k-NLC Graphs and Polynomial Algorithms*

Egon Wanke† Frank Gurski ‡

Abstract

We introduce the class of *k-node label controlled* (k-NLC) *graphs* and the class of k-NLC *trees*. Each k-NLC graph is an undirected tree-structured graph, where k is a positive integer. The class of k-NLC trees is a proper subset of the class of k-NLC graphs. Both classes include many interesting graph families. For instance, each *partial k-tree* is a $(2^{k+1}-1)$-NLC tree and each co-graph is a 1-NLC graph.

Furthermore, we introduce a very general method for the design of polynomial algorithms for NP-complete graph problems, where the input graphs are restricted to tree-structured graphs. We exemplify our method with the SIMPLE MAX-CUT PROBLEM and the HAMILTONIAN CIRCUIT PROPERTY on k-NLC graphs.

1 Introduction

A large number of papers have been published concerning special classes of tree-structured graphs. A well-known and extensively studied class of tree-structured graphs is the class of *k-trees* [Ros74] and *partial k-trees* which is equivalent to the class of graphs having *tree-width k* [RS86]. In this paper, we define a more extensive class of graphs. We call it the class of *k-node label controlled* (k-NLC) *graphs*. k-NLC graphs are recursively defined. The definition is based on a certain labeling of vertices in undirected graphs. The class of k-NLC graphs seems to be the most general class of graphs defined by adjacencies. A simple natural restriction of the recursive definition of k-NLC graphs leads to the class of k-NLC *trees*, which is a proper subset of the class of k-NLC graphs.

The class of k-NLC graphs has some nice and interesting properties. It includes several other important classes of graphs. For instance, each partial k-tree is a $(2^{k+1}-1)$-NLC tree. The class of k-NLC graphs also contains *dense* graphs, i.e., graphs with n vertices and $\Omega(n^2)$ edges. For instance, the class of 1-NLC graphs is exactly the class of all *co-graphs* [CPS84], and thus contains all complete graphs.

*Rivised version of [Wan94]

†Department of Computer Science, Mathematical Institute, Heinrich-Heine-University Düsseldorf, D-40225 Düsseldorf, Germany, e-mail: wanke@cs.uni-duesseldorf.de

‡Department of Computer Science, Mathematical Institute, Heinrich-Heine-University Düsseldorf, D-40225 Düsseldorf, Germany, e-mail: gurski@cs.uni-duesseldorf.de

Several authors discussed the problem how to solve NP-complete graph problems on tree-structured graphs in polynomial time. Three of these approaches are [ALS91, BLW87, Bod88]. Arnborg, Lagergren, and Seese in [ALS91] are interested in giving a very general and logical criterion for a graph property to be decidable in polynomial time when restricted to partial k-trees. Bodlaender introduces in [Bod88] two classes of graph problems called ECC and LCC problems that can be solved in polynomial time on partial k-trees with dynamic programming methods. Bern, Lawler, and Wong give in [BLW87] a frame for linear time algorithms for the computation of certain optimal subgraphs of partial k-trees.

All authors in [ALS91, BLW87, Bod88] assume that a *tree-structured expression* or equivalently a *composition tree* for the input graph is given. Such a composition can be found in linear time for each partial k-tree, see [ACP87]([Bod93]). The recognition problem for k-NLC graphs and k-NLC trees, however, is still open for each $k > 1$. Thus, we shall assume that when we are given an input graph, we are also given its composition.

In this paper, we introduce a simple and very general method to solve difficult graph problems on tree-structured graphs in polynomial time. Our method works as follows: Let Π be the graph problem that we want to solve on a tree-structured graph G. Assume that the graph G is defined by a system M. Usually, M is an algebra of graphs. We map M onto a polynomial algebra M_Π of abstract structures that has the same signature as M. That is, for each G in M given by its composition, we can compute in polynomial time an abstract structure G' in M_Π such that $\Pi(G) = \Pi'(G')$. Here, Π' is a problem on the abstract structures of M_Π. The mapping from M to M_Π has to be a polynomial outspace transformation. Such a graph problem is called a *polynomial recognizable* graph problem.

We exemplify our method with the SIMPLE MAX-CUT PROBLEM and the HAMILTONIAN CIRCUIT PROPERTY on k-NLC graphs. That is, we show that the SIMPLE MAX-CUT PROBLEM and the HAMILTONIAN CIRCUIT PROPERTY are polynomial recognizable with respect to k-NLC graphs. The second example also shows that the LONGEST CIRCUIT and the LONGEST PATH PROBLEM can be solved in polynomial time on k-NLC graphs.

2 Node label controlled graphs

We work with finite (undirected) *graphs* $G = (V_G, E_G)$, where V_G is a finite set of *vertices* and $E_G \subseteq \{\{u, v\} | u, v \in V_G \wedge u \neq v\}$ is a set of *(undirected) edges*. $size(G) := |V_G| + |E_G|$ denotes the size of G, and $\mathcal{G}$ denotes the set of all graphs.

Let G be a graph. A graph J is a *subgraph* of G, denoted $J \subseteq G$, if V_J is a subset of V_G and E_J is a subset of E_G. J is an *induced subgraph* of G if J is a subgraph of G and $E_J = \{\{u, v\} \in E_G | u, v \in V_J\}$. J and G are called *equal*, denoted $J = G$, if there is a bijection b from V_J to V_G that preserves adjacencies, i.e., $\{u, v\} \in E_J \Leftrightarrow \{b(u), b(v)\} \in E_G$.

Let $k \in \mathbb{N}$ be a positive integer and $[k]$ be the set of integers $\{1, 2, \ldots, k\}$. A *k-node labeled* (k-NL) *graph* is a system $G = (V_G, E_G, lab_G)$, where (V_G, E_G) is a graph and $lab_G : V_G \to [k]$ is a *vertex label mapping* that labels each vertex $u \in V_G$ with a positive integer $lab(u)$ from $[k]$. $unlab(G) := (V_G, E_G)$ denotes the unlabeled graph underlying G, $size(G) := size(unlab(G))$ denotes the size of G, and NL_k denotes the set of all k-NL

graphs.

Let $G, J \in \mathrm{NL}_k$ be two k-NL graphs. J is a k-NL *subgraph* of G, denoted $J \subseteq G$, if $V_J \subseteq V_G$, $E_J \subseteq E_G$, and $lab_J(u) = lab_G(u)$ for all vertices $u \in V_J$. J is an *induced* k-NL *subgraph* of G if $J \subseteq G$ and $E_J = \{\{u, v\} \in E_G | u, v \in V_J\}$. J and G are called *equal*, denoted $J = G$, if there is a bijection b from V_J to V_G that preserves adjacencies and the labeling, i.e., $\{u, v\} \in E_J \Leftrightarrow \{b(u), b(v)\} \in E_G$ and $lab_J(u) = lab_G(b(u))$ for all vertices $u \in V_J$.

We start with the recursive definition of the class of k-NLC *graphs*. Usually, a *recursive definition* of a class of graphs consists of a finite number of *primitive graphs* that serves as a base case and some *instructions* about how to manufacture further graphs by certain rules.

