposition of a graph G. Q may be labeled or unlabeled. (If Q is the top-level quotient, then Q is labeled if G is. Otherwise, Q is unlabeled.) If Q has two vertices, its NLC-width is at most

2, and if Q is properly degenerate, its NLC-width is 1. In each of these cases, it is trivial to find an NLC-decomposition of Q with minimal width in linear time.

From here on, we study the remaining case — Q is properly prime — and we restrict our discussion to NLC_2-decomposition. Thus, Q will be unlabeled or labeled with numbers in $\{1, 2\}$.

Assume that there exists an NLC_2-decomposition D of Q. We shall look at the presence of relabeling operations in D, and argue that none is needed, except for one at the outermost level if Q is uniformly labeled. Consider the relabeling operator, $\circ_R$. In an NLC_2-decomposition, the mapping R may be one of the following:

(i) $\{(1,1),(2,2)\}$; (ii) $\{(1,2),(2,1)\}$; (iii) $\{(1,1),(2,1)\}$; or (iv) $\{(1,2),(2,2)\}$.

We may rule out relabelings using mapping (i), since they do not do anything. We may also rule out those using mapping (ii), since for this mapping, a subexpression of D on the form $\circ_R(D')$ can always be replaced by D'', obtained from D' by changing each 1 to a 2 and vice versa. (This turns each (iii)-relabeling in D' into a (iv)-relabeling in D'', and (iv)-relabelings in D' become (iii)-relabelings in D'', but no other relabelings are affected.) Finally, we may rule out relabelings using mappings (iii) and (iv), except possibly for the outermost operation of D. For naturally, we do not need to relabel single vertices, since we can give them any label to begin with. And if a subexpression $\circ_R(D')$ of D produces a graph with two or more vertices, but fewer than $Q = \mathrm{G}(D)$ has, then R may not be the mapping (iii) or (iv). This follows from Lemma 4 below, since Q is assumed to be prime.

Lemma 4 *Let D be an NLC-decomposition of a graph G. If a subexpression of D produces a uniformly labeled graph, then this graph is a module of G.*

We summarize:

Proposition 3 *Let the properly prime graph Q be NLC_2-decomposable.*

- *If Q is nonuniformly labeled (meaning that $\mathrm{L}(Q) = \{1, 2\}$), then it has a relabeling-free NLC_2-decomposition.*
- *If Q is unlabeled, it likewise has a relabeling-free NLC_2-decomposition, producing a nonuniformly labeled version of Q.*
- *If Q is uniformly labeled, then it has an NLC_2-decomposition on the form $\circ_R(D')$, where D' is relabeling-free, and where R is the mapping (iii) or (iv) above.*

This leads immediately to the procedure we will use:

Algorithm for properly prime graphs. To NLC_2-decompose, if possible, a properly prime graph Q, we shall do as follows:

- If Q is nonuniformly labeled, then we use Algorithm 1 (below), which searches for a relabeling-free NLC_2-decomposition of Q.
- If Q is unlabeled, then we use Algorithm 2. It searches for a nonuniform labeling of Q that permits a relabeling-free NLC_2-decomposition, as determined by Algorithm 1.
- If Q is uniformly labeled, then we use Algorithm 2 to search for an NLC_2-decomposition D' of $\mathrm{unlab}(Q)$. If we find D', we can easily construct an NLC_2-decomposition $\circ_R(D')$ of Q.

We now turn to the details of Algorithms 1 and 2. Algorithm 1 is rather simple, but Algorithm 2 is structured in cases of iterations of stages involving more iterations and cases, and the reader is warned that the motivations become quite long. After that, we finish with a concluding analysis, which also serves as a brief summary of the whole decomposition process.

5.1 Algorithm 1

The input to this algorithm is a graph G labeled with numbers in $\{1, 2\}$. The output is a relabeling-free NLC_2-decomposition D of G, if such a decomposition exists.

The algorithm constructs the decomposition in a top-down fashion, by successively partitioning the vertices of G. We know that if D exists, it has the form $\times_S (D_1, D_2)$, where S is one of the 16 subsets of $\{(1,1),(1,2),(2,1),(2,2)\}$. More interesting, as soon as we find a partition $\{V_1, V_2\}$ of the vertices of G such that $G = \times_S (G_1, G_2)$ for $G_1 = G|V_1$, $G_2 = G|V_2$, and S among the 16 possibilities above, we know that G has a relabeling-free NLC_2-decomposition if and only if G_1 and G_2 do. For if D is a relabeling-free NLC_2-decomposition of G, then the restrictions $D|V_1$ and $D|V_2$ are relabeling-free NLC_2-decompositions of G_1 and G_2. And conversely, if D_1 and D_2 are relabeling-free NLC_2-decompositions of G_1 and G_2, then $\times_S (D_1, D_2)$ is a relabeling-free NLC_2-decomposition of G.

To find, if possible, a partition $\{V_1, V_2\}$ such that $G = \times_S (G_1, G_2)$, where $G_1 = G|V_1$ and $G_2 = G|V_2$, we are going to try each S, if needed. But first, we select any vertex u in G and specify that u shall belong to V_2. This will be no restriction, since $\times_S (G_1, G_2) = \times_{S'} (G_2, G_1)$, where S' is obtained by reversing each pair in S. We now try each relation S as follows: First, we let $V_2 = \{u\}$. For each vertex v in $V_1 = \mathrm{V}(G) \setminus u$, we check if u has an edge to v if and only if it should, according to S. If not, we move v from V_1 to V_2. Each time we move a vertex to V_2, we check this vertex with respect to those left in V_1; we compare with S, and move more vertices if needed. Continuing like this, we end up either with a valid partition, or with an empty V_1.

