

University of Leeds  
**SCHOOL OF COMPUTING**  
**RESEARCH REPORT SERIES**  
Report 2008.07

**On a Disparity Between Relative Cliquewidth and Relative NLC-width**

by

**Haiko Müller & Ruth Uerner**

December 2008

## Abstract

Cliquewidth and NLC-width are two closely related parameters that measure the complexity of graphs. Both clique- and NLC-width are defined to be the minimum number of labels required to create a labelled graph by certain terms of operations. Many hard problems on graphs become solvable in polynomial time if the inputs are restricted to graphs of bounded clique- or NLC-width. Cliquewidth and NLC-width differ at most by a factor of two.

The relative counterparts of these parameters are defined to be the minimum number of labels necessary to create a graph while the tree-structure of the term is fixed. We show that RELATIVE CLIQUEWIDTH and RELATIVE NLC-WIDTH differ significantly in computational complexity. While the former problem is NP-complete the latter is solvable in polynomial time. The relative NLC-width can be computed in  $\mathcal{O}(n^3)$  time, which also yields an exact algorithm for computing the NLC-width in time  $\mathcal{O}(3^n n)$ . Additionally, our technique enables a combinatorial characterisation of NLC-width that avoids the usual operations on labelled graphs.

## 1 Introduction

Treewidth is an important graph-parameter, which is useful in many ways regarding algorithms for graphs, and has therefore been thoroughly studied for decades now. One of the most important results in this context is that every problem which is expressible in Monadic Second Order Logic (MSO for short) is solvable in linear time when restricted to graphs of bounded treewidth. This includes NP-complete problems such as 3-colourability and hamiltonicity. However, graphs which contain large cliques have large treewidth and the MSO-result does therefore not apply to classes containing such graphs.

Cliquewidth, introduced in [3], is an alternative graph-parameter that generalises treewidth in the sense that every class of graphs that has bounded treewidth has also bounded cliquewidth, but not *vice versa* as complete graphs have cliquewidth at most 2. There is an analogous, slightly weaker MSO-result for graphs of bounded cliquewidth [4], which shows that cliquewidth is interesting in this respect as well.

Cliquewidth (denoted by  $\text{cwd}$ ) is defined as the minimum number of labels necessary in order to construct a given graph from the following four operations:  $\bullet_i$  creates a single vertex labelled  $i$ ,  $\rho_{i \rightarrow j}$  relabels all vertices labelled by  $i$  to  $j$ ,  $\eta_{i,j}$  adds edges between vertices labelled  $i$  and  $j$  (where  $i \neq j$ ) and  $\oplus$  creates the disjoint union of two graphs.

The NLC-width  $\text{nlc}(G)$  of a graph  $G$  was introduced in [15] by a slightly different set of operations. From the algorithmic point of view, both parameters are equivalent, because  $\text{nlc}(G) \leq \text{cwd}(G) \leq 2 \cdot \text{nlc}(G)$  holds for all graphs  $G$  [10].

Considerable efforts have been made to investigate the complexity of computing the cliquewidth and the NLC-width. Although conjectured for a long time, NP-completeness results were obtained only in 2005 by FELLOWS *et al.* [7] and by GURSKI and WANKE [9]. For fixed  $k$  there is a polynomial-time algorithm by OUM and SEYMOUR [14] that either decides  $\text{cwd}(G) \geq k$  or certifies  $\text{cwd}(G) \leq 2^{3k+2} - 1$ . There are polynomial time algorithms known to recognise graphs of cliquewidth at most three [1] and NLC-width at most two [11]. Yet, it is still open whether or not there exists a polynomial time algorithm to check  $\text{cwd}(G) \leq k$  for any fixed  $k \geq 4$ .

With regard to these difficulties several restrictions of cliquewidth have been introduced. GURSKI and WANKE investigated the sequential cliquewidth [8], see [12] too, which then

became crucial in proving the NP-completeness result [7]. COURCELLE and TWIGG introduce the notion of  $m$ -cliquewidth in [2] to construct efficient routing schemes. LOZIN and RAUTENBACH defined in [12] the relative cliquewidth in order to exhibit the tree-structure of  $k$ -expressions and thus stress similarities with other graph-parameters like branchwidth or rankwidth. They also provide an algorithm to approximate the relative cliquewidth in polynomial time. We define the relative NLC-width analogously.

In this paper we will show how to compute the relative NLC-width of a graph exactly in polynomial time. In contrast we obtain an NP-completeness result for the relative cliquewidth.

**Theorem 1.** *The relative NLC-width of a graph on  $n$  vertices can be computed in time  $\mathcal{O}(n^3)$  while the problem RELATIVE CLIQUEWIDTH is NP-complete.*

This result shows that the hard part of an NLC-width computation consists in finding the right tree-structure. Yet, in order to compute the cliquewidth of a graph efficiently it is not sufficient to know the optimal tree-structure. Our technique admits even more structural insight. For treewidth many characterisations are known, including partial  $k$ -trees or embeddings into chordal graphs of bounded clique-size. However, until now, there has been no definition of cliquewidth or NLC-width that avoids the operations used on labelled graphs. Our results yield such a characterisation for NLC-width, which might be useful in further investigations on this parameter.

Furthermore, we obtain an exact algorithm to compute the NLC-width in exponential time:

**Theorem 2.** *The NLC-width of a graph on  $n$  vertices can be computed in time  $\mathcal{O}(3^n n)$ .*

We show that this timebound can be improved for sequential terms and also obtain a timebound for the sequential cliquewidth. To give a brief overview on the structure of the paper: Section 2 recalls the definitions of NLC-width and provides further notation. In Section 4 we develop the tools to characterize the relative NLC-width of a graph and show how to compute the relative NLC-width in polynomial time. The following Section 5 contains the algorithms for general and sequential NLC-width. In the short Section 6 we characterize NLC-width without operations. Finally, Section 7 is devoted to the NP-completeness proof for the computation of the relative cliquewidth.

## 2 Preliminaries

We will first introduce some basic notions and explain what NLC-width and relative NLC-width are. A *graph* is a pair  $G = (V(G), E(G))$  where  $V(G)$  is the set of vertices and  $E(G)$  the set of edges and each edge is a two-element subset of  $V(G)$ . We will always suppose our graphs to be finite, undirected and without multiple edges or loops. For convenience we also use  $V$  and  $E$  instead of  $V(G)$  and  $E(G)$ .

For a vertex  $v$  of  $G = (V, E)$  let  $N(u) = \{v \mid \{u, v\} \in E\}$  denote its (open) *neighbourhood*. The *closed neighbourhood* of  $u$  is  $N[u] = \{u\} \cup N(u)$ . We extend these notions to sets  $U \subseteq V$  by setting  $N[U] = \bigcup_{u \in U} N[u]$  and  $N(U) = N[U] \setminus U$ . Moreover,  $G[U]$  denotes the subgraph of  $G$  induced by the vertices of  $U \subseteq V$ .

## 2.1 NLC-width

A *labelled graph* is a triple  $(V, E, \lambda)$  such that  $(V, E)$  is a graph and  $\lambda : V \rightarrow \mathbb{N}$  is a function that defines a labelling of the vertices. We describe labelled graphs by terms. Every term  $t$  defines a labelled graph which is denoted by  $\text{val}(t)$ . Instead of  $\text{val}(t) = (V, E, \lambda)$  we also write  $V = V_t$ ,  $E = E_t$  and  $\lambda = \lambda_t$ .

Since we consider only finite graphs, the range of each labelling  $\lambda$  is finite too. If it is a subset of  $[k] = \{1, 2, \dots, k\}$  then the labelled graph is called  $k$ -graph. Similarly, a  $k$ -term uses labels in  $[k]$  only. In line with the countable set  $\mathbb{N}$  of potential labels, we assume a countable set of potential vertices, from which we can choose actual vertices one by one.

The set NLC of *terms* is recursively defined as follows:

**create:** For all labels  $i$  and all vertices  $v$ , there is a term  $\bullet_i(v)$  with  $\text{val}(\bullet_i(v)) = (\{v\}, \emptyset, \lambda)$ , where  $\lambda(v) = i$ .