Definition 1: Let $k \in \mathbb{N}$ be a positive integer. A *k-node label controlled* (k-NLC) *graph* is a k-NL graph defined as follows:

1. The k-NL graph that consists of a single vertex labeled by a positive integer $t \in [k]$ is a k-NLC graph denoted by $\bullet_t$.

2. Let G be a k-NLC graph and R be a mapping from $[k]$ to $[k]$. The k-NL graph J defined by

$$\begin{array}{lcl} V_J & := & V_G, \\ E_J & := & E_G, \\ lab_J(u) & := & R(lab_G(u)) \text{ for all vertices } u \text{ in } J \end{array}$$

   is a k-NLC graph denoted by $\circ_R(G)$.

3. Let G and J be two vertex disjoint k-NLC graphs and $S \subseteq [k]^2$. The k-NL graph H defined by

$$\begin{array}{lcl} V_H & := & V_G \cup V_J, \\ E_H & := & E_G \cup E_J \cup \{\{u, v\} | u \in V_G \wedge v \in V_J \wedge (lab_G(u), lab_J(v)) \in S\}, \\ lab_H(u) & := & lab_G(u) \text{ for all vertices } u \text{ from } G, \\ lab_H(u) & := & lab_J(u) \text{ for all vertices } u \text{ from } J \end{array}$$

   is a k-NLC graph denoted by $G \times_S J$.

   Intuitively speaking, H is the disjoint union of G and J including all edges $\{u, v\}$ between a vertex u from G and a vertex v from J if $(lab(u), lab(v)) \in S$.

4. NLC_k denotes the set of all k-NLC graphs.

Figure 1 shows an example for the construction of a 4-NLC graph. A simple modification of Definition 1 yields the definition of the class of k-NLC *trees*. We define $G \times_S J$ to be a k-NLC tree if G and J are k-NLC trees and either $S = \emptyset$ or one of the k-NLC trees

G or J has a fixed size. Without loss of generality, we will assume that either G or J has exactly one vertex. We have the following definition for k-NLC trees:

Definition 2: Let $k \in \mathbb{N}$ be a positive integer. A *k-node label controlled* (k-NLC) *tree* is a k-NL graph defined as follows:

1. The k-NLC graph $\bullet_t$ for $t \in [k]$ is a k-NLC tree.
2. Let G be a k-NLC tree and $R : [k] \to [k]$. The k-NLC graph $\circ_R(G)$ is a k-NLC tree.
3. Let G and J be two k-NLC trees, $t \in [k]$, and $S \subseteq [k]^2$. The k-NLC graphs $G \times_\emptyset J$ and $G \times_S \bullet_t$ are k-NLC trees.
4. NLCT_k denotes the set of all k-NLC trees.

The class of k-NLC trees is an auxiliary construction that could facilitate the analysis of graph problems on a subset of k-NLC graphs. However, we do not know any problem polynomial for k-NLC trees and NP-complete for k-NLC graphs. This also includes the recognition of k-NLC trees and k-NLC graphs.

The vertices in k-NL graphs are labeled with positive integers from $[k]$. This labeling is important for further constructions of k-NLC graphs or k-NLC trees, see Definition 1.3 and 2.3. However, if we talk about graph properties of k-NL graphs or if we compare k-NL graphs with usual graphs, we will ignore the labeling of the vertices.

Let us compare the class of k-NLC graphs and the class of k-NLC trees with other classes of graphs. First we will show that the class of 1-NLC trees is exactly the class of all *co-graphs*. Co-graphs are known by several names (Heredity Dacey [Sum74], D^*, 2-parity [BU84], complement reducible [CPS84], P_4-free), see [Bra90] for a survey. Co-graphs can be defined as follows:

1. A single vertex is a co-graph denoted by $\bullet$.
2. The disjoint union of two co-graphs G and J is a co-graph denoted by $G \cup J$.
3. The complement graph $(V_G, \{\{u,v\} | u,v \in V_G \wedge u \neq v \wedge \{u,v\} \notin E_G\})$ of a co-graph G is a co-graph denoted by $\overline{G}$.

Fact 1: G is a 1-NLC graph if and only if $unlab(G)$ is a co-graph.

Proof: Each composition of a co-graph can be transformed into a composition of an equivalent 1-NLC graph with the transformation steps defined in the following table:

co-graph		1-NLC graph
$\bullet$	$\Rightarrow$	$\bullet_1$
$G \cup J$	$\Rightarrow$	$G \times_\emptyset J$
$\overline{\bullet}$	$\Rightarrow$	$\bullet_1$
$\overline{G \cup J}$	$\Rightarrow$	$\overline{G} \times_{\{(1,1)\}} \overline{J}$

Since the identity is the only function R from [1] to [1], we have $\circ_R(G) = G$ for all 1-NLC graphs G. That is, each composition of a 1-NLC graph can be transformed into a composition of an equivalent co-graph with the transformation steps defined in the following table:

1-NLC graph		co-graph
$\bullet_1$	$\Rightarrow$	$\bullet$
$\circ_R(G)$	$\Rightarrow$	G
$G \times_{\emptyset} J$	$\Rightarrow$	$G \cup J$
$G \times_{\{(1,1)\}} J$	$\Rightarrow$	$\overline{\overline{G} \cup \overline{J}}$

This shows that the class of all 1-NLC graphs is the class of all co-graphs. □

It is also easy to verify that the class of k-NLC graphs is closed under complementation for each integer k. Note that this fact does not hold for k-NLC trees.

Fact 2: G is a k-NLC graph if and only if its complement $\overline{G} := (V_G, \{\{u,v\} | u,v \in V_G \wedge u \neq v \wedge \{u,v\} \notin E_G\}, lab_G)$ is a k-NLC graph.

Proof: For all $t \in [k]$, $R : [k] \to [k]$, $S \subseteq [k]^2$, and $G, J \in \mathrm{NLC}_k$, we have:

$$\begin{array}{rcl} \overline{\bullet_t} & = & \bullet_t \\ \overline{\circ_R(G)} & = & \circ_R(\overline{G}) \\ \overline{G \times_S J} & = & \overline{G} \times_{\overline{S}} \overline{J} \end{array}$$

where $\overline{S} = \{(i,j) \in [k]^2 | (i,j) \notin S\}$. □

However, both classes NLC_k and NLCT_k are closed under induced subgraphs. This follows directly by Definition 1 and 2. However, NLC_k and NLCT_k are not closed under all subgraphs, because for each positive integer k, each complete graph is in NLC_k and NLCT_k, but neither NLCT_k nor NLC_k include all graphs.

It is also important to note that the class of k-NLC graphs does not include all *proper interval graphs* and all *permutation graphs*. This implies that NLC_k also does not include all *interval, chordal, split, circular permutation, or circular arc graphs*. For a definition of all these classes of graphs see [Bra90].

Another interesting class of recursively defined graphs is the class of *partial k-trees* that is defined by the class of k-trees including all its subgraphs. The class of all k-trees can be defined as follows, see [Ros74]:

1. A complete graph with k vertices is a k-tree.
2. If G is a k-tree, then add a vertex adjacent to all k vertices of a complete subgraph of G. The result is a k-tree.

There is a close relation between the class of partial k-trees and the class of k-NLC trees. Let us show that each partial k-tree G is a $(2^{k+1}-1)$-NLC tree and thus a $(2^{k+1}-1)$-NLC graph.