If a partition is found, we continue to partition V_1 and V_2, and so on, until all obtained sets have size one. This means $n - 1$ partitions all in all (where $n = |\mathrm{V}(G)|$). Each partition step can be carried out in $O(n^2)$ time, so the total time for Algorithm 1 is $O(n^3)$. Besides the input, only $O(n)$ space is needed.

5.2 Algorithm 2

The input to this algorithm is a properly prime unlabeled graph G. The output is a relabeling-free NLC_2-decomposition D of G, if such a decomposition exists.

As before, if D exists, it has the form $\times_S (D_1, D_2)$, where S is a subset of $\{(1,1),(1,2),(2,1),(2,2)\}$. However, G is now unlabeled, and the number of interesting possibilities for S is then smaller. Firstly, D produces a labeled graph such that $G = \mathrm{unlab}(\mathrm{G}(D))$. Our freedom in choosing the labeling makes many possibilities for S equivalent. For example, if $\times_S (D_1, D_2)$ is an NLC_2-decomposition of G,

so is $\times_{S'}(D_1, D_2')$, where S' is obtained by changing each 1 to a 2 and vice versa in the second position of each pair in S, and where D_2' is likewise obtained from D_2 by switching all 1s and 2s. Secondly, many values of S would make G contain modules. These values can be excluded.

We shall now characterize each subset S of $\{(1,1),(1,2),(2,1),(2,2)\}$ with respect to the expression $G = \text{unlab}(\times_S(G_1, G_2))$. Let V_1 and V_2 denote $\text{V}(G_1)$ and $\text{V}(G_2)$ respectively. We have to observe that one of these may be a single vertex. We will treat that case later. Under the assumption that both G_1 and G_2 have several vertices, the feasibility of the subsets S of $\{(1,1),(1,2),(2,1),(2,2)\}$ is as follows:

- $\emptyset$ is not possible. Both V_1 and V_2 would be modules of G.
- $\{(1,1)\}$, $\{(1,2)\}$, $\{(2,1)\}$, and $\{(2,2)\}$ are possible and equivalent.
- $\{(1,1),(1,2)\}$ and $\{(2,1),(2,2)\}$ would make V_2 a module of G. And $\{(1,1),(2,1)\}$ and $\{(1,2),(2,2)\}$ would make V_1 a module of G. All four are thus impossible.
- $\{(1,1),(2,2)\}$ and $\{(1,2),(2,1)\}$ are possible and equivalent.
- There are four possible and equivalent subsets containing three pairs.
- $\{(1,1),(1,2),(2,1),(2,2)\}$ is impossible, just as $\emptyset$ is.

Thus, if both G_1 and G_2 are to have several vertices, there are three cases that we have to try for S. We represent them as $\{(1,1)\}$, $\{(1,1),(2,2)\}$, and $\{(1,2),(2,1),(2,2)\}$. Of course, both G_1 and G_2 must be nonuniformly labeled — otherwise we again have a forbidden module of G.

There remains the case that one of G_1 and G_2, let us say the former, has only one vertex. We may then assume that this vertex is labeled with 1. The alternatives for S that we have to consider are then $\emptyset$, $\{(1,1)\}$, $\{(1,2)\}$, and $\{(1,1),(1,2)\}$. As before, $\emptyset$ and $\{(1,1),(1,2)\}$ would make V_2 a module of G, whereas $\{(1,1)\}$ and $\{(1,2)\}$ are possible and equivalent. Thus we can cover the case that one of G_1 and G_2 has just one vertex by trying $S = \{(1,1)\}$.

Below, we describe how to search for an NLC_2-decomposition D of G on the form $\times_S(D_1, D_2)$, where S is $\{(1,1)\}$, $\{(1,1),(2,2)\}$, or $\{(1,2),(2,1),(2,2)\}$.

5.2.1 The case $S = \{(1,1)\}$

Let $S = \{(1,1)\}$. To find a decomposition of G on the form $\times_S(D_1, D_2)$, we may go through all edges $\{v_1, v_2\}$ in G, and determine for each the satisfiability of $G = \text{unlab}(\times_S(G_1, G_2))$, where G_1 and G_2 are required to be NLC_2-decomposable and to contain v_1 and v_2 respectively, both of which must then be labeled with 1. We will later on develop this idea a little further, in order to reduce the number of edges $\{v_1, v_2\}$ that we have to go through.

We assume from now on that v_1 and v_2 have been fixed like this. The fact that $S = \{(1,1)\}$ then implies that as soon as we place any other vertex v in G_1 or G_2, we know what its label must be. For example, if v is placed in G_1, its label must be 1 if it has an edge to v_2, and 2 otherwise. Therefore, given a subset V of $\text{V}(G)$ containing v_1 possibly, but not v_2, let $\text{G}_{\mathit{left}}(V)$ denote the graph on V whose edges

are induced by G, and in which a vertex is labeled with 1 if it has an edge to v_2, and 2 otherwise. Similarly, given a subset V of $\mathrm{V}(G)$ containing v_2 possibly, but not v_1, let $\mathrm{G}_{right}(V)$ denote the graph on V whose edges are induced by G, and in which a vertex is labeled with 1 if it has an edge to v_1, and 2 otherwise.