**join:** For all terms  $p, q \in \text{NLC}$  with  $V_p \cap V_q = \emptyset$  and all relations  $S \subseteq \mathbb{N} \times \mathbb{N}$ ,  $p \times_S q$  is a term, and  $\text{val}(p \times_S q) = (V_p \cup V_q, E_p \cup E_q \cup F, \lambda)$ , where  $F = \{\{u, w\} \mid u \in V_p, w \in V_q, (\lambda_p(u), \lambda_q(w)) \in S\}$ ,  $\lambda(u) = \lambda_p(u)$  for  $u \in V_p$  and  $\lambda(w) = \lambda_q(w)$  for  $w \in V_q$ .

**relabel:** For all terms  $s \in \text{NLC}$  and all functions  $R : \mathbb{N} \rightarrow \mathbb{N}$ ,  $\circ_R(s)$  is a term with  $\text{val}(\circ_R(s)) = (V_s, E_s, R \circ \lambda_s)$ .

**otherwise:** There are no other terms.

By  $\text{NLC}_k$  we denote the set of  $k$ -terms. The *NLC-width* of a labelled graph  $G = (V, E, \lambda)$ , is the minimum  $k$  such that there is a  $k$ -term  $t$  with  $\text{val}(t) = G$ . For an unlabelled graph  $G = (V, E)$  we define the *NLC-width*  $\text{nlc}(G)$  of  $G$  by

$$\text{nlc}(G) = \min\{k \mid \exists t \in \text{NLC}_k \exists \lambda : V \rightarrow [k] . \text{val}(t) = (V, E, \lambda)\}.$$

Among others, the minimum is attained for a constant function  $\lambda$ .

## 2.2 Relative NLC-width

The construction of a graph by a term in  $t \in \text{NLC}$  corresponds to a rooted binary *parse tree*, which is denoted as  $\text{tree}(t)$ . The leaves of  $\text{tree}(t)$  correspond to the creations, parents of two children correspond to joins and inner nodes with one child correspond to relabellings. We will often identify a term in NLC with its parse tree, and operations with nodes.

The reduced tree  $\text{red}(T)$  of a parse tree  $T$  is obtained from  $T$  by contracting edges incident to relabellings. Thereby we forget all information about labels, but remember the vertices in the leaves of  $T$  in  $\text{red}(T)$ . Hence, the reduced tree of  $\text{tree}(t)$  is a rooted binary tree with leaves in one-to-one correspondence to the vertices of  $\text{val}(t)$ . A *reduced tree* of a graph  $G$  is any rooted binary tree with leaves labelled by the vertices of  $G$  in this way. Figure 1 illustrates this notion. Subtrees of  $\text{tree}(t)$  correspond in a natural way to subterms of  $t$ .

**Observation 3.** *Let  $s$  be any subterm of  $t \in \text{NLC}$  with  $G = (V_t, E_t)$ . Then  $(V_s, E_s) = G[V_s]$ .*

If  $c$  is a node of a reduced tree  $R$  of a graph  $G$  we set  $V_c \subseteq V(G)$  to be the subset of vertices that are referred to in the leaves below  $c$  in  $R$ .

A subterm  $p$  of  $t$  is called *principal subterm* if  $p = t$  or  $p$  is the child of a join. A term is *sequential* if every join in  $t$  has one child that is a create. The *sequential NLC-width*

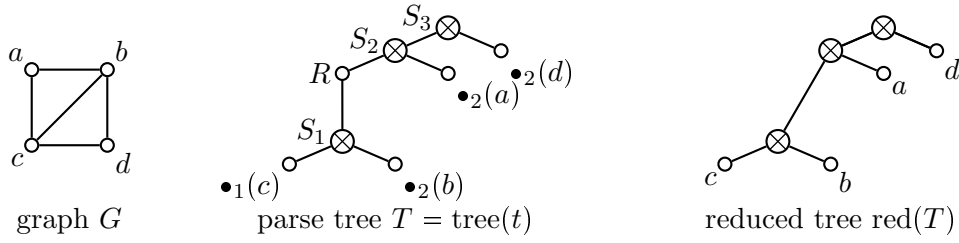


Figure 1: For the 2-term  $t = ((\circ_R(\bullet_1(c) \times_{S_1} \bullet_2(b))) \times_{S_2} \bullet_2(a)) \times_{S_3} \bullet_2(d) \in \text{NLC}$  where  $S_i = \{(1, 2)\}$  for  $i = 1, 2, 3$  and  $R : [2] \rightarrow [2]$  with  $1 \mapsto 1$  and  $2 \mapsto 1$ , we have  $\text{val}(t) = (G, \lambda)$ , where  $\lambda$  maps  $a \mapsto 2$ ,  $b \mapsto 1$ ,  $c \mapsto 1$  and  $d \mapsto 2$ .

$\text{s-nlc}(G)$  of a graph  $G$  differs from the ordinary NLC-width by the additional condition that  $t$  is sequential, see [7] for an analogous definition of sequential cliquewidth. Similarly, the *relative NLC-width*  $\text{r-nlc}(G, R)$  of a graph  $G = (V, E)$  with respect to a reduced tree  $R$  of  $G$  has the additional constraint  $\text{red}(\text{tree}(t)) = R$ , compare [12].

$$\text{r-nlc}(G, R) = \min\{k \mid \exists t \in \text{NLC}_k \exists \lambda : V \rightarrow [k] . \text{val}(t) = (V, E, \lambda) \wedge \text{red}(\text{tree}(t)) = R\}$$

### 3 Atoms

Here we provide a simple lower bound on the NLC-width of graphs.

Let  $G = (V, E)$  be any graph and  $U \subseteq V$ . We define an equivalence relation  $\sim$  on  $U$  by

$$u_1 \sim u_2 \iff N(u_1) \setminus U = N(u_2) \setminus U.$$

The equivalence classes of this relation form a partition  $\mathcal{A}(U)$  of  $U$  into *atoms*. By  $A(U, v)$  we denote the equivalence class of  $v \in U$ , i.e. the atom  $A \in \mathcal{A}(U)$  with  $v \in A$ , while  $a(U)$  denotes the number of atoms the set  $U$  partitions into.

#### 3.1 Computing all Atoms

In this subsection we show how to compute the atoms  $\mathcal{A}(U)$  for all  $U \subseteq V$  in time  $\mathcal{O}(2^n n)$ . An outline of our algorithm is given in Table 1.

```

procedure caa( $V, E$ )
begin
   $\mathcal{A}(V) \leftarrow \{V\}$ ;
  for  $s \leftarrow |V| - 1$  downto 1 do
    for  $U \in \binom{V}{s}$  do
       $w \leftarrow \min(V \setminus U)$ ;
       $\mathcal{A}(U) \leftarrow \{A \cap N(w), A \setminus N[w] \mid A \in \mathcal{A}(U \cup \{w\})\} \setminus \{\emptyset\}$ 
    od od
end.

```

Table 1: Outline of algorithm caa

For a systematic time analysis we need more detail. Without loss of generality we assume  $V = \{0, 1, \dots, n-1\}$ . Then a subset  $U \subseteq V$  is represented by the number  $\sum_{u \in U} 2^u$ . With

this encoding we have  $v \in U$  if and only if bit  $v$  of  $U$  is 1, where the least significant bit is bit 0. Each atom is represented by its minimal element. We store all atoms in an array  $\text{atom}[1..(2^n - 1), 0..(n - 1)]$ , where  $\text{atom}(U, v) = \min(\mathcal{A}(U, v))$ . So  $u, v \in U$  are  $U$ -equivalent if and only if  $\text{atom}(U, u) = \text{atom}(U, v)$ , and for  $U \subseteq W$  we have  $\mathcal{A}(U, u) \subseteq \mathcal{A}(W, w)$  if and only if  $\text{atom}(W, \text{atom}(U, u)) = \text{atom}(W, w)$ .

