

# On the OBDD Size for Graphs of Bounded Tree- and Clique-Width

Klaus Meer<sup>1</sup> and Dieter Rautenbach<sup>2</sup>

<sup>1</sup> Department of Mathematics and Computer Science  
Syddansk Universitet, Campusvej 55, 5230 Odense M, Denmark  
e-mail: meer@imada.sdu.dk; FAX: 0045 6593 2691

<sup>2</sup> Forschungsinstitut für Diskrete Mathematik  
Universität Bonn, Lennéstrasse 2, 53113 Bonn, Germany  
e-mail: rauten@or.uni-bonn.de; FAX: 0049 228 738771

**Abstract.** We study the size of OBDDs (ordered binary decision diagrams) for representing the adjacency function  $f_G$  of a graph  $G$  on  $n$  vertices. Our results are as follows:

- for graphs of bounded tree-width there is an OBDD of size  $O(\log n)$  for  $f_G$  that uses encodings of size  $O(\log n)$  for the vertices;
- for graphs of bounded clique-width there is an OBDD of size  $O(n)$  for  $f_G$  that uses encodings of size  $O(n)$  for the vertices;
- for graphs of bounded clique-width such that there is a *reduced term* for  $G$  (to be defined below) that is balanced with depth  $O(\log n)$  there is an OBDD of size  $O(n)$  for  $f_G$  that uses encodings of size  $O(\log n)$  for the vertices;
- for cographs, i.e. graphs of clique-width at most 2, there is an OBDD of size  $O(n)$  for  $f_G$  that uses encodings of size  $O(\log n)$  for the vertices. This last result improves a recent result by Nunkesser and Woelfel [14].

## 1 Introduction

The most usual way to represent graphs certainly is by adjacency lists or adjacency matrices. However, for applications like traffic scheduling, Travelling Salesman, and many more such representations can easily become very large. A way of trying to circumvent the storing problems related to such large graphs is to consider data structures for Boolean functions, hoping for a more succinct representation of the graph's adjacency function by such structures. One of the most important data structures for Boolean functions that has been used quite successfully are *Ordered Binary Decision Diagrams*, OBDDs for short. For a comprehensive introduction into the theory of OBDDs we refer to [17, 18].

By cardinality arguments we cannot represent any arbitrary graph by OBDDs of small size. Thus it is a reasonable question whether succinct OBDD representations can be found at least for significant graph classes. We shall look for such classes among graphs whose local structure is somehow controlled. For some graph classes including cographs and unit interval graphs this kind of problem was recently studied by Nunkesser and Woelfel [14], where several upper and lower bounds for the sizes of OBDDs representing such graphs were given. Other work relating algorithmic graph problems with OBDD representations are [10, 11, 15, 16, 19].

The present paper takes the work of [14] as starting point and analyzes further important graph classes with respect to their representability by small sized OBDDs. Two of the most important parameters for graphs are the tree-width and the clique-width [8,9]. For graphs of bounded tree- or bounded clique-width many otherwise intractable algorithmic problems are known to be efficiently solvable [1, 6, 7]. Note that bounded tree-width implies bounded clique-width [8].

We study the representation of graphs belonging to one of the above classes by OBDDs. Section 2 recalls the main concepts used in the paper. In Section 3 it is shown that for each graph with  $n$  vertices and of bounded tree-width there exists an OBDD of size  $O(\log n)$  that uses vertex encodings of length  $O(\log n)$ . Section 4 is devoted to graphs of bounded clique-width. First, we show that when using vertex encodings of length  $O(n)$ , then each graph with  $n$  vertices having a bounded clique-width allows an OBDD representation of size  $O(n)$ . Thereafter, we improve the above result for cographs, i.e. graphs of clique-width at most 2. Here, representing OBDDs of size  $O(n)$  exist which use vertex encodings of length  $O(\log n)$ . This last result improves a related result in [14] by a factor of  $\log n$ .

## 2 Basic Concepts

In order to make the paper self-contained we briefly recall the main notions we are dealing with in this paper, i.e. ordered binary decision diagrams as well as the tree- and the clique-width of a graph. For more elaborate treatments confer [18] and [5,9]. We then collect some results from [12] which are important later on.

### 2.1 OBDDs; Tree- and Clique-Width of a Graph

Binary decision diagrams (shortly BDDs) and their variants are by now among the most frequently used data structures for Boolean functions [18]. We are in particular interested in OBDDs (ordered BDDs).