The fixation of v_1 and v_2 not only helps us to label the vertices in $V^* = \mathrm{V}(G) \setminus \{v_1, v_2\}$ once they have been placed in G_1 or G_2, but it also creates a useful dependency among these vertices with respect to their placement. For $i, j \in \{1, 2\}$, let an i–j-vertex — a vertex of *type* i–j — be a vertex in V^* which will be labeled with i if placed in G_1, and with j if placed in G_2. Notice that each vertex in V^* is an i–j-vertex for some i and j. As an example of the dependency, let us look at two 1–1-vertices u and v. If there is no edge between u and v, then they must be placed together, either in G_1 or in G_2, since $\times_S (G_1, G_2)$ produces edges between 1-labeled vertices in G_1 and 1-labeled vertices in G_2.

We use a directed graph, G_{dep}, to reflect this dependency. G_{dep} is unlabeled, has vertex set V^*, and there is an edge from u to v in G_{dep}, also written $u \to v$, whenever the existence or not of an edge between u and v does not match S when u is placed in G_2 and v is placed in G_1. So if $u \to v$, then u cannot be placed in G_2 without v being placed there too. We let $u \leftrightarrow v$ mean that both $u \to v$ and $v \to u$ hold, and we let $u \mid v$ mean that neither $u \to v$ nor $v \to u$ holds. Finally, we define $\lesssim$ to be the reflexive and transitive closure of the relation $\to$.

A partition $\{V_1^*, V_2^*\}$ of V^* is said to *respect* $\lesssim$ if $u \lesssim v$ does not hold for any vertices $v \in V_1^*$ and $u \in V_2^*$. Notice that given a partition $\{V_1^*, V_2^*\}$ of V^* (where we allow one of V_1^* and V_2^* to be empty), $G = \mathrm{unlab}(\times_S (G_1, G_2))$ is true for $G_1 = \mathrm{G}_{left}(v_1 \cup V_1^*)$ and $G_2 = \mathrm{G}_{right}(v_2 \cup V_2^*)$ if and only if $\{V_1^*, V_2^*\}$ respects $\lesssim$. As soon as this is the case, we can use Algorithm 1 to search for NLC_2-decompositions D_1 and D_2 of G_1 and G_2. If they exist, $D = \times_S (D_1, D_2)$ is an NLC_2-decomposition of G, and $\{V_1^*, V_2^*\}$ is said to be a *successful* partition. If D_1 and D_2 do not both exist, we can try another partition of V^*. Below we show that if we choose these partitions carefully, we only need to try $O(\log(|\mathrm{V}(G)|))$ of them. If we have not found D after that, we can conclude that that we have to continue with a new fixation of v_1 and v_2.

To bound the number of partitions we have to consider, we first collect vertices into *clusters*. If C is a strongly connected component in G_{dep}, then all vertices of C must be placed together, either in G_1 or in G_2. We then say that C is a cluster of V^*. For clusters C_1 and C_2, we may write $C_1 \lesssim C_2$ if $u \lesssim v$ for some $u \in C_1$ and $v \in C_2$. However, unless stated otherwise, clusters will be assumed distinct, and we will write $C_1 < C_2$ instead of $C_1 \lesssim C_2$. (To have both $C_1 \lesssim C_2$ and $C_2 \lesssim C_1$ is then not possible.) If neither $C_1 < C_2$ nor $C_2 < C_1$ holds, we write $C_1 \parallel C_2$.

In agreement with previous notation, we also write $C_1 \to C_2$ if $u \to v$ for some $u \in C_1$ and $v \in C_2$, and we write $C_1 \mid C_2$ if neither $C_1 \to C_2$ nor $C_2 \to C_1$ holds. Of course, we never have "$C_1 \leftrightarrow C_2$". Note that $C_1 \to C_2$ implies $C_1 < C_2$. Conversely, $C_1 \parallel C_2$ implies $C_1 \mid C_2$.

We can get a deeper understanding of clusters by looking at $\to$ for specific pairs of vertex types:

- Let u be a 1–1-vertex and v a 1–2-vertex. If there is an edge (in G) between u and v, then $v \to u$. (Since $(1,2) \notin S$, we cannot place v in G_2 and u in G_1.) On the other hand, if there is no edge between u and v, then $u \to v$. (Since $(1,1) \in S$, we cannot place u in G_2 and v in G_1.)
- Let u be a 1–1-vertex and v a 2–1-vertex. If there is an edge between u and v, then $u \to v$. On the other hand, if there is no edge between u and v, then $v \to u$.
- Let u be a 1–2-vertex and v a 2–1-vertex. If there is an edge between u and v, then $u \to v$. If there is no edge between u and v, then $v \to u$.

As a consequence, if C_1 and C_2 are two different clusters, one containing a 1–1-vertex and the other a 1–2-vertex, or one containing a 1–1-vertex and the other a 2–1-vertex, or one containing a 1–2-vertex and the other a 2–1-vertex, then we have either $C_1 < C_2$ or $C_2 < C_1$.

We should also look at 2–2-vertices. Let u be a 2–2-vertex and v any other vertex in V^*. If there is an edge between u and v, then $u \leftrightarrow v$, so u and v must belong to the same cluster. If there is no edge between u and v, then $u \mid v$.