Theorem 1: For each partial k-tree G there is a $(2^{k+1}-1)$-NLC tree J with $G = unlab(J)$.

Proof: Let G be a graph with n vertices and $o = (v_1, \ldots, v_n)$ be an order of the n vertices of G. The sets $N^+(G, o, i)$ and $N^-(G, o, i)$ for $1 \leq i \leq n$ are defined by the neighbour vertices v_j of vertex v_i with $j > i$ and $j < i$, respectively. That is,

$$N^+(G, o, i) := \{v_j | \{v_i, v_j\} \in E_G \wedge i < j\}$$

and

$$N^-(G, o, i) := \{v_j | \{v_j, v_i\} \in E_G \wedge j < i\}.$$

The order $(v_1, \ldots, v_n)$ is called a *perfect elimination order* (PEO) of G if the vertices from N_i^+ for $1 \leq i \leq n$ induce a complete subgraph of G.

Assume G is a partial k-tree. Let G be a subgraph of a k-tree H. Without loss of generality, G has the same vertex set as H, i.e., $V_G = V_H$. By the recursive definition of k-trees, we know that each k-tree H has a PEO. Let $o = (v_1, \ldots, v_n)$ be a PEO of H. Then, the vertices of the sets $N^+(G, o, 1), \ldots, N^+(G, o, n-k)$ induce k vertex complete subgraphs of H. The vertices of each $N^+(G, o, i)$ for $n - k < i \leq n$ induce an $n - i$ vertex complete subgraph of H.

The structure of the k-tree H can be characterized by the tree $T(H, o) := (V_T, E_T)$ defined by

$$\begin{array}{lcl} V_T & := & V_H \\ E_T & := & \{\{v_i, v_j\} \in E_H | i < j \wedge \text{ if } i < j' < j \text{ then } \{v_i, v_{j'}\} \notin E_H\}. \end{array}$$

$T(H, o)$ is a tree, because it is connected and each vertex v_i of $T(H, o)$ has at most one edge $\{v_i, v_j\} \in E_H$ with $i < j$.

Let col be a $k+1$-coloring of H. That is, there is a mapping $col : V_H \to [k+1]$ with $col(u) \neq col(v)$ for all $\{u, v\} \in E_H$. It is also easy to see that each k-tree is $k+1$ colorable.

Note that col is also a coloring of G. We define C_i to be the set of all colors of the vertices from $N^+(G, o, i)$, i.e.,

$$C_i := \{col(v_j) | \{v_i, v_j\} \in E_G \wedge i < j\}.$$

Let b be any bijection from $2^{[k+1]}$ to $[2^{k+1}]$. The bijection b is used to represent each set of colors C_i from $[k+1]$ by an integer $b(C_i)$ from $[2^{k+1}]$.

Now we will recursively define a $(2^{k+1}-1)$-NLC tree J_i for $1 \leq i \leq n$.

1. If $N^-(T(H, o), o, i) = \emptyset$, then let $J_i := \bullet_{b(C_i)}$.

2. If $N^-(T(H,o),o,i) = \{v_{j_1},\ldots,v_{j_m}\}$, then let $J_i := \circ_R(\bullet_{b(C_i)} \times_S J)$, where

$$\begin{array}{lcl} J & := & J_{j_1} \times_\emptyset \cdots \times_\emptyset J_{j_m}, \\ S & := & \{(b(C_i), b(T)) | T \subseteq [k+1] \wedge col(v_i) \in T\}, \\ R(b(T)) & := & b(T - col(v_i)) \text{ for all } T \subseteq [k+1]. \end{array}$$

Figure 2 shows an example of such a composition. Since each set C_i has at most k integers, J_i is a $(2^{k+1}-1)$-NLC tree. Let us show that the graph underlying J_n is equal to the partial k-tree G.

By the recursive definition of J_n, there is a natural one to one relation between the vertices of G and the vertices of J_n. Let $u_1,\ldots,u_n$ be the vertices of J_n such that vertex u_i is the single vertex $\bullet_{b(C_i)}$ defined in step i.

If $u_{l_1},\ldots,u_{l_m}$ are the vertices of J_i, we define G_i to be the graph induced by the vertices $v_{l_1},\ldots,v_{l_m}$ of G. Now we show that $unlab(J_i) = G_i$ for $1 \leq i \leq n$. Furthermore, we show that the vertices of J_i have a special labeling. Vertex u_l of J_i is labeled by $b(T)$ if and only if T is the set of all colors $col(u_j)$ with $\{v_l, v_j\} \in E_G$ and $l \leq i < j$.

Induction on i. Basis: Let $i = 1$. G_1 is a single vertex u_1 labeled with $b(C_1)$, where $C_1 := \{col(v_j) | \{v_1, v_j\} \in E_G \wedge 1 < j\}$.

Induction: Let $i > 1$. If $N^-(T(H,o),o,i) = \emptyset$ then G_i is a single vertex u_i labeled with $b(C_i)$, where $C_i := \{col(v_j) | \{v_i, v_j\} \in E_G \wedge i < j\}$.

If $N^-(T(H,o),o,i) = \{v_{j_1},\ldots,v_{j_m}\}$, then J is the disjoint union of all $J_{j_1},\ldots,J_{j_m}$ and J_i is defined by a composition of J and $\bullet_{b(C_i)}$ (the vertex u_i). The composition creates an edge between u_i and a vertex u_j of J labeled with $b(T)$ if and only if $col(v_i) \in T$. By the inductive hypothesis for $J_{j_1},\ldots,J_{j_m}$, it follows that $unlab(J_i) = G_i$.

The relabeling of J_i changes each label $b(T)$ to $b(T - col(v_i))$. Note that this relabeling does not change the labeling $b(C_i)$ of u_i, because $col(v_i) \notin C_i$. By the inductive hypothesis and the relabeling above each vertex u_l from J_i is labeled by $b(T)$ if and only if T is the set of all $col(v_j)$ with $\{u_l, u_j\} \in E_G$ and $l \leq i < j$. □

An important cautionary note is that it may be difficult to recognize whether or not a graph G is a k-NLC graph or a k-NLC tree. The k-NLC graph and k-NLC tree recognition problems for each $k > 1$ are still open. We avoid these recognition problems, and in subsequent sections, we shall simply assume that, when we are given a k-NLC graph, we are also given its composition.

3 Polynomial algorithms

Most graph problems of interest search for a structure of a certain type. Such a structure can consist of sets, graphs, functions, numbers, true and false etc. We are interested in computing such structures for tree-structured graphs in polynomial time. We restrict ourselves to graph problems $\Pi : \mathcal{G} \to \mathcal{C}$ that take a single graph $G \in \mathcal{G}$ as input. $\mathcal{C}$ denotes

the set of all *output structures*. The type of the output structures will be unimportant for the results in this paper.

We illustrate the correctness of our solution method in the very general context of algebraic specifications. The set of all k-NL graphs together with the *constants* $\bullet_t \in \mathrm{NL}_k$ for $t \in [k]$, the *operations* $\circ_R : \mathrm{NL}_k \to \mathrm{NL}_k$ for $R : [k] \to [k]$, and the *operations* $\times_S : (\mathrm{NL}_k \times \mathrm{NL}_k) \to \mathrm{NL}_k$ for $S \subseteq [k]^2$ can be considered as an *algebra of graphs* with *base set* NL_k. This algebra of graphs is similar to that in [Cou87].