**Definition 1** *a) A binary decision diagram or BDD is a directed acyclic graph having two kinds of nodes. Output nodes are nodes with no outgoing edge and are labeled with a Boolean constant from  $\{0,1\}$ . Inner nodes are labeled with an element from some variable set  $\{x_1, \dots, x_n\}$ . They have two outgoing edges one of which is labeled by 0 and the other by 1.*

*b) Each node  $v$  of a BDD computes a Boolean function  $f_v : \{0,1\}^n \rightarrow \{0,1\}$  in the following way: Given an assignment for the boolean variables  $x_1, \dots, x_n$  one follows, starting at  $v$ , the edges according to the value of the corresponding label until an output node is reached whose label gives  $f_v(x_1, \dots, x_n)$ . If the underlying graph has a root  $r$ , then  $f_r$  is also called the function computed by the BDD.*

*c) The size of a BDD is the number of nodes of the underlying graph.*

*d) A BDD is an ordered binary decision diagram with respect to the variable ordering  $x_1 < \dots < x_n$ , or OBDD for short, if for every edge of the BDD from an inner node with label  $x_i$  to an inner node with label  $x_j$  the indices satisfy  $j > i$ .*

In an OBDD each variable is read at most once and always in the same order.

Let  $G = (V, E)$  be a graph. We are interested in OBDDs computing the adjacency function  $f_G : V^2 \rightarrow \{0,1\}$  with  $f_G(i, j) = 1$  iff  $(i, j) \in E$ . Here, we first have to specify how a node is encoded in order to use it as (part of) the argument of a Boolean function.

**Definition 2** Let  $G = (V, E)$  be a graph with  $n := |V|$  nodes and let  $\text{name} : V \rightarrow \{0, 1\}^L$  be an encoding of the nodes by binary strings of length  $L$ . An OBDD  $O$  is said to represent  $G$  if  $O$  computes a Boolean function  $f_O$  from  $\{0, 1\}^{2L} \rightarrow \{0, 1\}$  such that for all  $i, j \in V$  it is

$$f_O(\text{name}(i), \text{name}(j)) = f_G(i, j).$$

A few words concerning the used encoding are appropriate. The following upper bound is well known.

**Theorem 3.** ([3]) Let  $f : \{0, 1\}^N \rightarrow \{0, 1\}$ . Then there is an OBDD of size  $(2 + o(1)) \cdot \frac{2^N}{N}$  that computes  $f$ .

For the adjacency function of a graph with  $n$  nodes this implies that we can always find a representing OBDD of size at most  $O(\frac{n^2}{\log n})$  by choosing the binary representations of the elements in  $\{1, \dots, n\}$  as names for the vertices. The adjacency function then takes  $2 \cdot \lceil \log n \rceil$  many arguments.

Thus, if below we shall use encodings of a different length we have to keep in mind this upper bound when comparing the sizes of the OBDDs constructed.

**Definition 4 (Tree-width of a graph)** A tree decomposition of width  $k \in \mathbb{N}$  of a (simple and undirected) graph  $G = (V, E)$  is a pair  $(T, (X_t)_{t \in V_T})$  where  $T = (V_T, E_T)$  is a rooted tree and  $(X_t)_{t \in V_T}$  is a collection of subsets  $X_t \subseteq V$  of cardinality at most  $k + 1$  such that for every edge  $uv \in E$  there is a  $t \in V_T$  with  $u, v \in X_t$  and for every vertex  $u \in V$  the set  $\{t \in V_T \mid u \in X_t\}$  induces a subtree of  $T$  having at least one vertex.

The tree-width of  $G$  is the minimum  $k$  such that there is a tree decomposition of width  $k$  of  $G$ .

The final concept we recall is that of the clique-width of a graph. For  $k \in \mathbb{N}$  a  $k$ -graph is a graph  $G = (V, E)$  whose vertices are labeled with a label from  $\{1, \dots, k\}$ . We consider three operations on  $k$ -graphs: First, for two  $k$ -graphs  $G = (V, E)$  and  $H = (W, F)$  with disjoint sets of vertices the  $k$ -graph  $G \oplus H$  is obtained by taking  $V \cup W$  as new vertex set and  $E \cup F$  as new edge set. The original labels are maintained. Secondly, for  $i, j \in \{1, \dots, k\}, i \neq j$  and a  $k$ -graph  $G$  the  $k$ -graph  $\eta_{i,j}(G)$  is obtained from  $G$  by connecting all vertices labeled  $i$  with those labeled  $j$  in  $G$ . Thirdly, for  $i, j \in \{1, \dots, k\}, i \neq j$  and a  $k$ -graph  $G$  the  $k$ -graph  $\rho_{i \rightarrow j}(G)$  is obtained from  $G$  by relabeling all nodes having label  $i$  with label  $j$ . Finally, for  $i \in \{1, \dots, k\}$  the operation  $i(v)$  creates a  $k$ -graph with one vertex  $v$  that carries label  $i$ .

**Definition 5** a) Let  $k \in \mathbb{N}$  be fixed. A  $k$ -expression is a well formed term using the above mentioned operations that represents a graph  $G$  in the obvious way.

b) The clique-width  $\text{cw}(G)$  of a graph  $G$  is the minimal  $k \in \mathbb{N}$  such that there exists a  $k$ -expression representing  $G$ .