Using the first observation in the previous paragraph, we can show that no cluster may consist of only 2–2-vertices: Since G is properly prime, it is connected. Therefore, from a 2–2-vertex u, there is a path (in G) to the fixed vertex v_1 for example. Let v be the first vertex on this path which is not a 2–2-vertex. Certainly v exists and belongs to V^*, for a 2–2-vertex cannot have an edge to either v_1 or v_2. By the previous paragraph, all vertices from u to v belong to the same cluster.

Three pairs of vertex types remain. Let us come to them via a quick backward look. We found above that if u is a 2–2-vertex and v any other vertex in V^*, then either $u \leftrightarrow v$ or $u \mid v$ — in other words, if u and v are in different clusters, then $u \mid v$. It is not hard to see that the same is true if u and v are both 1–1-vertices, both 1–2-vertices, or both 2–1-vertices.

To summarize our findings, we call the vertex types 1–1, 1–2, and 2–1, *determining*. We have:

- Each cluster contains one or more vertices of at least one determining type.
- If t is a determining type in a cluster C_1, and a cluster C_2 contains a vertex of some other determining type, then $C_1 \to C_2$ or $C_2 \to C_1$.
- If two clusters, C_1 and C_2, contain exactly one and the same determining type, then $C_1 \mid C_2$.

The most interesting thing comes next. Let C_1 and C_2 be clusters satisfying $C_1 \parallel C_2$ — let t be their only determining type — and let C be another cluster. Suppose that $C < C_1$. Then there is a cluster C', identical to C possibly (but distinct from C_1), such that $C \lesssim C' \to C_1$. We can conclude that C' contains a determining type $t' \neq t$, and that $C' < C_2$ or $C_2 < C'$. The latter would imply that $C_2 < C_1$, though, contradicting our initial assumptions. So, it follows that $C < C_2$. Analogously, we find that if $C_1 < C$, then $C_2 < C$ also.

It is now easy to see that we can group (in a unique way) clusters into *boxes*, so that we satisfy the following *box structure properties*:

- There is a linear order, $<$, on the boxes.
- Each box contains at least one cluster.
- If B_1 and B_2 are boxes with $B_1 < B_2$, then $C_1 < C_2$ for any clusters $C_1 \in B_1$ and $C_2 \in B_2$.
- If C_1 and C_2 are clusters in the same box, then $C_1 \parallel C_2$.

We define boxes like this, and for simplicity, we let each box denote the union of its clusters. We can observe that a partition $\{V_1^*, V_2^*\}$ of V^* respects $\lesssim$ if and only if the following *monotonicity conditions* are satisfied:

- When V_1^* contains a box B_1, it also contains each box $B < B_1$.
- When V_2^* contains a box B_2, it also contains each box B such that $B_2 < B$.
- At most one box is *split* by the partition — that is, has some clusters in V_1^* and some in V_2^*.

Thereby we are ready to discuss the partitioning procedure. We will use a somewhat informal language — the boxes are assumed to be ordered from left to right, so that if $B_1 < B_2$, we can formulate this as "B_1 is to the left of B_2".

We first try to partition in between boxes. We describe this by extending the total order to include *separator* elements between the boxes, and at the ends. Given a separator s, we partition V^* as $\{V_1^*, V_2^*\}$, where V_1^* is the union of all boxes to the left of s, and V_2^* is the union of all boxes to the right of s. As described previously, we then define $G_1 = \mathrm{G}_{left}(v_1 \cup V_1^*)$ and $G_2 = \mathrm{G}_{right}(v_2 \cup V_2^*)$. From what we already know about partitions respecting $\lesssim$, we note, with the help of Lemma 1:

- If G_1 is not NLC_2-decomposable, any successful partition $\{V_1', V_2'\}$, must satisfy $V_1' \subset V_1^*$.
- If G_2 is not NLC_2-decomposable, any successful partition $\{V_1', V_2'\}$, must satisfy $V_2' \subset V_2^*$.

We can therefore use binary search among separators with one of the following results:

- We find a successful partition.
- We find a partition such that neither G_1 nor G_2 is NLC_2-decomposable. We can conclude that there is no successful partition for the current fixation of v_1 and v_2.
- We find separators s_l and s_r immediately to the left and to the right of some box, B, such that, when s_l is used, G_1 is NLC_2-decomposable but G_2 is not, and when s_r is used, G_2 is NLC_2-decomposable but G_1 is not. We can conclude that if there exists a successful partition, it must split B.

In the last case, we must examine B more closely. As we shall see, we only need to try one more partition, and we can find it as follows: First, for each cluster C in B, we use Algorithm 1 to search for NLC_2-decompositions of $\mathrm{G}_{left}(C)$ and $\mathrm{G}_{right}(C)$. If only one of these is decomposable, there is no doubt about in what part of a successful partition that C must be placed. (If neither $\mathrm{G}_{left}(C)$ nor $\mathrm{G}_{right}(C)$ is decomposable, the conclusion is of course simple.) We may now be left with a number of clusters for whose placement we have not yet seen any restrictions. Let us call them *remaining* clusters. Fortunately, all of them can safely be placed

together. It is the one determining type in B that matters: When B contains 1–2-vertices, the remaining clusters can be placed in V_1^*. When B contains 2–1-vertices, the remaining clusters can be placed in V_2^*. And when B contains 1–1-vertices, the remaining clusters can be placed anywhere. The detailed arguments are as follows:

Case 1. B consists of 1–2-vertices, and 2–2-vertices possibly. Let C be a cluster in B such that $\mathrm{G}_{left}(C)$ and $\mathrm{G}_{right}(C)$ are both NLC_2-decomposable, and let $\{V_1^*, V_2^*\}$ be a successful partition of V^* in which $C \subseteq V_2^*$. Thus, V_1^* contains all boxes to the left of B, and V_2^* contains all boxes to the right of B, so $\{V_1^* \cup C, V_2^* \setminus C\}$ respects $\lesssim$. We now show that this partition also is successful. Since $\{V_1^*, V_2^*\}$ is successful, $G_1 = \mathrm{G}_{left}(v_1 \cup V_1^*)$ and $G_2 = \mathrm{G}_{right}(v_2 \cup V_2^*)$ are both NLC_2-decomposable. By Lemma 1, so is $G_2' = \mathrm{G}_{right}(v_2 \cup V_2^* \setminus C)$. It remains to show that $G_1' = \mathrm{G}_{left}(v_1 \cup V_1^* \cup C)$ is NLC_2-decomposable. But $\mathrm{G}_{right}(C)$ has all vertices labeled with 2, so there are no edges from C to $v_1 \cup V_1^*$, and we have $G_1' = \times_\emptyset (G_1, \mathrm{G}_{left}(C))$. It follows that $\{V_1^* \cup C, V_2^* \setminus C\}$ is successful.

Case 2. B consists of 2–1-vertices, and 2–2-vertices possibly. Let C be a cluster in B such that $\mathrm{G}_{left}(C)$ and $\mathrm{G}_{right}(C)$ are both NLC_2-decomposable, and let $\{V_1^*, V_2^*\}$ be a successful partition of V^* in which $C \subseteq V_1^*$. This situation is symmetric to that in the previous case. It follows that $\{V_1^* \setminus C, V_2^* \cup C\}$ is a successful partition of V^*.

Case 3. B consists of 1–1-vertices, and 2–2-vertices possibly. Let C be a cluster in B. In this case, $\mathrm{G}_{left}(C)$ and $\mathrm{G}_{right}(C)$ are identical. Let them be NLC_2-decomposable, and let $\{V_1^*, V_2^*\}$ be a successful partition of V^* in which $C \subseteq V_2^*$. As before, the assumptions imply that $\{V_1^* \cup C, V_2^* \setminus C\}$ respects $\lesssim$. We now show that it also is successful. Since $\{V_1^*, V_2^*\}$ is successful, $G_1 = \mathrm{G}_{left}(v_1 \cup V_1^*)$ and $G_2 = \mathrm{G}_{right}(v_2 \cup V_2^*)$ are both NLC_2-decomposable. By Lemma 1, so is $G_2' = \mathrm{G}_{right}(v_2 \cup V_2^* \setminus C)$. It remains to show that $G_1' = \mathrm{G}_{left}(v_1 \cup V_1^* \cup C)$ is NLC_2-decomposable. But $\mathrm{G}_{left}(C) = \mathrm{G}_{right}(C)$, so $G_1' = \times_{\{(1,1)\}} (G_1, \mathrm{G}_{left}(C))$. It follows that $\{V_1^* \cup C, V_2^* \setminus C\}$ is successful. By symmetry, it conversely follows that if $\{V_1^*, V_2^*\}$ is a successful partition of V^* in which $C \subseteq V_1^*$, then $\{V_1^* \setminus C, V_2^* \cup C\}$ is successful too.

Let us now summarize: To determine the satisfiability of $G = \mathrm{unlab}(\times_S (G_1, G_2))$, where $S = \{(1,1)\}$, and where G_1 and G_2 are required to be NLC_2-decomposable and to contain v_1 and v_2 respectively, we first group the vertices in $V^* = \mathrm{V}(G) \setminus \{v_1, v_2\}$ into clusters by computing the strongly connected components of G_{dep} — the dependency graph with respect to v_1 and v_2. This can be done with two depth-first searches, as described in [1]. The time needed is linear in the size of G_{dep}, which is $O(n^2)$, where $n = |\mathrm{V}(G)|$. We assume here that G_{dep} is stored explicitly.

We thereafter compute the box structure. This we do by inserting one cluster C at a time. Either C fits in an existing box, or it must be placed in a new one. This new box will appear either between two unaffected old boxes (or at an end), or between the divided contents of an old box. The arrangement of all clusters can easily be computed in $O(n^2)$ time.

We are now set to search for a successful partition of V^*. The binary search

phase involves $O(\log n)$ partitions, each of which takes $O(n^3)$ time to check with Algorithm 1. If needed, we continue with the "box-splitting" phase. We then call Algorithm 1 twice for each cluster in the box in question. The total time for this sums to $O(n^3)$. The final partition can then be checked, again in $O(n^3)$ time. All in all, we use $O(n^3 \log n)$ time and $O(n^2)$ temporary space for each fixation of v_1 and v_2.