<pre> <b>procedure</b> caa(<math>V, E</math>) <b>begin</b>   <b>for</b> <math>v \in V</math> <b>do</b> <math>\text{atom}[V, v] \leftarrow 0</math> <b>od</b>;   <b>for</b> <math>U \leftarrow V - 1</math> <b>downto</b> 1 <b>do</b>     <math>w \leftarrow 0</math>; <b>while</b> <math>w \in U</math> <b>do</b> <math>w \leftarrow w + 1</math> <b>od</b>;     <math>W \leftarrow U + 2^w</math>;     <b>for</b> <math>v \leftarrow 0</math> <b>to</b> <math>n - 1</math> <b>do</b> <math>b[v] \leftarrow n</math>; <math>c[v] \leftarrow n</math> <b>od</b>;     <b>for</b> <math>v \leftarrow 0</math> <b>to</b> <math>n - 1</math> <b>do</b> <math>a \leftarrow \text{atom}[W, v]</math>;       <b>if</b> <math>a = n</math> <b>or</b> <math>v = w</math> <b>then</b> <math>\text{atom}[U, v] \leftarrow n</math>         <b>else if</b> <math>\{v, w\} \in E</math>           <b>then if</b> <math>b[a] = n</math> <b>then</b> <math>b[a] \leftarrow v</math> <b>fi</b>;             <math>\text{atom}[U, v] \leftarrow b[a]</math>           <b>else if</b> <math>c[a] = n</math> <b>then</b> <math>c[a] \leftarrow v</math> <b>fi</b>;             <math>\text{atom}[U, v] \leftarrow c[a]</math>         <b>fi</b>       <b>fi</b>     <b>od od</b>   <b>end.</b> </pre>
---

Table 2: Details of algorithm `caa`

A detailed version of algorithm `caa` is given in Table 2. The vertices  $b[v]$  and  $c[v]$  denote  $\min(\mathcal{A}(W, v) \cap N(w))$  and  $\min(\mathcal{A}(W, v) \setminus N[w])$ , where  $W = U \cup \{w\}$ , and  $n$  means “undefined”.

**Lemma 4.** *All atoms of a graph can be computed in time  $\mathcal{O}(2^n n)$ .*

*Proof.* The outlined algorithm closely follows the definition of atoms and is therefore correct. In the detailed version we slightly changed the order in which we consider the subsets  $U$ . Note that the encoding of  $U \cup \{\min(V \setminus U)\}$  is greater than the encoding of  $U$  for all  $U \subset V$ .

To analyse the running time of `atom` we consider its detailed version. The first for-loop is executed  $n$  times and requires time  $\mathcal{O}(n)$  in total. We consider  $2^n - 2$  subsets  $U$  of  $V$  in the second for-loop. Inside there is a while-loop to compute  $w = \min(V \setminus U)$ , which requires time  $\mathcal{O}(n)$ , and two for-loops cycling through all  $v \in V$ . The bodies of both for-loops can be executed in constant time if the input graph is given by an adjacency matrix. Therefore `atom` runs in time  $\mathcal{O}(2^n n)$ .  $\square$

### 3.2 Canonical labelling

The following observation is recorded here for further reference.

**Observation 5.** *For all  $U, W \subseteq V(G)$  and for every atom  $A \in \mathcal{A}(U)$  there is an atom  $B \in \mathcal{A}(U \cup W)$  such that  $A \subseteq B$ .*

Two vertices cannot receive the same label under  $\lambda_s$  if there is a vertex outside  $V_s$  that distinguishes them. Hence we have

**Lemma 6.** *Let  $s$  be a subterm of  $t$ . Then the partition of  $V_s$  into classes of equal labels with respect to  $\lambda_s$  refines the partition of  $V_s$  into atoms.*

**Corollary 7.** *For each term  $t$  in  $\text{NLC}_k$  we have  $k \geq \max\{a(V_c) \mid c \in \mathbf{V}(\text{tree}(t))\}$ . In particular, if  $R$  is a reduced tree of  $G$  then we have  $\text{r-nlc}(G, R) \geq \max\{a(V_c) \mid c \in \mathbf{V}(R)\}$ .*

Let  $s$  be a subterm of a term  $t$  and  $G = (V_t, E_t)$ . Then  $\lambda_s$  is a *canonical labelling* of  $V_s$  if, for all  $u, v \in V_s$ ,  $\lambda_s(u) = \lambda_s(v)$  if and only if  $u$  and  $v$  are  $V_s$ -equivalent.

Lemma 6 means that the labellings that occur in NLC-terms are always refinements of canonical labellings. In the next subsection we will see that we can actually guarantee canonical labellings at certain points.

### 3.3 Normal form

Now we introduce a normal form for terms that reduces the number of labels as much as possible.

**Definition 8.** A term is in *normal form* if the labelling  $\lambda_p$  is canonical for every principal subterm  $p$  of  $t$ .

Two terms  $s, t \in \text{NLC}$  are *equivalent* if  $\text{val}(s) = \text{val}(t)$  and  $\text{red}(\text{tree}(s)) = \text{red}(\text{tree}(t))$ . The following observations illustrates the equivalence of terms.

**Observation 9.** *Let  $s'$  be a term equivalent to a subterm  $s$  of  $t \in \text{NLC}$ . Then the term  $t'$  obtained from  $t$  by substituting  $s'$  for  $s$  is equivalent to  $t$ .*

Let  $\pi$  be a permutation of  $[k]$ , i.e.  $\pi : [k] \rightarrow [k]$  is a bijection. For every term  $t \in \text{NLC}_k$  we define  $t\langle\pi\rangle$  recursively as follows:

**create:**  $(\bullet_i(v))\langle\pi\rangle = \bullet_{\pi(i)}(v)$

**join:**  $(p \times_S q)\langle\pi\rangle = p\langle\pi\rangle \times_T q\langle\pi\rangle$  where  $T = \{(\pi(i), \pi(j)) \mid (i, j) \in S\}$ .

**relabel:**  $(\circ_R(s))\langle\pi\rangle = \circ_P(s\langle\pi\rangle)$  where  $P = \pi \circ R \circ \pi^{-1}$ , i.e. for all  $l \in [k]$  we have  $P(l) = \pi(R(\pi^{-1}(l)))$ .

**Observation 10.** *For every permutation  $\pi$  of  $[k]$  and every  $t \in \text{NLC}_k$ ,  $t\langle\pi\rangle$  is equivalent to  $\circ_\pi(t)$ .*

*Proof.* By induction and Observation 9. □

Now we are ready to prove our normal form lemma for NLC-terms.

**Lemma 11.** *For every term  $t \in \text{NLC}_k$  with constant  $\lambda_t$  there is an equivalent term  $t' \in \text{NLC}_k$  in normal form.*

*Proof.* Let  $t$  be a term with constant  $\lambda_t$ , and let  $s$  any subterm of  $t$ . Let  $a(s) = |\mathcal{A}(V_s)|$ , and  $b(s) = |\{\lambda_s^{-1}(i) \mid i \in [k]\} \setminus \{\emptyset\}|$ . By Lemma 6 we have  $a(s) \leq b(s)$ . Let  $c(t)$  denote the sum of all  $b(p) - a(p)$  where  $p$  is a principal subterm of  $t$ . Among all terms  $r$  that are equivalent to  $t$  we chose one that minimises  $c(r)$ . Without loss of generality  $t$  realises the minimum. We claim that  $t$  is in normal form.

For the sake of a contradiction we assume a principal subterm  $p$  of  $t$  with  $a(p) < b(p)$ . Then  $V_p$  contains two equivalent vertices  $u$  and  $v$  with different labels, without loss of generality

with  $\lambda_p(u) = 1$  and  $\lambda_p(v) = 2$ . Let  $R : [k] \rightarrow [k] \quad i \mapsto \max\{2, i\}$ ,  $p' = \circ_R(p)$  and  $t'$  the term obtained from  $t$  by substituting  $p$  by  $p'$ . We clearly have  $c(t') < c(t)$ . We are done by showing that  $t'$  is equivalent to  $t$ .

It is easy to see that  $t' \in \text{NLC}_k$ ,  $V_{t'} = V_t$ ,  $\lambda_{t'} = \lambda_t$  and  $\text{red}(\text{tree}(t')) = \text{red}(\text{tree}(t))$ . It remains to prove  $E_{t'} = E_t$ .