It is well known that a graph of tree-width  $k$  has a clique-width bounded by  $2^{k+1} + 1$ , see [8]. Cographs, which will be important below in Section 4 can be characterized as being exactly the graphs of clique-width at most 2.

The main concern of this paper is to find small OBDD representations of adjacency functions of graphs for which one of the above parameters is bounded. As the following

easy example shows the converse cannot be achieved, i.e. the existence of a small sized OBDD representing the adjacency function of a graph does not imply any significant bounds on the tree- or clique-width of the graph.

*Example 1.* For  $n \in \mathbb{N}$ ,  $V := \{1, \dots, n\}^2$  the  $n \times n$  square grid is the graph  $G_n = (V, E)$ , where

$$((i_1, j_1); (i_2, j_2)) \in E \Leftrightarrow |i_1 - i_2| + |j_1 - j_2| = 1.$$

The clique-width of  $G_n$  and thus also the tree-width is unbounded as  $n$  tends to  $\infty$  [5]. However, an OBDD of size  $O(\log n)$  for  $f_{G_n}$  can easily be designed as follows. A vertex  $(i_1, j_1)$  of  $G_n$  is coded by the binary representations  $\text{bin}(i_1)$  and  $\text{bin}(j_1)$  of  $i_1$  and  $j_1$ . Next, for two vertices  $(i_1, j_1)$  and  $(i_2, j_2)$  the OBDD first reads alternately the bits of  $\text{bin}(i_1)$  and  $\text{bin}(i_2)$ . If both strings are the same the OBDD does the same with  $\text{bin}(j_1)$  and  $\text{bin}(j_2)$  and checks, whether  $|j_1 - j_2| = 1$ . This obviously can be done using an OBDD of size  $O(\log n)$ . Similarly, if  $\text{bin}(i_1) \neq \text{bin}(i_2)$  it has to be checked whether  $|i_1 - i_2| = 1$  and  $j_1 = j_2$ . The total OBDD size is  $O(\log n)$ . We leave it to the reader to work out detailed description of such an OBDD.

## 2.2 The Reduced Term of a $k$ -Expression

The construction of OBDDs representing graphs of bounded clique-width that we perform in Section 4 relies on results established in [12]. We recall these results before we turn to our constructions.

To each  $k$ -expression  $t$  representing a graph  $G$  there is a naturally related *reduced term*  $\text{red}(t)$ . It is basically obtained from  $t$  by deleting all information about the labeling. More precisely, define  $\mathcal{R}(V)$  to be the set of well-formed terms  $r$  written with the nullary symbols  $v$  for  $v \in V$ , the binary symbol  $\oplus$  and such that for every  $v \in V$  the term  $r$  contains exactly once the symbol  $v$ . Then for a  $k$ -expression  $t$  we have that  $\text{red}(t) \in \mathcal{R}(V)$ .

*Example 2.* For  $k = 3$  and  $V = \{v_1, \dots, v_7\}$  consider the 3-expression

$$t := \eta_{2,3}(((\rho_{2 \rightarrow 1}(\eta_{2,3}((\eta_{1,2}(1(v_1) \oplus 2(v_2)))) \oplus 3(v_3)))) \oplus 2(v_4)) \oplus (\eta_{1,3}(\eta_{1,2}(1(v_5) \oplus 2(v_6)))) \oplus 3(v_7))$$

Its reduced term is  $r = \text{red}(t) = (((v_1 \oplus v_2) \oplus v_3) \oplus v_4) \oplus ((v_5 \oplus v_6) \oplus v_7)$ . Each reduced term  $r = \text{red}(t)$  encodes a binary tree  $T_r$  whose leaves are precisely the nodes of  $G$ . The graph created by  $t$  and the binary tree related to  $\text{red}(t)$  are shown below.

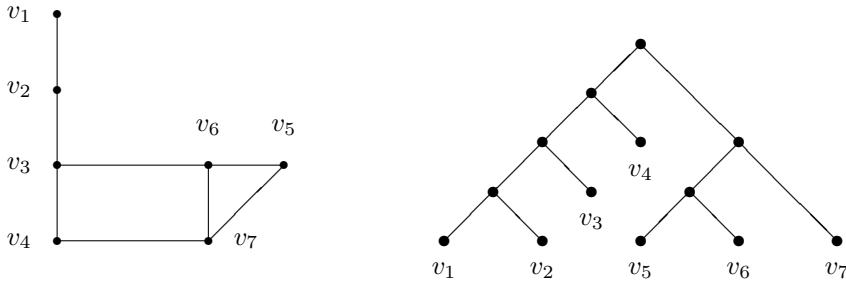


Figure 1

In what follows subterms of an  $r \in \mathcal{R}(V)$  and subtrees of  $T_r$  are important. For  $r \in \mathcal{R}(V)$  and a subterm  $s$  of  $r$  denote by  $V_s$  those vertices occurring in  $s$ . Define a graph  $H_s := (V_s, E_s)$ , where for  $u, v \in V_s$  there is an edge  $(u, v) \in E_s$  if and only if  $u$  and  $v$  have at least one different neighbor in  $G$  outside  $V_s$ , i.e.  $(u, v) \in E_s \Leftrightarrow N_G(u) \setminus V_s \neq N_G(v) \setminus V_s$ , where  $N_G(u)$  is the set of neighbors of  $u$  in  $G$ .