To find out if G has an NLC_2-decomposition on the form $\times_S (D_1, D_2)$, we can now repeat the above procedure for each edge $\{v_1, v_2\}$. However, without making things more than marginally more complicated, we can get by with only $n-1$ such repetitions. By the symmetry of S, we can take any vertex $u \in \mathrm{V}(G)$ and require that it shall belong to $G_2 = \mathrm{G}(D_2)$. First, we let $v_2 = u$, and we let each neighbor of u play the role of v_1. If this does not lead us to a successful partition, we know that u must be labeled with 2. This in turn brings all neighbors of u to G_2. Next, we let one of these neighbors, u', play the role of v_2, and we let each neighbor of u' that is not already in G_2 play the role of v_1.

The new thing here is that not only v_1 and v_2 are fixed, but other vertices are fixed too — some to G_2, and some of these even to the label 2. However, the latter have all their neighbors in G_2, so the label 2 is automatically compatible with the choice of v_1. For the previously described procedure, the extra requirement that some vertices (and thus clusters) must be placed in G_2 poses no problem. In the binary search phase, it means that some boxes will be predestined for G_2, and this only shortens this search. In the box-splitting phase, an extra requirement on a cluster C can be handled just as the requirements caused by indecomposability of $\mathrm{G}_{left}(C)$ and/or $\mathrm{G}_{right}(C)$.

Thus we can advance as indicated above. If we find no successful partition for $v_2 = u'$, we know that u' also must be labeled with 2, and that all its neighbors must go to G_2. Again, the role of v_2 can be assigned to one of the vertices that have been restricted to G_2, but not yet to the label 2. Particularly, we follow the described procedure by repeatedly letting v_2 and v_1 be parent and child in a breadth-first search through G, starting from u. This means $n-1$ edges $\{v_1, v_2\}$, and it follows that for $S = \{(1,1)\}$, we can find a possible NLC_2-decomposition of G on the form $\times_S (D_1, D_2)$ in $O(n^4 \log n)$ time and $O(n^2)$ space.

5.2.2 The case $S = \{(1,1),(2,2)\}$

Let $S = \{(1,1),(2,2)\}$. As for $S = \{(1,1)\}$, we shall discuss first how to determine the satisfiability of $G = \mathrm{unlab}(\times_S (G_1, G_2))$, where G_1 and G_2 are required to be NLC_2-decomposable and to contain v_1 and v_2 respectively, both labeled with 1. Our algorithm for this will be similar to that for $S = \{(1,1)\}$.

Like in that case, we find that when v_1 and v_2 have been fixed, we know what label any other vertex v must have when it is placed in either G_1 or G_2. In fact, the description of this for $S = \{(1,1)\}$ is still valid, including the definitions of $\mathrm{G}_{left}()$ and $\mathrm{G}_{right}()$. This is quite typical. In the following therefore, we will leave out much of what would be mere repetitions, and concentrate instead on those things that are — or might have been — different. We assume the previous presentation

to be fresh in the reader's mind.

Once again, there will be a dependency between the vertices in $V^* = \mathrm{V}(G) \setminus \{v_1, v_2\}$, which leads us to form clusters. We will use previous notation, but the relationship between clusters must be characterized anew. We look at $\rightarrow$ for specific pairs of vertex types:

- Let u be a 1–1-vertex and v a 1–2-vertex. If there is an edge (in G) between u and v, then $v \rightarrow u$. If there is no edge between u and v, then $u \rightarrow v$.
- Let u be a 1–1-vertex and v a 2–1-vertex. If there is an edge between u and v, then $u \rightarrow v$. If there is no edge between u and v, then $v \rightarrow u$.

Thus, if C_1 and C_2 are two different clusters, C_1 containing a 1–1-vertex and C_2 containing a 1–2-vertex or a 2–1-vertex, then we have either $C_1 < C_2$ or $C_2 < C_1$. By the symmetry of S, the same holds if C_1 contains instead a 2–2-vertex.

The remaining vertex type pairs are covered next:

- Let u be a 1–1-vertex and v a 2–2-vertex. If there is an edge between u and v, then $u \leftrightarrow v$. If there is no edge between u and v, then $u \mid v$.
- Let u be a 1–2-vertex and v a 2–1-vertex. If there is an edge between u and v, then $u \mid v$. If there is no edge between u and v, then $u \leftrightarrow v$.
- Let u and v be two vertices of the same type. Then either $u \leftrightarrow v$ or $u \mid v$.

Motivated by our new findings, we will speak of two *type categories*. 1–1 and 2–2 form one of these, and 1–2 and 2–1 the other. We can note that if two clusters, C_1 and C_2, together contain vertices of both categories, then $C_1 \rightarrow C_2$ or $C_2 \rightarrow C_1$. If instead they (together) contain vertices of only one category, then $C_1 \mid C_2$.

As before, we can now show that if C_1 and C_2 are clusters satisfying $C_1 \parallel C_2$, and if C is a third cluster, then $C < C_1$ implies $C < C_2$, and $C_1 < C$ implies $C_2 < C$. Let us provide the argument in the first case: $C < C_1$. There is then a cluster C', identical to C possibly (but distinct from C_1), such that $C \lesssim C' \rightarrow C_1$. We can conclude that C' contains a vertex type in the category which is not represented in C_1 and C_2. This means that either $C' < C_2$ or $C_2 < C'$. The latter would imply that $C_2 < C_1$ though, contradicting our initial assumptions, so we must have $C < C_2$.