First we consider an additional edge  $\{w, x\} \in E_{t'} \setminus E_t$ , without loss of generality  $w \in V_p$  and  $x \in V_t \setminus V_p$ . Then  $t'$  contains a subterm  $r' \times_S s$  such that  $u, v, w \in V_{r'}$  and  $x \in V_s$ , i.e.  $p'$  is a subterm of  $r'$ . We let  $r \times_S s$  denote the corresponding subterm in  $t$ . Then we have  $(\lambda_{r'}(w), \lambda_s(x)) \in S$  although  $(\lambda_r(w), \lambda_s(x)) \notin S$ . This implies  $\lambda_{r'}(w) \neq \lambda_r(w)$ . As we only added the relabelling  $R$  to obtain  $p'$  from  $p$  we infer that  $\lambda_p(w) = \lambda_p(u)$ , hence  $\lambda_p(w) = 1$ . Thus  $w$  and  $u$  are  $V_p$ -equivalent and by transitivity  $w$  is also equivalent to  $v$  in  $V_p$ . As  $\lambda_p(w) = 1$ , the vertex  $w$  obtains the same label as  $v$  in  $p'$ . Thus, we have also  $\lambda_{r'}(w) = \lambda_{r'}(v)$ . Now,  $(\lambda_{r'}(w), \lambda_s(x)) \in S$  implies that  $x$  is adjacent to  $v$ , contradicting the  $V_p$ -equivalence of  $w$  and  $v$ .

Finally let  $\{w, x\} \in E_t \setminus E_{t'}$  be a missing edge. Again, we can assume that  $w, u, v \in V_{r'}$  and  $x \in V_s$  for some subterm  $r' \times_S s$  of  $t'$ , where  $V_p \subseteq V_{r'}$ . We let again  $r \times_S s$  denote the corresponding subterm of  $t$ . This time we know that  $(\lambda_{r'}(w), \lambda_s(x)) \notin S$  although  $(\lambda_r(w), \lambda_s(x)) \in S$ . As above, we can conclude that  $\lambda_{r'}(w) \neq \lambda_r(w)$ . Further as above, this implies that  $\lambda_p(w) = 1$ , that  $w$  is equivalent to  $v$  in  $V_p$  and that  $\lambda_{r'}(w) = \lambda_{r'}(v)$ . Here, we can conclude that  $v$  is not adjacent to  $x$ , again contradicting the equivalence of  $v$  and  $w$ .  $\square$

## 4 Relative NLC-width

LOZIN and RAUTENBACH gave in [12] an  $\mathcal{O}(n^2m)$ -time algorithm approximating the relative cliquewidth by a factor of at most 2. In 7.2 we will see that the decision problem related to relative cliquewidth is NP-complete. In contrast this section shows how to compute the relative NLC-width in polynomial time.

### 4.1 Compatible atoms

The normal form for terms minimises the number of different labels at each principal subterm. Now we will show how to reuse as many labels as possible when joining two subgraphs in such a term in normal form.

**Definition 12.** Let  $G = (V, E)$  be any graph and let  $U$  and  $W$  be two disjoint subsets of  $V$ . Two atoms  $A, B \in \mathcal{A}(U) \cup \mathcal{A}(W)$  are *compatible* if there is an atom in  $\mathcal{A}(U \cup W)$  containing both  $A$  and  $B$ .

The compatibility of atoms is an equivalence relation.

**Lemma 13.** *Let  $s = p \times_S q$  be a subterm of a term  $t$ . If two vertices  $u \in V_p$  and  $w \in V_p \cup V_q$  in different atoms of  $\mathcal{A}(V_p) \cup \mathcal{A}(V_q)$  share the same label  $\lambda_s(u) = \lambda_s(w)$  then  $w \in V_q$  and the atoms  $A(V_p, u)$  and  $A(V_q, w)$  are compatible.*

*Proof.* Vertex  $w$  belongs to  $V_q$  as vertices in different atoms of  $V_p$  do not share a label by Lemma 6. For the sake of a contradiction we assume that  $A(V_p, u)$  and  $A(V_q, w)$  are incompatible. Then there is a vertex  $x \in V_t \setminus (V_p \cup V_q)$  in the symmetric difference  $N(p) \triangle N(q)$  contradicting  $\lambda_p(u) = \lambda_q(w)$ .  $\square$

For two disjoint subsets  $U$  and  $W$  of  $V$ , the *compatibility graph*  $Q$  of  $U$  and  $W$  is a bipartite graph with partite sets  $\mathcal{A}(U)$  and  $\mathcal{A}(W)$ , where two atoms  $A \in \mathcal{A}(U)$  and  $B \in \mathcal{A}(W)$  are adjacent if compatible.

**Lemma 14.** *Let  $G = (V, E)$  be a graph and  $U$  and  $W$  disjoint subsets of  $V$ . The compatibility graph of  $U$  and  $W$  can be computed in time  $\mathcal{O}(n^2)$ , where  $n = |V|$ .*

*Proof.* First we compute the adjacency matrix of  $G$ . Then we radix-sort the rows corresponding to vertices in  $U$ ,  $W$  and  $U \cup W$  using the bits in columns that correspond to vertices outside these sets. This way we find the atoms of  $U$ ,  $W$  and  $U \cup W$ .  $\square$

By transitivity of compatibility for atoms we have:

**Observation 15.** *Compatibility graphs are  $P_4$ -free.*

## 4.2 Characterisation

A *matching* of a graph  $Q = (X, F)$  is a set  $M \subseteq F$  of edges such that no vertex in  $X$  is adjacent to two edges in  $M$ . By  $\mu(Q)$  we denote the size of a *maximum matching* of  $Q$ , that is a matching with maximum cardinality. Similarly, let  $\alpha(Q)$  denote the size of a maximum independent set in  $Q$  and  $\nu(Q)$  the size of a minimum vertex cover. If  $Q$  has  $n$  vertices, then  $\alpha(Q) + \nu(Q) = n$ , and  $\mu(Q) \leq \nu(Q)$  with equality if  $Q$  is bipartite [13]. For a matching  $M$  of  $Q$  we let  $\alpha(Q, M)$  denote the size  $|M|$  of  $M$  plus the number of  $M$ -unsaturated vertices; if  $M$  is a maximum matching in a bipartite graph we have  $\alpha(Q, M) = \alpha(Q)$ .

The following lemma is an immediate consequence of Corollary 7 and Lemma 13.

**Lemma 16.** *Let  $s = p \times_S q$  be a subterm of a term  $t \in \text{NLC}_k$  in normal form and let  $Q$  be the compatibility graph of  $V_p$  and  $V_q$ . Then the set*

$$M_s = \{\{A, B\} \in E(Q) \mid \lambda_s(a) = \lambda_s(b) \text{ for all } a \in A, b \in B\}$$

*is a matching of  $Q$  with  $\alpha(Q, M_s) \leq k$ .*

We call  $Q_s$  the compatibility graph of  $V_p$  and  $V_q$  and  $M_s$  its matching *induced by  $s$* . Now we will show that we can adapt a term in normal form to given maximum matchings, whereby we minimize the number of labels at each join of the parse tree.

**Lemma 17.** *Let  $s = p \times_S q$  be a subterm of a term  $t \in \text{NLC}_k$  in normal form, and let  $M$  be a maximum matching of  $Q_s$ . Then there is a term  $t' \in \text{NLC}_k$  in normal form with corresponding subterms  $s' = p' \times_{S'} q'$  such that  $t'$  is equivalent to  $t$  and  $M_{s'} = M$ .*

*Proof.* We may assume a principal subterm  $r$  of  $t$  such that  $r = \circ_R(s)$  for a suitable function  $R: [k] \rightarrow [k]$ . We construct permutations  $\pi$  and  $\omega$ , a relation  $S'$  and a function  $R'$  such that  $r' = \circ_{R'}(p \langle \pi \rangle \times_{S'} q \langle \omega \rangle)$  is equivalent to  $r$ , and obtain  $t'$  from  $t$  by substituting  $r'$  for  $r$ . Then  $t'$  is equivalent to  $t$  by Observation 9.

Let  $\text{val}(r) = (V_p \cup V_q, F, \rho)$  and  $\text{val}(s) = (V_p \cup V_q, F, \sigma)$ . Let  $M = \{\{A_i, B_i\} \in E(Q_s) \mid i = 1, 2, \dots, m\}$  with  $A_i \in \mathcal{A}(V_p)$  and  $B_i \in \mathcal{A}(V_q)$  for  $1 \leq i \leq m$ . Let  $\{C_j \mid j = m+1, m+2, \dots, l\}$  be the set of  $M$ -unsaturated atoms in  $\mathcal{A}(V_p) \cup \mathcal{A}(V_q)$ .