The following result from [12] collects those properties of the graphs  $H_s$  that will be important below.

**Theorem 6.** *Let  $G$  be a graph of clique-width  $k$ ,  $t$  a  $k$ -expression of  $G$ ,  $r = \text{red}(t)$  a reduced term and  $s$  a subterm of  $r$ .*

- a) *The graph  $H_s$  is a complete multipartite graph of at most  $k$  different components.*
- b) *If  $s = s_1 \oplus s_2$ , then every partite set of  $H_s$  arises as the union of partite sets of  $H_{s_1}$  and  $H_{s_2}$ .*

This result implies in particular that in order to figure out whether two different vertices in  $V_{s_1}$  and  $V_{s_2}$  are adjacent in  $G$  it is sufficient to analyze the relation between the at most  $k$  many partite sets in each of the graphs  $H_{s_1}$  and  $H_{s_2}$ .

### 3 Succinct Representations for Graphs of Bounded Tree-Width

Let  $G = (V, E)$  be a graph with  $V = \{1, \dots, n\}$  of tree-width at most  $k$  for some fixed  $k \in \mathbb{N}$ . In this section we prove that the adjacency function  $f_G$  has a succinct representation by an OBDD. More precisely, we shall assign to each vertex  $i \in V$  a name  $\text{name}(i)$  of length  $O(\log n)$  and construct an OBDD of size  $O(\log n)$  that represents  $f_G$  in the sense of Definition 2.

Suppose that  $G$  has a tree-decomposition  $(T, (X_\ell)_{\ell \in V_T})$  such that  $T$  is a binary tree of depth  $d$  and each  $X_\ell$  contains at most  $\tilde{k}$  vertices for some  $\tilde{k} \in \mathbb{N}$ . The values  $d$  and  $\tilde{k}$  will be specified below.

The main idea for constructing an OBDD of size  $O(\log n)$  is as follows. Let  $r$  denote the root of  $T$ . We shall assign to each vertex  $i$  an encoding  $\text{name}(i)$  that consists of three parts. The entire length of  $\text{name}(i)$  is  $O(\log n)$ . By the definition of a tree-decomposition there is a unique node  $\ell(i)$  of  $T$  with  $i \in X_{\ell(i)}$  that is closest to the root  $r$  of  $T$ . The first part of the encoding for  $i$  is the binary string encoding the unique path from  $r$  to  $\ell(i)$ . Its length is at most  $d$ . For reasons that become clear below we double each bit and add a pair 01 at the end. For example, a path 011 is encoded as 00111101. Denote this first part of  $\text{name}(i)$  by  $p(i)$ .

The second part of  $\text{name}(i)$  is simply taken as the binary representation  $\text{bin}(i)$  of  $i$  of length  $\lceil \log n \rceil$ .

The third part of  $\text{name}(i)$ , denoted by  $\text{adj}(i)$ , consists of the binary representation of at most  $\tilde{k} - 1$  other vertices. More precisely,  $\text{adj}(i)$  is the concatenation of  $\text{bin}(s)$  for all those vertices  $s \in X_{\ell(i)}$  that are adjacent to  $i$ . If  $X_{\ell(i)} \setminus \{i\}$  has less than  $\tilde{k} - 1$  many vertices we add dummy 0's at the end of  $\text{adj}(i)$  such that it has precisely length  $(\tilde{k} - 1) \cdot \lceil \log n \rceil$ . We order the  $\tilde{k} - 1$  strings in  $\text{adj}(i)$  with respect to the numerical value of the integers they represent from the largest to the smallest.

Altogether,  $\text{name}(i)$  is the concatenation  $p(i)\text{bin}(i)\text{adj}(i)$  and has length at most  $L := 2d + 2 + \tilde{k} \cdot \lceil \log n \rceil$ .

For arbitrary tree-decompositions of tree-width  $k$  we use the following theorem by Bodlaender [2] to obtain a situation as above with values  $d = O(\log n)$  and  $\tilde{k} = O(k)$ .

**Theorem 7.** ([2]) *Let  $G = (V, E)$  be a graph of tree-width  $k$  with  $n$  vertices. Then there exists a tree-decomposition  $(T, (X_\ell)_{\ell \in V_T})$  of  $G$  of width  $3k + 2$  such that  $T$  is a binary tree of depth at most  $2 \cdot \lceil \log_{\frac{5}{4}}(2n) \rceil$ .*

In the present section we only need the statement of the theorem. In Section 4 below we also have to study its proof more closely in the case where  $G$  is a tree.