It follows that we can group (in a unique way) clusters into boxes, so that we satisfy the previously formulated box structure properties. This means that we are "back on track". For example, a partition $\{V_1^*, V_2^*\}$ of V^* respects $\lesssim$ if and only if the monotonicity conditions are satisfied. This gives us the opportunity to use, as before, binary search among separators. We repeat: For a separator s, we partition V^* as $\{V_1^*, V_2^*\}$, where V_1^* is the union of all boxes to the left of s, and V_2^* is the union of all boxes to the right of s. We then define $G_1 = \mathrm{G}_{left}(v_1 \cup V_1^*)$ and $G_2 = \mathrm{G}_{right}(v_2 \cup V_2^*)$, and we use Algorithm 1 to check whether G_1 and G_2 are NLC_2-decomposable.

It was argued for $S = \{(1, 1)\}$ that binary search among separators ends in one of the following ways:

- We find a successful partition.
- We find a partition such that neither G_1 nor G_2 is NLC_2-decomposable. We

can conclude that there is no successful partition for the current fixation of v_1 and v_2.

- We find separators s_l and s_r immediately to the left and right of some box, B, such that when s_l is used, G_1 is NLC_2-decomposable but G_2 is not, and such that when s_r is used, G_2 is NLC_2-decomposable but G_1 is not. We can conclude that if there exists a successful partition, it must split B.

The argument is still correct, but when $S = \{(1,1),(2,2)\}$, the third case is no longer possible. If a successful partition splits a box B, we can re-split it any way. In particular, both the separator to the left of B and the one to the right will produce successful partitions. The arguments for this are as follows:

Case 1. B consists of 1–1-vertices and/or 2–2-vertices. Let C be a cluster in B. Note that $\mathrm{G}_{\mathit{left}}(C)$ and $\mathrm{G}_{\mathit{right}}(C)$ are identical. Let $\{V_1^*, V_2^*\}$ be a successful partition of V^* in which $C \subseteq V_2^*$. Thus, V_1^* contains all boxes to the left of B, and V_2^* contains all boxes to the right of B, so $\{V_1^* \cup C, V_2^* \setminus C\}$ respects $\lesssim$. We now show that it also is successful. Since $\{V_1^*, V_2^*\}$ is successful, $G_1 = \mathrm{G}_{\mathit{left}}(v_1 \cup V_1^*)$ and $G_2 = \mathrm{G}_{\mathit{right}}(v_2 \cup V_2^*)$ are both NLC_2-decomposable. By Lemma 1, so is $G_2' = \mathrm{G}_{\mathit{right}}(v_2 \cup V_2^* \setminus C)$. It remains to show that $G_1' = \mathrm{G}_{\mathit{left}}(v_1 \cup V_1^* \cup C)$ is NLC_2-decomposable. But $\mathrm{G}_{\mathit{left}}(C) = \mathrm{G}_{\mathit{right}}(C)$, so $G_1' = \times_{\{(1,1),(2,2)\}}(G_1, \mathrm{G}_{\mathit{left}}(C))$. It follows that $\{V_1^* \cup C, V_2^* \setminus C\}$ is successful. By symmetry, it conversely follows that if $\{V_1^*, V_2^*\}$ is a successful partition of V^* in which $C \subseteq V_1^*$, then $\{V_1^* \setminus C, V_2^* \cup C\}$ is successful too.

Case 2. B consists of 1–2-vertices and/or 2–1-vertices. Let C be a cluster in B. Note that $\mathrm{G}_{\mathit{left}}(C)$ and $\mathrm{G}_{\mathit{right}}(C)$ are complementary in their labeling, so if one of them is NLC_2-decomposable, the other is too. Let $\{V_1^*, V_2^*\}$ be a successful partition of V^* in which $C \subseteq V_2^*$. Thus, $\{V_1^* \cup C, V_2^* \setminus C\}$ respects $\lesssim$. We now show that it also is successful. Since $\{V_1^*, V_2^*\}$ is successful, $G_1 = \mathrm{G}_{\mathit{left}}(v_1 \cup V_1^*)$ and $G_2 = \mathrm{G}_{\mathit{right}}(v_2 \cup V_2^*)$ are both NLC_2-decomposable. By Lemma 1, so is $G_2' = \mathrm{G}_{\mathit{right}}(v_2 \cup V_2^* \setminus C)$. It remains to show that $G_1' = \mathrm{G}_{\mathit{left}}(v_1 \cup V_1^* \cup C)$ is NLC_2-decomposable. But since $\mathrm{G}_{\mathit{left}}(C)$ can be obtained from $\mathrm{G}_{\mathit{right}}(C)$ by switching the roles of 1 and 2, we have $G_1' = \times_{\{(1,2),(2,1)\}}(G_1, \mathrm{G}_{\mathit{left}}(C))$. It follows that $\{V_1^* \cup C, V_2^* \setminus C\}$ is successful. By symmetry, it conversely follows that if $\{V_1^*, V_2^*\}$ is a successful partition of V^* in which $C \subseteq V_1^*$, then $\{V_1^* \setminus C, V_2^* \cup C\}$ is successful too.

Let us then summarize: To determine the satisfiability of $G = \mathrm{unlab}(\times_S(G_1, G_2))$, where $S = \{(1,1),(2,2)\}$, and where G_1 and G_2 are required to be NLC_2-decomposable and to contain v_1 and v_2 respectively, both labeled with one, we group the vertices in $V^* = \mathrm{V}(G) \setminus \{v_1, v_2\}$ into clusters, and we continue by computing the box structure. We then use binary search among box separators and check each generated partition with Algorithm 1. As already argued, we can do this in $O(n^3 \log n)$ time and $O(n^2)$ temporary space.