We define  $\sigma': V_p \cup V_q \rightarrow [k]$  by

$$\sigma'(u) = \begin{cases} i & u \in A_i \\ j & u \in C_j \end{cases} \quad \text{and} \quad \sigma'(w) = \begin{cases} i & w \in B_i \\ j & w \in C_j \end{cases}$$

and

$$S' = \{(\sigma'(u), \sigma'(w)) \mid u \in V_p, w \in V_q, \{u, w\} \in E_t\}.$$

By Lemma 16 we know that the set  $M(s)$  is a matching of  $Q_s$ . Since  $M$  is a maximum matching we have  $m = |M| \geq |M_s|$  and  $l = \alpha(Q_s, M) \leq \alpha(Q_s, M_s) \leq k$ . By construction  $\sigma'$  restricted to  $V_p$  and  $V_q$  is canonical on these domains. Its range is  $[l] \subseteq [k]$ .

Since  $\rho$  is a canonical labelling of  $V_p \cup V_q$ , there is a function  $R' : [k] \rightarrow [k]$  such that  $R'(\sigma'(v)) = \rho(v)$  for all  $v \in V_p \cup V_q$ . Since  $\sigma$  restricted to  $V_p$  and  $V_q$  is canonical on these domains, there exist permutations  $\pi$  and  $\omega$  such that  $\pi(\sigma(u)) = \sigma'(u)$  for all  $u \in V_p$  and  $\omega(\sigma(w)) = \sigma'(w)$  for all  $w \in V_q$ . We set  $p' = p\langle\pi\rangle$  and  $q' = q\langle\omega\rangle$ . With  $s' = p' \times_{S'} q'$  we have  $M_{s'} = M$ . It remains to show that  $r' = \circ_{R'}(s')$  is equivalent to  $r$ .

We have  $V_p = V_{p'}$  and  $V_q = V_{q'}$ , hence  $V_r = V_{r'}$ . Similarly,  $E_p = E_{p'}$ ,  $E_q = E_{q'}$  and the definition of  $S'$  implies  $E_r \subseteq E_{r'}$ . We consider an edge  $\{u, w\} \notin E_r$  with  $u \in V_p$  and  $w \in V_q$ . Then  $\{u', w'\} \notin E_r$  for all  $u' \in \mathcal{A}(V_p, u)$  and  $w' \in \mathcal{A}(V_q, w)$ , and hence  $(\sigma'(u), \sigma'(w)) \notin S'$ . Therefore we have  $\{u', w'\} \notin E_{r'}$ . Finally,  $R'$  was constructed such that  $\lambda_r = \lambda_{r'}$ .  $\square$

Lemmas 13, 16 and 17 imply the following theorem.

**Theorem 18.** *Let  $G$  be a graph and  $R$  a reduced tree of  $G$ . Then the relative NLC-width of  $G$  is  $\text{r-nlc}(G, R) = \max\{\alpha(Q_c) \mid c \in V(R)\}$ .*

Because of Observation 15, all the connected components of an NLC-compatibility graph are complete bipartite graphs, and we have

**Observation 19.** *The value of  $\alpha(Q)$  can be computed in linear time if  $Q$  is a compatibility graph.*

### 4.3 Computing the relative NLC-width in polynomial time

In this subsection we describe `NLCterm`, an algorithm with the following arguments:

- a graph  $G = (V, E)$ ,
- a reduced tree  $T$  of  $G$ ,
- a node  $c$  of  $T$  and
- a labelling  $\lambda : V \rightarrow \mathbb{N}$ .

On this input `NLCterm` returns a term  $t \in \text{NLC}$  with  $\text{val}(t) = (G[V_c], \lambda)$ . At the beginning, we let  $c$  be the root of  $T$  and  $\lambda(v) = 1$  for all  $v \in V$ .

The algorithm `NLCterm` works recursively from  $c$  down to the leaves of  $T$ . At each stage it computes the relation  $S$  for the edge insertions and the function  $R$  for the relabelling. This can be done in a way similar to the description in the proof of Lemma 17. As in Subsection 3.1, each atom  $A$  of  $V_c$  is represented by its minimal vertex with respect to a fixed linear ordering on  $V$ . The vertex  $a = \min(A)$  keeps its label, and so do all vertices in its atom  $A(V_b, a)$  or  $A(V_d, a)$ , where  $b$  and  $d$  are the children of  $c$  in  $T$ . If  $\{A, B\}$  is an edge of the maximum matching of the compatibility graph  $Q_c$ , then all vertices in  $B$  obtain the same label as those in  $A$ . For all other atoms in  $\mathcal{A}(V_b) \cup \mathcal{A}(V_d)$  sets we assign new labels. Thereby we reuse as many labels as possible. By Lemma 13 we can use each label at most twice. More precisely, atoms  $A$  and  $B$  use the same label if and only if  $\{A, B\}$  is in a fixed maximum

```

function NLCTerm( $G, T, c, \lambda$ )
begin
  if  $c$  is a leaf of  $T$  labelled  $v$  then return  $\bullet_{\lambda(v)}(v)$ 
  else /*  $c$  is an internal node of  $T$  */
    let  $b$  and  $d$  be the children of  $c$  in  $T$ ;
    define the compatibility graph  $Q_c = (\mathcal{A}(V_b), \mathcal{A}(V_d), F)$ ;
    compute a maximum matching  $M \subseteq F$  of  $Q_c$ ;
     $L \leftarrow \{1, 2, \dots, |V(G)|\}$ ; /* set of available labels */
     $S \leftarrow \emptyset$ ; for  $l \in L$  do  $R[l] \leftarrow l$  od; /* initialising  $S$  and  $R$  */
    for  $v \in V_c$  do  $\chi(v) \leftarrow 0$  od; /* 0 means undefined */
    for  $A \in \mathcal{A}(V_c)$  do  $v \leftarrow \min(A)$ ;  $L \leftarrow L \setminus \{\lambda(v)\}$ ;
      if  $v \in V_b$  then
        for  $z \in A(V_b, v)$  do  $\chi(z) \leftarrow \lambda(v)$  od;
        if  $\{A(V_b, v), B\} \in M$  then for  $z \in B$  do  $\chi(z) \leftarrow \lambda(v)$  od fi
      else /*  $v \in V_d$  */
        for  $z \in A(V_d, v)$  do  $\chi(z) \leftarrow \lambda(v)$  od;
        if  $\{A(V_d, v), B\} \in M$  then for  $z \in B$  do  $\chi(z) \leftarrow \lambda(v)$  od fi
      fi
    od;
    for  $A \in \mathcal{A}(V_b)$  do  $v \leftarrow \min(A)$ ;
      if  $\chi(v) = 0$  then
         $l \leftarrow \min(L)$ ;  $L \leftarrow L \setminus \{l\}$ ;  $R[l] \leftarrow \lambda(v)$ ;
        for  $z \in A$  do  $\chi(z) \leftarrow l$  od;
        if  $\{A, B\} \in M$  then for  $z \in B$  do  $\chi(z) \leftarrow l$  od fi
      fi
    od;
    for  $A \in \mathcal{A}(V_d)$  do  $v \leftarrow \min(A)$ ;
      if  $\chi(v) = 0$  then
         $l \leftarrow \min(L)$ ;  $L \leftarrow L \setminus \{l\}$ ;  $R[l] \leftarrow \lambda(v)$ ;
        for  $z \in A$  do  $\chi(z) \leftarrow l$  od
      fi;
      for  $s \in N(v) \cap V_b$  do
        if  $s = \min(A(V_b, s))$  then  $S \leftarrow S \cup (\chi(s), \chi(v))$  fi
      od
    od;
     $p \leftarrow \text{NLCTerm}(G, T, b, \chi)$ ;
     $q \leftarrow \text{NLCTerm}(G, T, d, \chi)$ ;
    return  $R(p \times_S q)$ 
  fi
end.

```

Table 3: Algorithm NLCTerm

matching of the compatibility graph. With the new labelling, the nodes  $b$  and  $d$  of  $T$  are handled recursively.

The recursion stops for leaves of the reduced tree, which contain a singleton  $\{v\}$ . The required NLC-term is  $\bullet_i(v)$ , where  $i$  is the label required at  $v$ . In Table 3 we give the pseudo-code of algorithm `NLCterm`.