A mapping h is called a *t-time* and *s-outspace* mapping, if $h(x)$ is computable in time $O(t(size(x)))$ and the size of $h(x)$ is in $O(s(size(x)))$, respectively. Here, $s, t : \mathbb{N} \to \mathbb{N}$ are two integer mappings and $size(x) \in \mathbb{N}$ denotes the size of x. By *P-time, P-outspace, and 1-outspace* we denote polynomial time, polynomial outspace, and constant outspace, respectively.

We continue with the definition of a *polynomial recognizable graph problem.*

Definition 3: A graph problem $\Pi : \mathcal{G} \to \mathcal{C}$ is called *polynomial recognizable (P-recognizable)* with respect to the class of all k-NLC graphs if there is

1. a set of *abstract objects* A_k,
2. *constants* $\bullet_{\Pi,t} \in A_k$ for $t \in [k]$,
3. P-time *operations* $\circ_{\Pi,R} : A_k \to A_k$ for $R : [k] \to [k]$,
4. P-time *operations* $\times_{\Pi,S} : (A_k \times A_k) \to A_k$ for $S \subseteq [k]^2$,
5. a P-outspace *transformation* $h_{k,\Pi} : \mathrm{NL}_k \to \mathrm{A}_k$, and
6. a P-time *problem* $\Delta_\Pi : \mathrm{A}_k \to \mathcal{C}$

such that for all $G, J \in \mathrm{NL}_k$ and all $t \in [k]$, $R : [k] \to [k]$, $S \subseteq [k]^2$:

$$\begin{array}{lllll} \text{a.} & h_{k,\Pi}(\bullet_t) & = & \bullet_{\Pi,t} \\ \text{b.} & h_{k,\Pi}(\circ_R(G)) & = & \circ_{\Pi,R}(h_{k,\Pi}(G)) \\ \text{c.} & h_{k,\Pi}(G \times_S J) & = & h_{k,\Pi}(G) \times_{\Pi,S} h_{k,\Pi}(J) \\ \text{d.} & \Pi(unlab(G)) & = & \Delta_\Pi(h_{k,\Pi}(G)) \end{array}$$

The notion of a *P-recognizable* graph problem is defined with respect to the algebra of k-NLC graphs. A more general definition can be obtained by a straightforward extension to algebras with arbitrary signatures.

Theorem 2: If Π is a P-recognizable graph problem, then for each k-NLC graph G, $\Pi(unlab(G))$ is computable in polynomial time in the size of G if G is given by its composition.

Proof: To determine $\Pi(unlab(G))$, we simply compute $\Delta_\Pi(h_{k,\Pi}(G))$, which is equivalent to $\Pi(unlab(G))$ by Definition 3.6.d. Definition 3.6.a, 3.6.b, and 3.6.c show that the abstract object $h_{k,\Pi}(G)$ is computable as follows:

1. If $G = \bullet_t$ for some $t \in [k]$, then $h_{k,\Pi}(G) := \bullet_{\Pi,t}$.

2. If $G = \circ_R(J)$ for some $R : [k] \to [k]$, then $h_{k,\Pi}(G) := \circ_{\Pi,R}(h_{k,\Pi}(J))$.

3. If $G = J \times_S H$ for some $S \subseteq [k]^2$, then $h_{k,\Pi}(G) := h_{k,\Pi}(J) \times_{\Pi,S} h_{k,\Pi}(H)$.

Note that $h_{k,\Pi}$ is a P-outspace transformation, see Definition 3.5. That is, for each k-NL graph G, the size of $h_{k,\Pi}(G)$ is polynomial in the size of G.

The computation of $h_{k,\Pi}(\circ_R(J))$, $h_{k,\Pi}(J \times_S H)$, and $\Delta_\Pi(h_{k,\Pi}(G))$ take polynomial time in the size of its inputs, because $\circ_{\Pi,R}$, $\times_{\Pi,S}$, and Δ_Π are P-time mappings by definition. □

Theorem 2 shows also that the P-outspace transformation $h_{k,\Pi}$ of a P-recognizable graph problem is a P-time transformation for all k-NLC graphs if a tree structured expression of a k-NLC graph is computable in polynomial time.

Assume that Π is a P-recognizable graph problem with a 1-space transformation $h_{k,\Pi}$. Such a graph problem is called a 1-*recognizable* graph problem. Then, we can compute each $\circ_R(X)$ and $X \times_{\Pi,S} Y$ in constant time for all $X, Y \in h_{k,\Pi}(\mathrm{NL}_k)$ $(= \{h_{k,\Pi}(G) | G \in \mathrm{NL}_k\})$, because each object in $h_{k,\Pi}(\mathrm{NL}_k)$ has a constant size, i.e, $h_{k,\Pi}(\mathrm{NL}_k)$ is finite. Now, the computation of $h_{k,\Pi}(G)$ with the procedure from the proof of Theorem 2 takes linear time in the size of the composition of G. This implies that $\Delta_\Pi(G)$, and thus $\Pi(\mathit{unlab}(G))$ is computable in linear time in the size of the number of vertices in G.

Note that for each 1-recognizable graph problem $\Pi : \mathcal{G} \to \mathcal{C}$, the set of output structures is always finite.

Courcelle [Cou87] has shown that each graph property $\Pi : \mathcal{G} \to \{true, false\}$— expressible in monadic second-order logic— is 1-recognizable with respect to a class of graphs similar to the class of k-NLC graphs. That is, such graph problems are decidable in linear time also for each k-NLC graph G given by its composition. The monadic second-order logic only allows quantifications on vertex sets and vertices. In this paper, however, we only consider graph problems Π that are P-recognizable and not 1-recognizable.

To show that a graph problem Π is not 1-recognizable, we simply show that there is an infinite number of k-NL graphs $G, J \in \mathrm{NL}_k$ such that

$$\Pi(\mathit{unlab}(G \times_S H)) \neq \Pi(\mathit{unlab}(J \times_S H))$$

for some $S \subseteq [k]^2$ and some $H \in \mathrm{NL}_k$. This implies

$$\Delta_\Pi(h_{k,\Pi}(G \times_S H)) \neq \Delta_\Pi(h_{k,\Pi}(J \times_S H))$$

and

$$h_{k,\Pi}(G) \neq h_{k,\Pi}(J).$$

If we have an infinite number of different abstract objects in $h_{k,\Pi}(\mathrm{NL}_k)$, we cannot represent each of them on constant space. Thus, Π is not 1-recognizable.

4 The simple max-cut problem

In this section, we will show that the SIMPLE MAX-CUT PROBLEM is P-recognizable with respect to the class of all k-NLC graphs. Then, by Theorem 2, it will be computable in polynomial time on a k-NLC graph G if G is given by its composition.

The SIMPLE MAX-CUT PROBLEM is defined as follows: Let G be a graph and V be a set of vertices from V_G. The system (G, V) is called a *cut* of G. The *cut-size* of a cut $C = (G, V)$ is the number of edges in G that have one vertex in V and one vertex in $V_G - V$. A cut C of G is called *maximal* if there is no cut of G with a cut-size greater than the cut-size of C. The SIMPLE MAX-CUT PROBLEM $\Pi : \mathcal{G} \to \mathbb{N}_0$ is the problem to compute the cut-size of a maximal cut of a given graph G. The corresponding decision problem is NP-complete in general, see problem DN16 in [GJ79].