Once again, to find out if G has an NLC_2-decomposition on the form $\times_S(D_1, D_2)$, we can repeat the above procedure for each edge $\{v_1, v_2\}$. As before however, we can get by with $n-1$ such repetitions. This time S is symmetric with respect to labels also. We can therefore take any vertex $u \in \mathrm{V}(G)$ and require that

it shall belong to $G_2 = \mathrm{G}(D_2)$ and have label 1 there. So, we let $v_2 = u$, and we let each neighbor of u play the role of v_1. It follows that for $S = \{(1,1),(2,2)\}$, we can find a possible NLC_2-decomposition of G on the form $\times_S (D_1, D_2)$ in $O(n^4 \log n)$ time and $O(n^2)$ space.

5.2.3 The case $S = \{(1,2),(2,1),(2,2)\}$

The final case, $S = \{(1,2),(2,1),(2,2)\}$, can easily be reduced to the first, $S = \{(1,1)\}$. Letting $\overline{S}$ denote $[2]^2 \setminus S$, we note: $\times_S (D_1, D_2)$ is an NLC_2-decomposition of G if and only if $\times_{\overline{S}} \left(\overline{D_1}, \overline{D_2}\right)$ is an NLC_2-decomposition of $\overline{G}$.

5.3 Summary and Concluding Analysis

To NLC_2-decompose a graph G that is unlabeled or labeled with numbers in $\{1,2\}$, we first compute the modular decomposition of G, $\mathrm{D_M}(G)$, as defined in Section 4. With the method described in Section 4.4, this takes $O(n^2)$ time, where $n = |\mathrm{V}(G)|$.

We then try to NLC_2-decompose each quotient graph Q in $\mathrm{D_M}(G)$. When Q is properly prime, we use the algorithm described in Section 5, running in $O({n_Q}^4 \log n_Q)$ time (where $n_Q = |\mathrm{V}(Q)|$). When Q is not properly prime, its vertices can be combined in any order, and we can surely construct a decomposition in linear time. By Lemma 3, the total time for decomposition of quotient graphs becomes $O(n^4 \log n)$. The space used never exceeds $O(n^2)$.

If we now have an NLC_2-decomposition of each quotient graph in $\mathrm{D_M}(G)$, then we piece together these decompositions into an NLC_2-decomposition of G, as described in the proof of Proposition 1. Only linear time is needed for this last step. In total, we have used $O(n^4 \log n)$ time and $O(n^2)$ space.

Acknowledgments

I am grateful to Stefan Arnborg for his advice and comments.

References

1. T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms* (MIT Press, Cambridge, 1990).
2. D. G. Corneil, H. Lerchs and L. S. Burlingham, "Complement reducible graphs," *Discrete Appl. Math.* **3** (1981) 163–174.
3. D. G. Corneil, Y. Perl and L. K. Stewart, "A linear recognition algorithm for cographs," *SIAM J. Comput.* **14** (1985) 926–934.
4. B. Courcelle, J. A. Makowsky and U. Rotics, "On the fixed parameter complexity of graph enumeration problems definable in monadic second order logic." To appear in *Discrete Appl. Math.*
5. B. Courcelle, J. A. Makowsky and U. Rotics, "Linear time solvable optimization problems on graphs of bounded clique width," in *Proc. 24th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science **1517** (Springer, Berlin, 1998) pp. 1–16.

6. B. Courcelle and S. Olariu, "Clique-width: A graph complexity measure—preliminary results and open problems," in *Proc. 5th Int. Workshop on Graph Grammars and Their Application to Computer Science*, Williamsburg, VA, November 1994, pp. 263–270.
7. A. Ehrenfeucht, H. N. Gabow, R. M. McConnell and S. J. Sullivan, "An $O(n^2)$ divide-and-conquer algorithm for the prime tree decomposition of two-structures and modular decomposition of graphs," *J. Algorithms* **16** (1994) 283–294.
8. A. Ehrenfeucht and G. Rozenberg, "Theory of 2-structures, part I: Clans, basic subclasses, and morphisms," *Theoret. Comput. Sci.* **70** (1990) 277–303.
9. A. Ehrenfeucht and G. Rozenberg, "Theory of 2-structures, part II: Representation through labeled tree families," *Theoret. Comput. Sci.* **70** (1990) 305–342.
10. Ö. Johansson, "Clique-decomposition, NLC-decomposition, and modular decomposition — relationships and results for random graphs," *Congr. Numer.* **132** (1998) 39–60.
11. J. A. Makowsky and U. Rotics, "On the clique-width of graphs with few P_4's." To appear in *Internat. J. Found. Comput. Sci.*
12. R. M. McConnell, "An $O(n^2)$ incremental algorithm for modular decomposition of graphs and 2-structures," *Algorithmica* **14** (1995) 229–248.
13. R. H. Möhring, "Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and boolean functions," *Ann. Oper. Res.* **4** (1985/6) 195–225.
14. R. H. Möhring and F. J. Radermacher, "Substitution decomposition for discrete structures and connections with combinatorial optimization," *Ann. Discrete Math.* **19** (1984) 257–356.
15. E. Wanke, "k-NLC graphs and polynomial algorithms," *Discrete Appl. Math.* **54** (1994) 251–266.