Applying the theorem to our above reasoning gives a balanced tree-decomposition and an encoding of the vertices of  $G$  by names of length  $L := 4 \cdot \lceil \log_{\frac{5}{4}}(2n) \rceil + 2 + (3k + 2 + 1) \cdot \lceil \log n \rceil$ , which is  $O(\log n)$  for fixed  $k$ .

We are now ready to state the main theorem of this section.

**Theorem 8.** *Let  $k \in \mathbb{N}$  be fixed and let  $G = (V, E)$  be a graph of tree-width  $k$ . Define  $L$  as above. Then there is an OBDD  $O$  of size  $O(\log n)$  which computes a Boolean function  $f_O$  of  $2L$  input bits such that for all  $i, j \in V$  we have*

$$f_O(\text{name}(i), \text{name}(j)) = f_G(i, j) .$$

*Proof.* Let  $(T, (X_\ell)_{\ell \in V_T})$  be a tree-decomposition of  $G$  according to Theorem 7 with tree-width  $\tilde{k} - 1 = 3k + 2$  and depth logarithmic in  $n$ .

We construct an OBDD using the above encoding as names for the vertices of  $G$ . The variable ordering requirements for the OBDD will be obvious from the description below. For vertices  $i, j \in V$  the OBDD works in two steps.

**Step 1:** First, the paths  $p(i)$  and  $p(j)$  are inspected alternately in blocks of two bits each until we reach the end of one of the two paths. This is indicated by reading a consecutive block 01. There are three different cases to treat: Either none of the two paths is a prefix of the other. Then  $i$  and  $j$  cannot be adjacent in  $G$  due to the properties of a tree-decomposition. Or  $p(i)$  is a prefix of  $p(j)$  (of course when considering  $p(i)$  without the final 01 block), or vice versa. Both cases are handled using the same idea in Step 2.

**Step 2:** Without loss of generality suppose  $p(i)$  is a prefix of  $p(j)$  (the reverse situation is treated similarly in a parallel part of the OBDD). If  $p(i) = p(j)$ , then  $i$  and  $j$  occur in the same set  $X_{\ell(i)} = X_{\ell(j)}$  for the first time due to our convention about the tree-decomposition. They are adjacent in  $G$  iff this is already visible in the adjacency list related to  $i$  and  $j$ . If  $p(i) \neq p(j)$ , then by the same argument adjacency has to be visible in the adjacency list  $\text{adj}(j)$ . By ignoring intermediate inputs the OBDD continues by reading the input part for  $\text{bin}(i)$  and  $\text{adj}(j)$ . Thus, in both cases the OBDD has to pattern match  $\text{bin}(i)$  in  $\text{adj}(j)$ . This can be done by parallel bitwise comparison of the string  $\text{bin}(i)$  with each of the  $\tilde{k} - 1$  many strings in  $\text{adj}(j)$ . Since the strings in  $\text{adj}(j)$  are ordered numerically after reading each new bit the OBDD can maintain two numbers in  $\{1, \dots, \tilde{k}\}$  that indicate where a string in  $\text{adj}(j)$  has to be looked for in order to pattern match  $\text{bin}(i)$  - if at all such a string exists in  $\{1, \dots, \tilde{k}\}$ . There occur at most  $O(\tilde{k}^2 \cdot \log(n))$  different situations with respect to how the two numbers look like and which components in  $\text{bin}(i)$  still have to be read.<sup>1</sup> Similarly for

<sup>1</sup> Another way of proceeding would be to include  $\tilde{k} - 1$  copies of  $\text{bin}(i)$  in  $\text{name}(i)$  and then compare  $\text{bin}(i)$  with each string in  $\text{adj}(j)$  in parallel parts of the OBDD. Clearly, the encodings of the vertices would still use  $O(\log(n))$  bits.

$p(j)$  is a prefix of  $p(i)$ . Thus, the variable ordering can be chosen as taking alternately bitwise the strings  $bin(i)$  and the  $\tilde{k} - 1$  strings in  $adj(j)$  followed by the corresponding ordering for  $bin(j)$  and  $adj(i)$ . Depending on which path is a prefix of the other the OBDD ignores the corresponding other half of the variables.

The size of this OBDD can be estimated as follows: Step 1 can be done by an OBDD of size  $O(\min\{|p(i)|, |p(j)|\}) = O(\log n)$ . The pattern matching as described above in Step 2 can be implemented by an OBDD of size  $O(\log n)$  for fixed  $k$ . Note that the strings  $bin(i)$  and  $bin(j)$  are compared to  $O(k)$  many strings each. Thus, the number of potential subcases that might occur with respect to the question whether  $bin(i)$  is one of the strings in  $adj(j)$  is bounded as a function in  $k$ . The total size of the OBDD therefore is of order  $O(\log n)$  as  $k$  is fixed.  $\square$

## 4 OBDDs for Graphs of Bounded Clique-Width

Whereas graphs of bounded tree-width allow succinct representations by small OBDDs, this is in general not the case for graphs of bounded clique-width. Actually, the following lower and upper bounds are known for cographs, which are precisely the graphs of clique-width at most 2.