The SIMPLE MAX-CUT PROBLEM is not 1-recognizable with respect to the class of k-NLC graphs, because the set of all output structures $\mathbb{N}_0$ is not finite. However, many graph problems that are certain versions of the SIMPLE MAX-CUT PROBLEM are also not 1-recognizable, even although they have a finite set of output structures. For instance, let $\Pi : \mathcal{G} \to \{true, false\}$ be the graph property such that $\Pi(G)$ holds true if and only if G has a maximal cut of cut-size $|V_G|^2/4$. The property Π is not 1-recognizable. Assume that G and J are two totally disconnected 1-NL graphs with a different number of vertices. Then, for each transformation $h_{k,\Pi}$, we have

$$h_{k,\Pi}(G) \neq h_{k,\Pi}(J),$$

because $H = G \times_{\{(1,1)\}} J$ has a maximal cut of cut-size $|V_G| \cdot |V_J|$, which is less than $|V_H|^2/4$, and $J \times_{\{(1,1)\}} J$ has a maximal cut of cut-size $|V_J|^2$, which is equal to $|V_H|^2/4$.

We continue with the definition of a set of abstract objects A_k, a transformation $h_{k,\Pi} : \mathrm{NL}_k \to A_k$, and a problem $\Delta_\Pi : A_k \to \mathbb{N}_0$. For each cut $C = (G, V)$ of a k-NLC graph G, let $f_C = (a_1, \ldots, a_k, b_1, \ldots, b_k, c)$ be a $(2 \cdot k + 1)$-tuple of nonnegative integers. The integer a_i is the number of vertices in V labeled with i, the integer b_i is the number of vertices in $V_G - V$ labeled with i, and the integer c is the cut-size of C. Let f_G be the set of all f_C, where C is a cut of G.

We define

$$\begin{array}{lcl} A_k & := & 2^{(\mathbb{N}_0^{2 \cdot k+1})} \\ h_{k,\Pi}(G) & := & f_G \\ \Delta_\Pi(X) & := & \max\limits_{(a_1,\ldots,a_k,b_1,\ldots,b_k,c) \in X} c \end{array}$$

The next lemma shows that the SIMPLE MAX-CUT PROBLEM is P-recognizable with respect to the class of all k-NLC graphs.

Lemma 1: Let Π be the SIMPLE MAX-CUT PROBLEM and A_k, $h_{k,\Pi}$, and Δ_Π as defined above. Then, for all $G, J \in \mathrm{NL}_k$ and all $t \in [k]$, $R : [k] \to [k]$, $S \subseteq [k]^2$:

1. The size of $h_{k,\Pi}(G)$ is polynomial in the size of G.

2. There is a P-time operation $\circ_{\Pi,R} : A_k \to A_k$ with

$$h_{k,\Pi}(\circ_R(G)) = \circ_{\Pi,R}(h_{k,\Pi}(G)).$$

3. There is a P-time operation $\times_{\Pi,S} : A_k \times A_k \to A_k$ with

$$h_{k,\Pi}(G \times_S J) = h_{k,\Pi}(G) \times_{\Pi,S} h_{k,\Pi}(J).$$

4. Δ_Π is a P-time mapping with

$$\Delta_\Pi(h_{k,\Pi}(G)) = \Pi(unlab(G)).$$

Proof:

1. Let $C = (G, V)$ be a cut of a k-NLC graph G. Each f_C has a size in $O((2 \cdot k + 1) \cdot \log(|V_G|))$. Since there are at most $|V_G|^{2 \cdot k} \cdot |E_G|$ different systems and k is constant, the size of f_G $(= h_{k,\Pi}(G))$ is polynomial in the size of G.

2. The operation $\circ_{\Pi,R}$ is defined by the transformation $h_{k,\Pi}$. That is,

$$\circ_{\Pi,R}(h_{k,\Pi}(G)) = h_{k,\Pi}(\circ_R(G)) = f_{\circ_R(G)}.$$

   The set $f_{\circ_R(G)}$ is computable in polynomial time in the size of f_G, because there is a system

$$(a_1, \ldots, a_k, b_1, \ldots, b_k, c) \in f_G$$

   if and only if there is a system

$$(a'_1, \ldots, a'_k, b'_1, \ldots, b'_k, c) \in f_{\circ_R(G)}$$

   such that

$$a'_i = \begin{cases} 0 & \text{if } R^{-1}(i) = \emptyset \\ \displaystyle\sum_{j \in R^{-1}(i)} a_j & \text{else} \end{cases}$$

   and

$$b'_i = \begin{cases} 0 & \text{if } R^{-1}(i) = \emptyset \\ \displaystyle\sum_{j \in R^{-1}(i)} b_j & \text{else} \end{cases}$$

   for $i = 1, \ldots, k$. Here, $R^{-1}(i) = \{j \in [k] | R(j) = i\}$.

3. Let $H = G \times_S J$. For each cut (H, W) of H there must be a cut (G, V) of G and a cut (J, U) of J such that $W = V \cup U$. This implies, that f_H is the set of all systems $(a_1, \ldots, a_k, b_1, \ldots, b_k, c)$ defined by a system

$$(a'_1, \ldots, a'_k, b'_1, \ldots, b'_k, c') \in f_G$$

and a system

$$(a''_1, \ldots, a''_k, b''_1, \ldots, b''_k, c'') \in f_J$$

such that $a_i = a'_i + a''_i$ and $b_i = b'_i + b''_i$ for $i = 1, \ldots, k$ and

$$c = c' + c'' + \sum_{(x,y) \in S} a'_x \cdot b''_y + b'_x \cdot a''_y.$$

Intuitively speaking, the cut-size of (H, W) is the cut-size of (G, V) plus the cut-size of (J, U) plus the number of edges between the vertices from V and $V_J - U$ plus the number of edges between the vertices from $V_G - V$ and U.

The computation of all these systems takes polynomial time in the size of f_G and f_J.

4. f_G has a system for each cut C of G and thus contains the cut-size of each cut of G. □

Corollary 1: Let Π be the SIMPLE MAX-CUT PROBLEM. For each k-NLC graph G, $\Pi(G)$ is computable in polynomial time in the size of G if G is given by its composition.

5 The Hamiltonian circuit property

As a second example, we will show that the HAMILTONIAN CIRCUIT PROPERTY is P-recognizable with respect to the class of all k-NLC graphs and thus decidable in polynomial time for k-NLC graphs.

We need the following important notations: A sequence $(v_1, \ldots, v_m)$ of m pairwise distinct vertices of a graph G is called a *path of length* m if $\{v_i, v_{i+1}\}$ is an edge in G for $i = 1, \ldots, m-1$. The vertices v_1 and v_m are called *end vertices* of the path. If, additionally, $\{v_1, v_m\}$ is an edge in G and m is greater than 2, then $(v_1, \ldots, v_m)$ is called a *circuit* of length m. A graph G has the HAMILTONIAN CIRCUIT PROPERTY if G has a circuit of length $|V_G|$. This property is NP-complete in general, see problem GT37 in [GJ79].