#### 4.4 Run-time analysis

**Theorem 20.** *The relative NLC-width of a graph can be computed in time  $\mathcal{O}(n^3)$ .*

*Proof.* The correctness of algorithm `NLCterm` follows from Theorem 18.

To analyse the running time of our algorithm, we firstly observe that  $T$  has  $n$  leaves and  $n - 1$  inner nodes. Leaves can be handled in constant time. For each inner node  $c$  we construct the compatibility graph  $Q_c$ . This can be done in  $\mathcal{O}(n^2)$ , see Lemma 14. By Observation 19,  $\alpha(Q_c)$  can be computed in linear time, and the stated bound of the overall running time follows.  $\square$

#### 4.5 Restriction to sequential terms in NLC

In this subsection we deal with the special case of restricted trees that have a spine, *i.e.* a path  $S$  such that every node not on  $S$  has a neighbour on  $S$ . The corresponding terms in NLC are sequential.

Let  $s = \bullet_i(u) \times_S r$  be a subexpression of a term  $t$ . Clearly  $\mathcal{A}(\{u\}) = \{\{u\}\}$ . In order to compute the atoms in  $\mathcal{A}(V_r)$ , we observe that each atom in  $\mathcal{A}(V_r \cup \{u\})$  splits in at most two: the neighbours and the non-neighbours of  $u$ . Therefore, for each  $v \in V_r$  we have to check whether it is adjacent to  $u$ , which can be looked up in an adjacency matrix of  $G$  in constant time. We check whether the compatibility graph has an edge by verifying whether  $A(V_r \cup \{u\}, u) \neq \{u\}$  or not. Hence the algorithm `NLCterm` runs in time  $\mathcal{O}(n^2)$  if the reduced tree  $T$  has a spine.

In [12] it was shown that in this case the relative cliquewidth can also be computed in polynomial time. In Section 7, we will present the NP-completeness result for relative cliquewidth in general.

## 5 Computing NLC-width

### 5.1 Idea

We can use algorithm `NLCterm` to compute the NLC-width  $G = (V, E)$  by checking all reduced trees. Algorithm `NLCwidth`, given in Table 4, does exactly this. If two reduced trees contain a common subtree, it computes the value for this subtree only once and stores it in an array  $D$ . For each set  $X \subseteq V$ ,  $D[X]$  is the value of  $\text{nlc}(G[X], \lambda_X)$ , where  $\lambda_X$  is a canonical labelling. For simplicity we restrict to these values.

To obtain the value  $\text{nlc}(G[X], \lambda_X)$ , the algorithm checks all partitions of  $X$  into two subsets  $U$  and  $W$  and computes the value  $\alpha(Q)$  for the compatibility graph  $Q = (\mathcal{A}(U), \mathcal{A}(W), F)$ . This value can be computed without actually constructing  $Q$ , see Table 5 and recall Subsection 3.1 for the encoding. The algorithm `NLCmatching` uses an array of stacks. For each atom  $A \in \mathcal{A}(U \cup W)$  it counts the number of atoms from  $\mathcal{A}(U)$  and  $\mathcal{A}(W)$  respectively that are included in  $A$  and matches these reducing the counter.

The algorithm `NLCterm` extends to a version storing optimal reduced trees or even optimal NLC-terms. But this requires the actual construction of the compatibility graph and a matching, which would cost an additional factor  $n$  for the runtime.

```

function NLCwidth( $V, E$ )
begin
  caa( $V, E$ );
  for  $X \leftarrow \emptyset$  to  $V$  do
     $k \leftarrow |X|$ ;
    for  $U \in 2^X \setminus \{\emptyset, X\}$  do
       $W \leftarrow X \setminus U$ ;
      compute the value  $\alpha(Q)$  for
      the compatibility graph  $Q = (\mathcal{A}(U), \mathcal{A}(W), F)$ ;
       $k \leftarrow \min(k, \max(\alpha(Q), D[U], D[W]))$ ;
    od;
     $D[X] \leftarrow k$ 
  od;
  return  $D[V]$ 
end.

```

Table 4: Algorithm `NLCwidth`

## 5.2 Run-time analysis

Let  $G$  be a graph on  $n$  vertices. Then `caa` runs in time  $\mathcal{O}(2^n n)$ , see Lemma 4. With this information on atoms, we can compute  $\alpha(Q)$  without constructing  $Q$  explicitly in time  $\mathcal{O}(n)$ , see `NLCmatching` in Table 5. We access the array  $D$  in constant time. For every set  $X \subseteq V$  we check  $2^{|X|} - 2$  partitions  $(U, W)$ . The overall running time of `NLCwidth` is  $\mathcal{O}(3^n n)$  because  $\sum_{X \subseteq V} 2^{|X|} = \mathcal{O}(3^n)$ .

**Theorem 2.** *The NLC-width of a graph on  $n$  vertices can be computed in time  $\mathcal{O}(3^n n)$ .*

## 5.3 Sequential NLC-width

To compute the sequential NLC-width of a graph we use an algorithm similar to `NLCwidth`, where  $U$  is a one-element subset of  $X$ . That is, the inner for-loop is executed less than  $n$  times.

**Theorem 21.** *The sequential NLC-width of a graph can be computed in time  $\mathcal{O}(2^n n^2)$ .*

## 6 Characterisation of NLC-width

Many width-parameters, like treewidth, branchwidth or rankwidth, are based on a decomposition tree. We can now provide a similar characterisation of NLC-width. The compatibility graph of two disjoint subsets  $U$  and  $W$  of the vertex set  $V$  of a graph is defined in terms of neighbourhoods and does not require the operations that are used for NLC-terms. The value  $\alpha(Q)$  as defined in Subsection 4.2 measures the bond between these two subsets.

```

function NLCmatching( $U, W$ )
begin
   $\alpha \leftarrow 0$ ;
  for  $v \in U \cup W$  do
    if  $\text{atom}[U \cup W, v] = v$  then  $S[v] \leftarrow \text{empty\_stack}$  fi
  od;
  for  $u \in U$  do
    if  $\text{atom}[U, u] = u$  then
       $\text{push}(S[\text{atom}(U \cup W, u)], u)$ ;
       $\alpha \leftarrow \alpha + 1$ ;
    fi
  od;
  for  $w \in W$  do
     $s \leftarrow S[\text{atom}(U \cup W, w)]$ ;
    if  $\text{atom}[W, w] = w$  then
       $\alpha \leftarrow \alpha + 1$ ;
      if not  $\text{isEmpty}(s)$  then
         $\text{pop}(s)$ ;
         $\alpha \leftarrow \alpha - 1$ ;
      fi
    fi
  od
  return  $\alpha$ 
end.

```

Table 5: matching algorithm NLCmatching

Let  $c$  be a node of a reduced tree  $R$  of  $G$ . Then we assign to  $c$  the value  $\alpha(Q_c)$ . The *width*  $w(R)$  of the decomposition is the maximum of these values over all inner nodes. The NLC-width of the graph is then equal to the minimum of  $w(R)$  over all reduced trees  $R$  of  $G$ .

## 7 Relative Cliquewidth

### 7.1 Definition

Cliquewidth is a graph-parameter similar to NLC-width. The only difference is the set of operations, in case of cliquewidth captured by expressions. Each expression  $t$  describes a labelled graph  $\text{val}(t) = (V_t, E_t, \lambda_t)$ . A  $k$ -expression uses labels from  $[k]$  only. Expressions are recursively defined as follows:

**singleton:** For all labels  $i$  and all vertices  $v$ , there is an expression  $\bullet_i(v)$  with  $\text{val}(\bullet_i(v)) = (\{v\}, \emptyset, \lambda)$ , where  $\lambda(v) = i$ .

**disjoint union:** For all expressions  $p$  and  $q$  with  $V_p \cap V_q = \emptyset$ ,  $p \oplus q$  is an expression with  $\text{val}(p \oplus q) = (V_p \cup V_q, E_p \cup E_q, \lambda)$ , where  $\lambda(u) = \lambda_p(u)$  for  $u \in V_p$  and  $\lambda(w) = \lambda_q(w)$  for  $w \in V_q$ .