**Proposition 1.** ([14]) *a) There exist cographs  $G$  with  $n$  nodes such that each OBDD that computes the adjacency function of  $G$  and uses names of size  $\lceil \log n \rceil$  has size at least  $1.832 \cdot \frac{n}{\log n} - O(1)$ .*

*b) For each cograph  $G$  with  $n$  nodes there exists an OBDD of size at most  $3n \cdot \log n + 2n - \log n + \frac{1}{2}$  that computes the adjacency function of  $G$  using names of length  $\lceil \log n \rceil$  for the nodes.*

Thus, for OBDDs representing graphs of bounded clique-width we can guarantee a size at most of order  $O(\frac{n^2}{\log n})$  and in the worst case we have at least a size of order  $\Omega(\frac{n}{\log n})$ .

The purpose of this section is to come closer to this lower bound. We first show in Section 4.1 that each graph of clique-width  $k$  can be represented by an OBDD of size  $O(n \cdot f(k))$  for some function  $f$  only depending on  $k$  when using vertex encodings of length  $O(n \cdot \log k)$ . The result can be improved into the direction of using shorter encodings in case there exists a reduced term  $red(t)$  for a  $k$ -expression  $t$  representing  $G$  such that  $red(t)$  is balanced. Unfortunately, we do not know a general result guaranteeing such a reduced term to exist. For cographs, however, we can prove more using once again Theorem 7. This will be done in Section 4.2.

### 4.1 Representations for Graphs of Bounded Clique-Width using Long Vertex Encodings

Our constructions in this section are based on the results from Section 2.2.

**Theorem 9.** *Let  $k \in \mathbb{N}$  be fixed. Let  $G = (V, E)$  be a graph with  $n := |V|$  vertices and let  $t$  be a  $k$ -expression representing  $G$ . There is an OBDD  $O$  of size  $O(n)$  representing  $G$ . The encodings of  $G$ 's nodes used by  $O$  have length at most  $O(d \cdot \log k)$ , where  $d$  is the depth of the binary tree  $T_r$  encoded by  $r = red(t)$ .*

*Proof.* Let  $G = (V, E)$  be a graph of clique-width  $k$ ,  $t$  a  $k$ -expression representing  $G$ , and  $r = \text{red}(t)$  its reduced term, see Section 2.2. Denote the depth of the binary tree  $T_r$  related to  $r$  by  $d$ ; clearly  $d \leq n - 1$ . We first explain the encoding we use for each  $i \in V$ . Since all the nodes of  $G$  occur as leaves in  $T_r$  there is a unique path  $p(i) \in \{0, 1\}^*$  from  $T_r$ 's root to leaf  $i$ . The length of  $p(i)$  is at most  $d$ . Given two different graph nodes  $i, j \in V$  the first task is to find in  $T_r$  the final common tree node  $s$  for  $p(i)$  and  $p(j)$ .

The OBDD reads alternately the bits of  $p(i)$  and  $p(j)$  until  $s$  is found. Thereafter, it proceeds with a different subprogram for each  $s$ . The first part causes an OBDD size of  $O(n)$  since there are  $n - 1$  many internal nodes in  $T_r$ .

Consider the subtrees of  $T_r$  rooted at  $s_1$  and  $s_2$ , where  $s$  is the final common node on  $p(i)$  and  $p(j)$  and  $s_1$  is the left and  $s_2$  the right son of  $s$ .

Now the following observation is crucial with respect to the question whether  $i$  and  $j$  are adjacent in  $G$ : According to Theorem 6 both  $H_{s_1}$  and  $H_{s_2}$  are multipartite with  $\leq k$  partite sets; and the question whether a node  $i$  in one of the partite sets of  $H_{s_1}$  is adjacent in  $G$  to a node  $j$  in one of the partite sets of  $H_{s_2}$  only depends on those sets. Thus, we can code each of the  $\leq k$  sets by a string of length  $\lceil \log k \rceil$ . We then include the information to which set the vertex  $i$  belongs in  $H_{s_1}$  as part of the encoding of  $i$ ; more precisely, each bit of  $p(i)$  is followed by a string of length  $\lceil \log k \rceil$  coding the partite sets of the subsequent subgraph to which  $i$  belongs. The encoding of each node therefore has length  $O(d \cdot \log k)$ . Finally, for deciding adjacency of nodes  $i$  and  $j$  the OBDD has to find the splitting node as explained above and then to compute a Boolean function depending on  $2 \lceil \log k \rceil$  variables that represents the adjacency relations between the partite sets of  $H_{s_1}$  and  $H_{s_2}$ . Using the upper bound from Theorem 3 this can be achieved with a sub-OBDD of size  $(2 + o(1)) \cdot 4 \cdot \frac{k^2}{\log k}$ . Thus, we obtain an OBDD of size  $O(n \cdot \frac{k^2}{\log k})$  using encodings of each node of length at most  $O(d \cdot \log k)$ .  $\square$

If we do not know more about an optimal (with respect to the depth  $d$  of  $T_r$ ) reduced term  $r$  for  $G$  we can only conclude  $d < n$  and obtain vertex names of length  $O(n \cdot \log k)$ .