A similar example as the one in Section 4 shows that the HAMILTONIAN CIRCUIT PROPERTY is not 1-recognizable. Let G and J be two totally disconnected 1-NL graphs with $1 < |V_G| < |V_J|$. Then, for each transformation $h_{k,\Pi}$, we have:

$$h_{k,\Pi}(G) \neq h_{k,\Pi}(J)$$

because $G \times_{\{(1,1)\}} G$ has a Hamiltonian circuit and $G \times_{\{(1,1)\}} J$ not.

We continue with some further notations. Let $P = \{p_1, \ldots, p_n\}$ be a set of n vertex disjoint paths of a k-NLC graph G such that each vertex of G is contained exactly in one path of P. Let f_P be the system $(a_{1,1}, \ldots, a_{k,k})$ of $k \cdot (k+1)/2$ nonnegative integers $a_{i,j}$ for $1 \leq i \leq j \leq k$. The integer $a_{i,j}$ is the number of paths in P that have one end vertex labeled with i and one end vertex labeled with j. Let f_G be the set of all f_P, where P is a set of paths of G such that each vertex of G is contained exactly in on path of P.

Now, we define a set of abstract objects A_k, a transformation $h_{k,\Pi} : \mathrm{NL}_k \to A_k$, and a problem $\Delta_\Pi : A_k \to \{true, false\}$.

$$\begin{array}{lcl} A_k & := & 2^{(\mathbb{N}_0^{k\cdot(k+1)/2})} \times \{true, false\} \times \mathbb{N} \\ h_{k,\Pi}(G) & := & (f_G, \Pi(unlab(G)), |V_G|) \\ \Delta_\Pi((X, x, y)) & := & x \end{array}$$

The next lemma shows that the HAMILTONIAN CIRCUIT PROPERTY is P-recognizable with respect to the class of all k-NLC graphs.

Lemma 2: Let Π be the HAMILTONIAN CIRCUIT PROPERTY and A_k, $h_{k,\Pi}$, and Δ_Π as defined above. Then, for all $G, J \in \mathrm{NL}_k$ and all $t \in [k]$, $R : [k] \to [k]$, $S \subseteq [k]^2$:

1. The size of $h_{k,\Pi}(G)$ is polynomial in the size of G.
2. There is a P-time operation $\circ_{\Pi,R} : A_k \to A_k$ with
$$h_{k,\Pi}(\circ_R(G)) = \circ_{\Pi,R}(h_{k,\Pi}(G)).$$
3. There is a P-time operation $\times_{\Pi,S} : A_k \times A_k \to A_k$ with
$$h_{k,\Pi}(G \times_S J) = h_{k,\Pi}(G) \times_{\Pi,S} h_{k,\Pi}(J).$$
4. Δ_Π is a P-time mapping with
$$\Delta_\Pi(h_{k,\Pi}(G)) = \Pi(unlab(G)).$$

Proof:

1. Each f_P has a size in $O(k \cdot (k+1)/2 \cdot \log(|V_G|))$. There are at most $|V_G|^{k\cdot(k+1)/2}$ different systems. If k is constant, then $h_{k,\Pi}(G)$ is polynomial in the size of G.
2. Since $\circ_{\Pi,R}$ is defined by $h_{k,\Pi}$, we have
$$\circ_{\Pi,R}(h_{k,\Pi}(G)) = h_{k,\Pi}(\circ_R(G)) = (f_{\circ_R(G)}, \Pi(unlab(\circ_R(G))), |V_{\circ_R(G)}|).$$
The result of Π is equal for $unlab(G)$ and $unlab(\circ_R(G))$. The number of vertices in G and $\circ_R(G)$ are also equal. The set $f_{\circ_R(G)}$ is computable in polynomial time, because
$$(a_{1,1}, \ldots, a_{k,k}) \in f_{\circ_R(G)}$$
if and only if there is a system
$$(b_{1,1}, \ldots, b_{k,k}) \in f_G$$
such that
$$a_{i,j} = \begin{cases} 0 & \text{if } R^{-1}(i) = \emptyset \vee R^{-1}(j) = \emptyset \\ \displaystyle\sum_{i' \in R^{-1}(i), j' \in R^{-1}(j)} b_{i',j'} & \text{else} \end{cases}$$
for all $1 \le i \le j \le k$.

3. Let $H = G \times_S J$ be a k-NLC graph that has a Hamiltonian circuit. If we delete all edges in G and J that are not used for the Hamiltonian circuit of H, we get two k-NLC graphs G' and J'. Both consist of paths. Each vertex in V_G and V_J is in exactly one path of G' and J', respectively. Let P and Q be the sets of all paths in G' and J', then $f_P \in f_G$ and $f_Q \in f_J$. On the other hand, H has a Hamiltonian circuit if and only if there is a set of paths P and Q with $f_P \in f_G$, $f_Q \in f_J$ and $G' \times_S J'$ has a Hamiltonian circuit, where G' and J' are the graphs representing the paths P and Q, respectively.

   Let $(f_G, x_G, y_G) = h_{k,\Pi}(G)$, $(f_J, x_J, y_J) = h_{k,\Pi}(J)$, and $(f_H, x_H, y_H) = h_{k,\Pi}(G \times_S J)$. We can compute all paths in H by concatenating paths from G with paths from J.

   We need the following data structure. Let (A, B, C) be a triple, where

$$\begin{array}{lll} A = (a_{1,1}, \ldots, a_{i,j}, \ldots, a_{k,k}) & \text{for} & 1 \le i \le j \le k \\ B = (b_{1,1}, \ldots, b_{i,j}, \ldots, b_{k,k}) & \text{for} & 1 \le i, j \le k \\ C = (c_{1,1}, \ldots, c_{i,j}, \ldots, c_{k,k}) & \text{for} & 1 \le i \le j \le k \end{array}$$

   are systems of nonnegative integers. Note that A and C have $k \cdot (k+1)/2$ integers and B has k^2 integers. The integers in A, B, and C will be used to represent the number of paths that have both end vertices in G, one end vertex in G and one end vertex in J, or both end vertices in J, respectively. For instance, $b_{i,j}$ is the number of paths that have one end vertex from G labeled with i and one end vertex from J labeled with j.

   The computation of f_H starts with the set of all triples (A, B, C), where $A \in f_G$, $b_{i,j} = 0$ for $1 \le i, j \le k$, and $C \in f_J$. There are three ways to create a new triple. We take a copy of a triple (A, B, C) and

   (a) concatenate a path represented by an integer in A with a path represented by an integer in C to create a path represented by an integer in B,

   (b) concatenate a path represented by an integer in A with a path represented by an integer in B to create a path represented by an integer in A, or

   (c) concatenate a path represented by an integer in C with a path represented by an integer in B to create a path represented by an integer in C.