**insert edges:** For all pairs  $(i, j)$  of distinct labels and all expressions  $t$ ,  $\eta_{i,j}(t)$  is an expression with  $\text{val}(\eta_{i,j}(t)) = (V_t, E_t \cup F, \lambda_t)$ , where  $F = \{\{v, w\} \mid v, w \in V_t, \lambda_t(v) = i, \lambda_t(w) = j\}$ .

**relabel:** For all pairs  $(i, j)$  of labels and all expressions  $t$ ,  $\rho_{i \rightarrow j}(t)$  is an expression with  $\text{val}(\rho_{i \rightarrow j}(t)) = (V_t, E_t, \lambda)$  where  $\lambda(v) = j$  if  $\lambda_t(v) = i$  and  $\lambda(v) = \lambda_t(v)$  otherwise.

**otherwise:** There is no other expression.

The *cliquewidth* of a labeled graph  $G$  is the minimum integer  $k \geq 0$  such that there exists a  $k$ -expression  $t$  with  $\text{val}(t) = G$ . Again, we can consider non-labeled graphs as graphs where all vertices are labelled by the same label. Or equivalently we can define the cliquewidth of an non-labelled graph  $(V, E)$  to be the minimum cliquewidth of a labelled graph  $(V, E, \lambda)$ , where the minimum is taken over all vertex-labellings  $\lambda$ .

Let  $t$  be an expression. We define the corresponding expression tree  $T = \text{tree}(t)$ , the reduced tree  $\text{red}(T)$  and a reduced tree of a graph  $G$  analogously to the definitions for NLC-width. The relative cliquewidth of a graph  $G$  with respect to a reduced tree  $R$  of  $G$  is then defined to be the minimum  $k$  such that there is a  $k$ -expression  $t$  that defines  $G$  with  $\text{red}(\text{tree}(t)) = R$ .

The operation  $\oplus$  is commutative and associative in the following sense: For all expressions  $t_1, t_2$  and  $t_3$  we have  $\text{val}(t_1 \oplus t_2) = \text{val}(t_2 \oplus t_1)$  and  $\text{val}(t_1 \oplus (t_2 \oplus t_3)) = \text{val}((t_1 \oplus t_2) \oplus t_3)$ . This enables us to save parentheses in  $t_1 \oplus t_2 \oplus \dots \oplus t_n$ , or  $\bigoplus_{i=1}^n t_i$  for short. The concatenation of edge-insertions is commutative and associative in the same sense. Therefore we use  $(\bigcirc_{i=1}^n \eta_{j(i), l(i)})(t)$  to abbreviate  $\eta_{j(1), l(1)}(\eta_{j(2), l(2)}(\dots(\eta_{j(n), l(n)}(t))\dots))$ . For relabellings we use  $(\bigcirc_{i=1}^n \rho_{j(i) \rightarrow l(i)})(t)$  only in case of disambiguity.

The following lemma can be seen as easily as Lemma 6.

**Lemma 22.** *Let  $s$  be a subexpression of  $t$ . Then the partition of  $V_s$  into classes of equal labels with respect to  $\lambda_s$  refines the partition of  $V_s$  into atoms.*

## 7.2 NP-completeness

We define the problem RELATIVE CLIQUEWIDTH as follows:

*Instance:* A graph  $G = (V, E)$ , a reduced tree  $R$  of  $G$  and an integer  $k$ .

*Question:* Is  $\text{r-cwd}(G, R) \leq k$ ?

Obviously this problem belongs to NP. To show the hardness we use a reduction from INDEPENDENT SET:

*Instance:* A graph  $G = (V, E)$  and an integer  $k$ .

*Question:* Does  $G$  contain an independent set of size at least  $k$ ?

Let  $G = (V, E)$  be a graph with  $V = \{v_1, \dots, v_n\}$ , let  $k$  be an integer, and let  $(G, k)$  be an instance of INDEPENDENT SET. We assume that  $G$  contains at least one edge. This restriction does not affect the NP-hardness of INDEPENDENT SET. We define a graph  $H = (A \cup B \cup C, F_1 \cup F_2 \cup F_3)$  by

$$\begin{aligned} A &= \{a_i \mid 1 \leq i \leq n\} & F_1 &= \{\{a_i, b_i\}, \{b_i, c_i\} \mid 1 \leq i \leq n\} \\ B &= \{b_i \mid 1 \leq i \leq n\} & F_2 &= \{\{a_i, c_j\} \mid \{v_i, v_j\} \in E \text{ and } i < j\} \\ C &= \{c_i \mid 1 \leq i \leq n\} & F_3 &= \{\{a_i, a_j\}, \{c_i, c_j\} \mid 1 \leq i < j \leq n\} \end{aligned}$$

In Figure 2 we give an example of this construction.

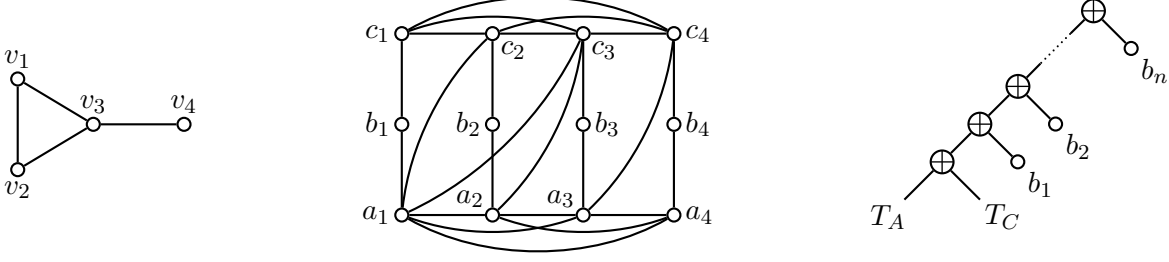


Figure 2: A graph  $G$ , the corresponding  $H$  and the reduced tree  $R$ .

Let  $R$  be the reduced tree of  $H$  given in the same figure, where  $T_A = \text{red}(\text{tree}(t_A))$  and  $T_C = \text{red}(\text{tree}(t_C))$  for the expressions  $t_A$  and  $t_C$  given below. To prove the NP-hardness of RELATIVE CLIQUEWIDTH we show  $\alpha(G) \geq k$  if and only if  $\text{r-cwd}(H, R) \leq 2n - k$ .

Let  $S \subseteq V$  be an independent set of  $G$  of size  $k$ . We construct a  $(2n - k)$ -expression for  $H$ . For all  $i \in [n]$  let  $l(i) = i$  if  $v_i \in S$  and let  $l(i) = \min([2n] \setminus ([n] \cup \{l(j) \mid j \in [i - 1]\}))$  if  $v_i \notin S$ .

The expressions  $t_A$  and  $t_C$  create the cliques  $A$  and  $C$  labelled with labels from  $[n]$  and  $\{l(i) \mid i \in [n]\}$ , respectively. Note that  $[n] \cup \{l(i) \mid i \in [n]\} = [2n - k]$  and  $\text{val}(t_A \oplus t_C) = (A \cup C, F_3, \chi_{A \oplus C})$  where  $\chi_{A \oplus C}(a_i) = i$  and  $\chi_{A \oplus C}(c_i) = l(i)$  for all  $i$ ,  $1 \leq i \leq n$ .

In expression  $t_0$  we first create exactly the edges in  $F_2$  and then relabel all vertices  $c_i$  with label  $l(i) > n$  to label  $i$ . The vertices  $b_j$  are added successively in the subterms  $t_j$ ,  $1 \leq j \leq n$ . Each  $b_j$  is created as singleton with label  $n + 1$  and united with  $t_{j-1}$ . Next we add the edges  $\{a_j, b_j\}$  and  $\{b_j, c_j\}$  and finally we relabel  $b_j$  to  $j$ . Since  $G$  contains at least one edge we have that  $n + 1 \leq 2n - k$ . For all  $j$  with  $1 \leq j \leq n$  we have  $\text{val}(t_j) = (A \cup B_j \cup C, F_{1,j} \cup F_2 \cup F_3, \chi_j)$  where  $B_j = \{b_i \mid 1 \leq i \leq j\}$ ,  $F_{1,j} = \{\{a_i, b_i\}, \{b_i, c_i\} \mid 1 \leq i \leq j\}$ ,  $\chi_j(a_i) = i$ ,  $\chi_j(c_i) = i$  for all  $i$  with  $1 \leq i \leq n$  and  $\chi_j(b_i) = i$  for all  $i$  with  $0 \leq i \leq j$ .