Recall how in the above proof the size of the OBDD depends on the depth of  $T_r$  and the number of its internal nodes. The depth of  $T_r$  enters into the length of the node encoding whereas the number  $n - 1$  of internal nodes always enters into the size estimate. We immediately obtain

**Corollary 1.** *Let  $G$  be a graph of bounded clique-width  $k$ ,  $t$  a  $k$ -expression of  $G$  such that the tree  $T_r$  related to  $r := \text{red}(t)$  has depth  $O(\log n)$ . Then there exists an OBDD of size  $O(n \cdot \frac{k^2}{\log k})$  that represents  $G$  and uses encodings of  $G$ 's nodes of length  $O(\log n \cdot \log k)$ .*

Currently, we do not know whether such a balanced term always exists for graphs of bounded clique-width. Balancing even at the cost of increasing the number of used labels might give better results. We thus pose the

**Problem 1.** Let  $G$  be a graph of order  $n$  and clique-width  $k$ . Is there always a  $\tilde{k}$ -expression  $t$  with  $\tilde{k} = O(f(k))$  for some function  $f$  representing  $G$  such that  $\text{red}(t)$  encodes a binary tree of depth  $O(\log n)$ ?

## 4.2 OBDDs representing Cographs

The goal of this section is to show that for cographs we can construct small sized OBDDs using vertex encodings of length  $O(\log n)$ . More precisely, the sizes of the OBDDs we design are of order  $O(n)$ , thus improving the result of Proposition 1, b) by a factor  $\log n$ . The proof relies on an application of the balancing algorithm behind Theorem 7 to the so called *cotree* that is related to each cograph.

**Definition 10** *A cograph  $G = (V, E)$  is a graph of clique-width at most 2. (This is equivalent to saying that  $G$  contains no induced  $P_4$ , i.e. no chordless path with four vertices and three edges.)*

**Proposition 2.** ([4]) *To each cograph  $G$  there exists an associated tree  $T(G)$  called cotree representing  $G$  as follows. The leaves of  $T(G)$  are precisely the vertices of  $G$ . The internal nodes of  $T(G)$  are labeled with either 0 or 1. Two vertices  $i, j$  of  $V$  are adjacent in  $G$  iff their least common ancestor  $\text{lca}(i, j)$  in  $T(G)$  is labeled with 1. Without loss of generality  $T(G)$  can be chosen to be a binary tree.*

The main result in this section is

**Theorem 11.** *For every cograph  $G = (V, E)$  with  $n$  vertices there exists an OBDD  $O$  representing  $G$  that has size  $O(n)$  and uses encodings of the vertices of  $G$  of length  $O(\log n)$ .*

*Proof.* Let  $G = (V, E)$  be a cograph and  $T(G)$  a binary cotree of  $G$  as explained in Proposition 2. According to a version of Theorem 7 dealing with trees and also proved in [2]  $T(G)$  has a tree decomposition of tree-width at most 3 whose underlying rooted tree  $T'$  is of depth  $d := 2\lceil \log_{\frac{5}{4}}(n) \rceil$ . In order to use  $T'$  for the design of an OBDD its construction from  $T(G)$  has to be studied more carefully. We thus recall the latter from [2], giving special emphasis to some additional information we encode in  $T'$ .

Balancing  $T(G)$  is based on the so called *parallel tree contraction* of Miller and Reif [13]. This contraction uses two basic operations RAKE and COMPRESS in order to contract a tree to a single vertex. The intermediate steps during the contraction process are used by Bodlaender to arrive at the desired tree decomposition.

Starting from  $T(G) =: T_0 = (V_0, E_0)$  the contraction process recursively constructs trees  $T_i = (V_i, E_i)$  for  $1 \leq i \leq r$  such that  $|V_r| = 1$  and  $r \leq 2\lceil \log_{\frac{5}{4}}(n) \rceil$  as follows. Each  $v \in V_i$  represents a subset  $\rho(v, i) \subseteq V$  of nodes of  $T(G)$  that after  $i$  steps are contracted to the one node  $v$ . Starting with  $\rho(v, 0) := \{v\}$  the tree  $T_{i+1} = (V_{i+1}, E_{i+1})$  and the new set  $\rho(v, i+1)$  are obtained from  $T_i$  by applying in parallel the following operations:

1. RAKE removes from each  $v \in V_i$  its children that are leaves in  $T_i$ . Then  $\rho(v, i+1) = \bigcup \{ \rho(w, i) \mid w = v, \text{ or } w \text{ is a child of } v \text{ in } T_i \text{ that is a leaf} \}$ .