   Each case has some sub-cases. A complete summary depicts the following table.

case	sub-case	increment	decrement
(a)	$a_{i,j} > 0$, $c_{n,m} > 0$, $(i,n) \in S$	$b_{j,m}$	$a_{i,j}$, $c_{n,m}$
	$a_{i,j} > 0$, $c_{n,m} > 0$, $(i,m) \in S$	$b_{j,n}$	$a_{i,j}$, $c_{n,m}$
	$a_{i,j} > 0$, $c_{n,m} > 0$, $(j,n) \in S$	$b_{i,m}$	$a_{i,j}$, $c_{n,m}$
	$a_{i,j} > 0$, $c_{n,m} > 0$, $(j,m) \in S$	$b_{i,n}$	$a_{i,j}$, $c_{n,m}$
(b)	$a_{i,j} > 0$, $b_{n,m} > 0$, $(i,m) \in S$, $j \leq n$	$a_{j,n}$	$a_{i,j}$, $b_{n,m}$
	$a_{i,j} > 0$, $b_{n,m} > 0$, $(i,m) \in S$, $j > n$	$a_{n,j}$	$a_{i,j}$, $b_{n,m}$
	$a_{i,j} > 0$, $b_{n,m} > 0$, $(j,m) \in S$, $i \leq n$	$a_{i,n}$	$a_{i,j}$, $b_{n,m}$
	$a_{i,j} > 0$, $b_{n,m} > 0$, $(j,m) \in S$, $i > n$	$a_{n,i}$	$a_{i,j}$, $b_{n,m}$
(c)	$c_{i,j} > 0$, $b_{n,m} > 0$, $(n,i) \in S$, $j \leq m$	$c_{j,m}$	$c_{i,j}$, $b_{n,m}$
	$c_{i,j} > 0$, $b_{n,m} > 0$, $(n,i) \in S$, $j > m$	$c_{m,j}$	$c_{i,j}$, $b_{n,m}$
	$c_{i,j} > 0$, $b_{n,m} > 0$, $(n,j) \in S$, $i \leq m$	$c_{i,m}$	$c_{i,j}$, $b_{n,m}$
	$c_{i,j} > 0$, $b_{n,m} > 0$, $(n,j) \in S$, $i > m$	$c_{m,i}$	$c_{i,j}$, $b_{n,m}$

The table above has to be read as follows: If the sub-case condition is true, then increment the number in the "increment" column by one to create a new path and decrement the two integers in the "decrement" column by one to delete the two old paths. The set of all possible triples can be constructed by a successive application of the instructions in the table. This takes polynomial time, because the size of each triple (A,B,C) is in $O(k \cdot (2 \cdot k+1) \cdot \log(|V_H|))$ and there are at most $|V_H|^{k \cdot (2 \cdot k+1)}$ many different triples. If the set of all generated triples does not change by any further path composition, we unite all paths of each triple to get a system for f_H. That is,

$$(d_{1,1}, \ldots, d_{k,k}) \in f_H$$

if and only if we have found a triple (A,B,C) such that $d_{i,j} = a_{i,j} + b_{i,j} + b_{j,i} + c_{i,j}$ for all $1 \leq i \leq j \leq k$. This completes the computation of f_H.

H has a Hamiltonian circuit if and only if we have found a triple (A,B,C) and there are two integers i,j such that $b_{i,j} = 1$, all other integers in A, B, and C are zero, $(i,j) \in S$, and $|V_H| > 2$, where $|V_H| = |V_G| + |V_J|$.

4. By the definition of $h_{k,\Pi}$ and Δ_Π. □

Corollary 2: Let Π be the HAMILTONIAN CIRCUIT PROPERTY. For each k-NLC graph G, $\Pi(G)$ is computable in polynomial time in the size of G if G is given by its composition.

A simple modification of the data structures used in the proof of Lemma 2.3 shows that the problem to compute the length of a longest path or longest circuit is also P-recognizable. Additionally, we have to store the length of the paths we have already found.

6 Conclusion

We have defined two new classes of recursively defined graphs, called k-NLC graphs and k-NLC trees. Furthermore, we have introduced a very simple method for the design of

polynomial algorithms on tree-structured graphs for difficult graph problems (NP-complete or worse). We have exemplified our method with the SIMPLE MAX-CUT PROBLEM and the HAMILTONIAN CIRCUIT PROPERTY for k-NLC graphs. The second example is already shown for co-graphs by Corneil et al.

Of course, a solution is usually obtained by intuition and hard work. Notice also that the algorithms are polynomial in the size of the problem instance, but exponential in k. Thus, it seems to be worth looking for very efficient operations $\circ_{\Pi,R}$ and $\times_{\Pi,S}$ and a very low space bounded transformation $h_{k,\Pi}$. Then, the graph problem can be solved in a reasonable amount of time and space in many practical applications.

Another interesting way to define classes of graphs is by graph grammars. For instance, for each positive integer k there is a *hyperedge replacement system* [HK87] whose language is the set of all partial k-trees. However, we do not know any already defined rewriting system whose language is exactly the set of all k-NLC graphs or k-NLC trees. Rewriting systems whose languages are similar to k-NLC graphs are certain versions of *node label controlled* (NLC) *graph grammars* and *neighbourhood controlled embedding* (NCE) *graph grammars*, see [JR80a, JR80b, JR82].

References

[ACP87] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Algebraic and Discrete Methods*, 8(2):227–284, April 1987.

[ALS91] S. Arnborg, J. Lagergren, and D. Seese. Problems easy for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

[BLW87] M.W. Bern, E.L. Lawler, and A.L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *Journal of Algorithms*, 8:216–235, 1987.

[Bod88] H.L. Bodlaender. Dynamic programming on graphs with bounded tree-width. In T. Lepistö and A. Salomaa, editors, *Proceedings of International Colloquium on Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 105–118. Springer-Verlag, Berlin/New York, 1988.

[Bod93] H.L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 226–234, 1993.

[Bra90] A. Brandstädt. Special graph classes—a survey. Technical Report 90/3/4/5/6, Fridrich-Schiller-Universität Jena, Jena, FRG, 1990.

[BU84] M. Burlet and J.P. Uhry. Parity graphs. *Annals of Discrete Mathematics*, 21:253–277, 1984.

[Cou87] B. Courcelle. An axiomatic definition of context-free rewriting and its application to NLC graph grammars. *Theoretical Computer Science*, 55:141–181, 1987.

[CPS84] D.G. Corneil, Y. Perl, and L.K. Stewart. Cographs: recognition, applications, and algorithms. In *Proceedings of 15th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, 1984.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[HK87] A. Habel and H.J. Kreowski. May we introduce to you: Hyperedge replacement. In H. Ehrig, M. Nagl, A. Rosenfeld, and G. Rozenberg, editors, *Graph-Grammars and Their Application to Computer Science*, volume 291 of *Lecture Notes in Computer Science*, pages 15–26. Springer-Verlag, Berlin/New York, 1987.

[JR80a] D. Janssens and G. Rozenberg. On the structure of node label controlled graph languages. *Information Science*, 20:191–216, 1980.

[JR80b] D. Janssens and G. Rozenberg. Restrictions, extensions, and variations of NLC grammars. *Information Science*, 20:217–244, 1980.

[JR82] D. Janssens and G. Rozenberg. Graph grammars with neighbourhood-controlled embedding. *Theoretical Computer Science*, 21:55–74, 1982.

[Ros74] D.J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7:317–322, 1974.

[RS86] N. Robertson and P.D. Seymour. Graph minors II. Algorithmic aspects of tree width. *Journal of Algorithms*, 7:309–322, 1986.