In  $t$  we relabel all vertices to label 1 and have  $\text{val}(t) = (H, \chi)$  where  $\chi(x) = 1$  for all  $x \in A \cup B \cup C$ . So we conclude  $\text{r-cwd}(H, R) \leq 2n - k$ . Formally:

$$\begin{aligned}
t_A &= \left( \bigcirc_{i=1}^{n-1} \bigcirc_{j=i+1}^n \eta_{i,j} \right) \left( \bigoplus_{i=1}^n \bullet_i(a_i) \right) \\
t_C &= \left( \bigcirc_{i=1}^{n-1} \bigcirc_{j=i+1}^n \eta_{(i),l(j)} \right) \left( \bigoplus_{i=1}^n \bullet_{l(i)}(c_i) \right) \\
t_0 &= \left( \bigcirc_{v_i \notin S} \rho_{l(i) \rightarrow i} \right) \left( \left( \bigcirc_{\{v_i, v_j\} \in E, i < j} \eta_{i,l(j)} \right) (t_A \oplus t_C) \right) \\
t_j &= \rho_{n+1 \rightarrow j} \left( \eta_{n+1,j} (t_{j-1} \oplus \bullet_{n+1}(b_j)) \right) \quad \text{for } 1 \leq j \leq n \\
t &= \left( \bigcirc_{i=2}^n \rho_{i \rightarrow 1} \right) (t_n)
\end{aligned}$$

Now we assume an expression defining  $H$  with reduced tree  $R$  that uses at most  $2n - k$  labels. We first observe that each vertex  $a_i$  forms an atom of  $A$ , and each vertex  $c_i$  forms an atom of  $C$ . By Lemma 22 we cannot save labels in  $t_A$  and  $t_C$ . When the trees  $T_A$  and  $T_C$  are united,  $a_i$  can share its label only with  $c_i$  because  $b_i$  exists.

We consider indices  $i < j$  such that the vertices  $a_i$  and  $c_i$  share a label, and  $a_j$  and  $c_j$  share another label. Then  $v_i$  and  $v_j$  are non-adjacent in  $G$ . Otherwise, there would be an

edge  $\{a_i, c_j\} \in F_2$ , but we would not be able to create it without creating an edge  $\{a_j, c_i\} \notin F_2$  at the same time. Thus the set of pairs  $(a_i, c_i)$  that share common labels corresponds to an independent set in  $G$ . Consequently,  $G$  contains an independent set of size at least  $k$ .

That is  $\text{r-cwd}(H, R) = 2n - \alpha(G)$ , and we have shown the following:

**Theorem 23.** *RELATIVE CLIQUEWIDTH is NP-complete.*

### 7.3 Sequential cliquewidth

As shown in [12] the relative cliquewidth can be computed in polynomial time in the sequential case. Each disjoint union unites a singleton set  $U = \{u\}$  and a general subset  $W$ . The authors show that we need either  $a(W)$  or  $a(W) + 1$  labels to unite these subsets. We can always unite these two subsets using  $a(W) + 1$  labels by assigning one label to each atom of  $W$  and an extra label to  $u$ . Thus, in order to check whether we can unite these subsets with  $a(W)$  labels, we have to decide whether we can reuse one of the labels of  $\mathcal{A}(W)$ .

By Lemma 14 we can compute the atoms of  $W$  in time  $\mathcal{O}(n^2)$ . Then, for each atom  $A$  of  $W$  we have to verify whether  $u$  and  $A$  lie in the same atom of  $W \cup \{u\}$  and whether  $N(u) \cap W$  is contained in  $N(v) \cap W$  for each  $v \in A$  in  $W$ . In this case and only in this case  $u$  and  $A$  can share the same label. We can obviously check this condition in time  $\mathcal{O}(n^2)$ . Thus, we can use an algorithm similar to `NLCwidth` to compute the sequential cliquewidth of a graph.

**Theorem 24.** *The sequential cliquewidth of a graph can be computed in time  $\mathcal{O}(2^n n^3)$ .*

## 8 Conclusions

In this paper we point out a discrepancy in the computational complexity of the closely related parameters of relative cliquewidth and relative NLC-width. Our reduction (showing that RELATIVE CLIQUEWIDTH is NP-hard) is much easier than the arguments used in [7] and [9], but does not imply the NP-hardness of CLIQUEWIDTH or NLC-WIDTH.

Cliquewidth and NLC-width are equally useful from the computational point of view, because  $\text{nlc}(G) \leq \text{cwd}(G) \leq 2 \cdot \text{nlc}(G)$  holds for all graphs  $G$ . But for some reason, cliquewidth has always been more popular than NLC-width. The characterisation we provide in Section 6 supports NLC-width: The bonding function  $f$  we use for NLC-width is computable in polynomial time.

Table 6 summarises the complexity results of this paper and related ones.

### Acknowledgement

We would like to thank anonymous referees for drawing our attention to relative NLC-width and hints that led to tighter bounds on running times.

## References

- [1] Corneil, D.G., M. Habib, J.-M. Lanlignel, B. Reed and U. Rotics: Polynomial time recognition of clique-width  $\leq 3$  graphs.  
In G.H. Gonnet *et al.* (eds): Proceedings LATIN 2000. Springer-Verlag Berlin, *Lecture Notes Computer Science* **1776** (2000) 126–134.

		NLC-width	cliquewidth
relative	sequential	$\mathcal{O}(n^2)$ Subsection 4.5	$\mathcal{O}(n^3)$ Subsection 7.3
	general	$\mathcal{O}(n^3)$ Theorem 20	NP-complete Theorem 23
general	sequential	$\mathcal{O}(2^n n^2)$ Theorem 21	$\mathcal{O}(2^n n^3)$ Theorem 24
	general	$\mathcal{O}(3^n n)$ Theorem 2	NP-complete [7]

Table 6: Summary of timebounds

- [2] Courcelle, B. and A. Twigg: Compact Forbidden-Set Routing.  
In W. Thomas, *et al.* (eds.), Proceedings STACS 2007. Springer-Verlag Berlin, *Lecture Notes Computer Science* **4393** (2007) 37–48.
- [3] Courcelle, B., J. Engelfriet and G. Rozenberg: Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences* **46** (1993) 218–270.
- [4] Courcelle, B., J.A. Makowsky and U. Rotics: Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems* **33** (2000) 125–150.
- [5] Golumbic, M.C. and U. Rotics: On the clique-width of some perfect graph classes. *International Journal of Foundations of Computer Science* **11** (2000) 423–443.
- [6] Espelage, W., F. Gurski and E. Wanke: Deciding clique-width for graphs of bounded tree-width. *Journal of Graph Algorithms and Applications* **7** (2003) 141–180.
- [7] Fellows, M.R., F.A. Rosamond, U. Rotics and St. Szeider: Clique-width minimization is NP-hard.  
In Kleinberg, J.M. (ed.): Proceedings 38th STOC (2006) 354–362.
- [8] Gurski, F. and E. Wanke: On the relationship between NLC-width and linear NLC-width. *Theoretical Computer Science* **347** (2005) 76–89.
- [9] Gurski, F. and E. Wanke: Minimizing NLC-width is NP-complete.  
In D. Kratsch (ed.): Proceedings WG 2005. Springer-Verlag Berlin, *Lecture Notes Computer Science*
- [10] Johansson, Ö.: Clique-decomposition, NLC-decomposition, and modular decomposition – relationships and results for random graphs. *Congressus Numerantium* **132** (1998) 39–60.
- [11] Johansson, Ö.: NLC<sub>2</sub>-Decomposition in polynomial time. *International Journal on Foundations of Computer Science* **11** (2000) 373–395.

- [12] Lozin, V. and D. Rautenbach: The relative clique-width of a graph. *Journal of Combinatorial Theory, Series B* **97** (2007) 846–858.
- [13] Kőnig, D.: Gráfok és mátrixok. *Matematikai és Fizikai Lapok* **38** (1931) 116–119.
- [14] Oum, D. and P. Seymour: Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B* **96** (2006) 514–528.
- [15] Wanke, E.:  $k$ -NLC graphs and polynomial algorithms. *Discrete Applied Mathematics* **54** (1994) 251–266.