[Sum74] P.D. Sumner. Dacey graphs. *Journal of Aust. Soc.*, 18:492–502, 1974.

[Wan94] E. Wanke. k-NLC Graphs and Polynomial Algorithms. *Discrete Appl. Math.*, 54:251-266, 1994.

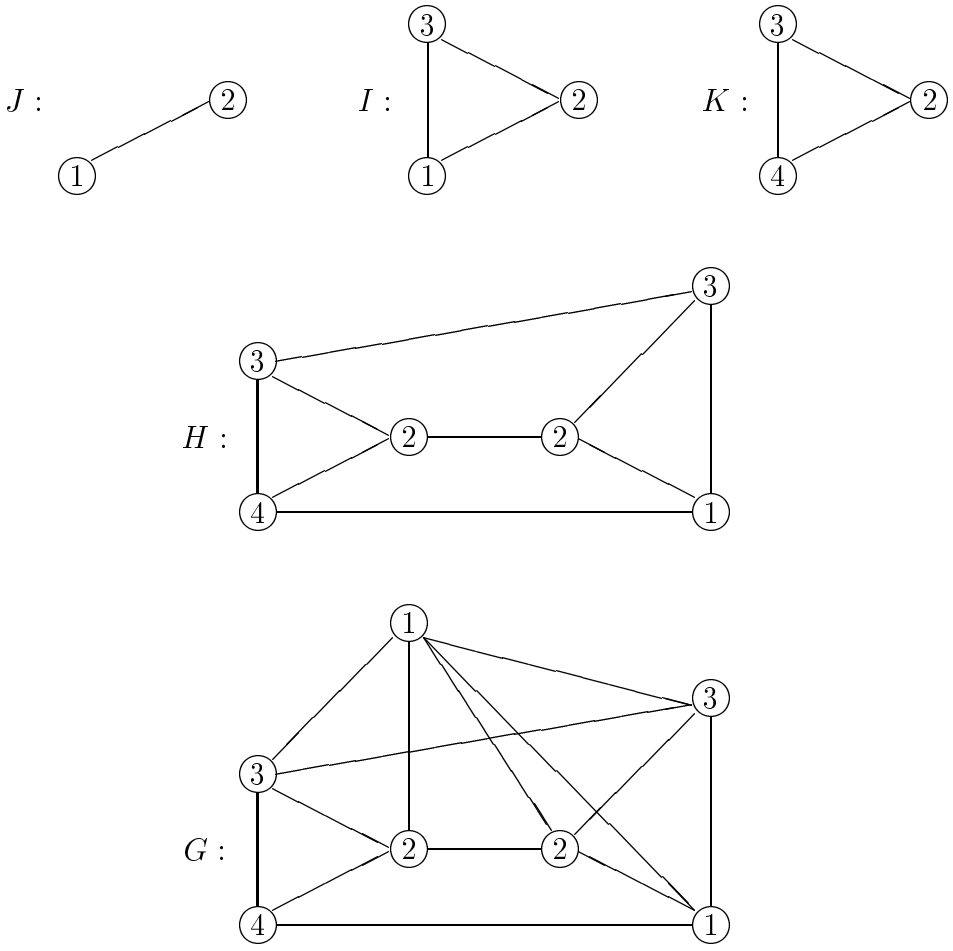

Figure 1: The composition of a 4-NLC graph G. The numbers in the cycles are the labels of the vertices.

$$
\begin{aligned}
G &= H \times_{\{(1,1),(2,1),(3,1)\}} \bullet_1 \\
H &= K \times_{\{(4,1),(2,2),(3,3)\}} I \\
K &= \circ_{\{(1,4),(2,2),(3,3),(4,4)\}}(I) \\
I &= J \times_{\{(1,3),(2,3)\}} \bullet_3 \\
J &= \bullet_1 \times_{\{(1,2)\}} \bullet_2
\end{aligned}
$$

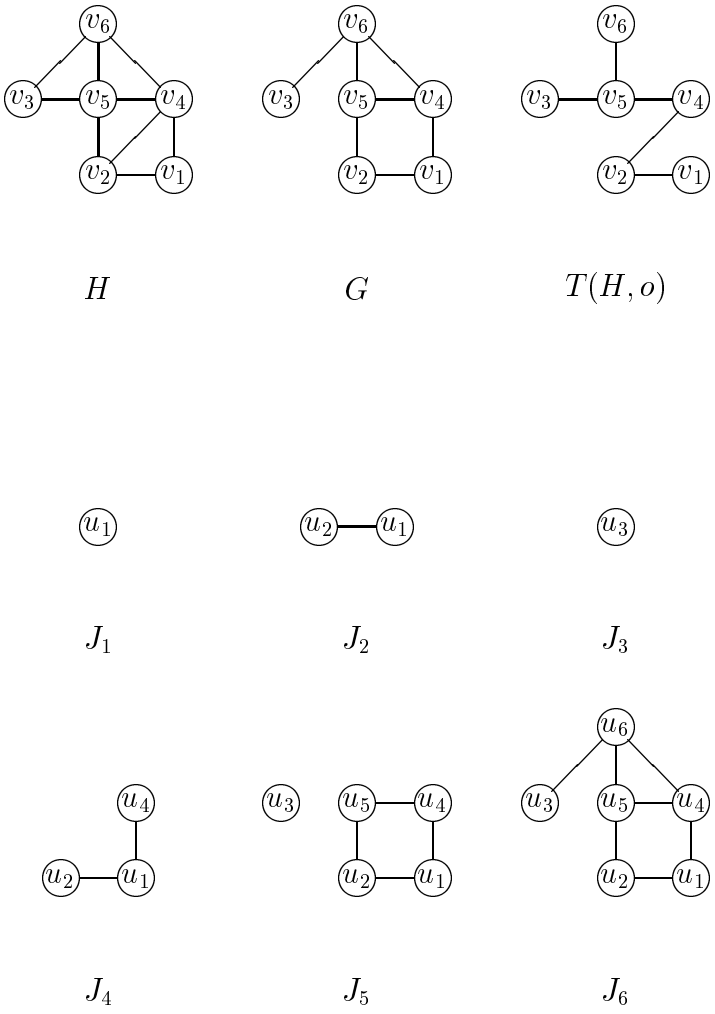

Figure 2: A 2-tree H with PEO $(v_1, \ldots, v_6)$, a partial 2-tree G with $V_G = V_H$, the tree $T(H, o)$, and the 7-NLC graphs $J_1, \ldots, J_6$. The vertices in H, G, and $J_1, \ldots, J_6$ are colored as follows:

	$col(v_i)$	C_i	$b^{-1}(lab(u_i))$					
i	H	G	J_1	J_2	J_3	J_4	J_5	J_6
1	1	$\{2,3\}$	$\{2,3\}$	$\{3\}$		$\{\}$	$\{\}$	$\{\}$
2	2	$\{2\}$		$\{2\}$		$\{2\}$	$\{\}$	$\{\}$
3	3	$\{1\}$			$\{1\}$		$\{1\}$	$\{\}$
4	3	$\{1,2\}$				$\{1,2\}$	$\{1\}$	$\{\}$
5	2	$\{1\}$					$\{1\}$	$\{\}$
6	1	$\{\}$						$\{\}$