2. COMPRESS contracts pairs of two subsequent nodes in a *chain* of  $T_i$ . A sequence of nodes  $v_1, \dots, v_k$  is a *chain* if  $v_{j+1}$  is the only child of  $v_j$  for  $1 \leq j \leq k-1$  and  $v_k$  has exactly one child in  $T_i$  not being a leaf. Then for each odd  $j$  the nodes  $v_j$  and  $v_{j+1}$  in a maximal chain are compressed to a single node  $w_j$  in  $T_{i+1}$ . It represents all the nodes previously contracted to either  $v_j$  or  $v_{j+1}$ , i.e.  $\rho(w_j, i+1) = \rho(v_j, i) \cup \rho(v_{j+1}, i)$ .

A tree decomposition of  $T(G)$  with underlying rooted tree  $T'$  is obtained by Bodlaender from the above procedure as follows. Here, we modify a bit Bodlaender's construction for our purposes.  $T'$  has  $r+1$  levels numbered bottom up from 0 to  $r$ . On the

bottom level (corresponding to  $T_0$  with the nodes  $V_0$  of  $T(G)$ ) tree  $T'$  has for each node of  $T(G)$  a box containing this node. Thus, the nodes of  $T(G)$  occur in  $T'$  as leaves. By convention we order the leaves of  $T(G)$  itself (which are the vertices of the given graph  $G$ ) in such a way that they occur as the first  $n$  leaves of  $T'$  from left to right. This will make it easy for the OBDD to detect inputs not encoding a vertex of  $G$ .

Since later on we need to mark one node in each box occurring on the upper levels of  $T'$  we mark each node of  $T(G)$  on the starting level 0 of  $T'$  in its corresponding box. The further levels of  $T'$  are now recursively determined according to the contracting operations used in Miller's and Reif's algorithm. Each level  $i$  (counted bottom up) in  $T'$  contains as many boxes as the tree  $T_i$  has nodes. Those boxes are connected in  $T'$  as described below:

- Suppose  $X_1, X_2, X_3$  are boxes on level  $i$  of  $T'$  that correspond to nodes in  $T_i$  such that  $X_2, X_3$  are children of  $X_1$  in  $T_i$  and at the same time  $X_2, X_3$  are leaves in  $T_i$ . Suppose furthermore that the nodes marked in these boxes are  $x_1, x_2$ , and  $x_3$ , respectively. Then on level  $i + 1$  of  $T'$  we take a box  $X$  containing the nodes  $x_1, x_2, x_3$  and mark  $x_1$  in this box. In  $T'$  we let  $X$  be a father (on level  $i + 1$ ) of  $X_1, X_2$  and  $X_3$ ;
- similarly, if  $X_1, X_2, X_3$  are boxes on level  $i$  of  $T'$  as above,  $X_2, X_3$  children of  $X_1$  in  $T_i$ , but only  $X_2$  is a leaf in  $T_i$ , then on level  $i + 1$  of  $T'$  we still have box  $X_3$  with  $x_3$  marked as well as a new box  $X := \{x_1, x_2\}$  with  $x_1$  marked.  $X$  (on level  $i + 1$ ) will be father of  $X_2$  (on level  $i$ ) and  $X_3$  (on level  $i + 1$ ) father of  $X_3$  (on level  $i$ );
- thirdly, if  $X_1, X_2$  are boxes on level  $i$  of  $T'$  corresponding to two nodes in  $T_i$  that are compressed in  $T_{i+1}$ ,  $x_1$  marked in  $X_1$  and  $x_2$  marked in  $X_2$ , and if  $X_1$  is father of  $X_2$ , then on level  $i + 1$  of  $T'$  we include a box  $X = \{x_1, x_2\}$  with  $x_1$  marked such that  $X$  in  $T'$  is father of  $X_1$  and  $X_2$ .
- finally, if a node is not changed when going from  $T_i$  to  $T_{i+1}$ , then the same box is used on both levels of  $T'$  and connected by an edge.

For designing the desired OBDD we encode each node  $i$  in the vertex set  $V$  of  $G$  by a bitstring of length  $O(\log n)$ . One part of  $i$ 's encoding name will be the path  $path(i)$  from  $T'$ 's root to the leaf  $i$ . Since  $T'$  has depth  $d \leq 2\lceil \log_{\frac{5}{4}}(n) \rceil$  this quantity bounds as well the length of this first part of the encoding.

In order to make  $T'$  providing all the information about  $G$ 's vertices that we need, we have to attach some additional information to each box occurring along  $path(i)$  for each  $i \in V$ . It is here where the labels of the original cotree representing the cograph  $G$  are important. The allover idea to decide whether two vertices  $i, j$  of  $G$  are adjacent in  $G$  is to find the final common box along  $path(i)$  and  $path(j)$  in  $T'$  and then include some additional information obtained from  $T(G)$ 's labeling.

Towards this aim first note that each node (box)  $X$  in  $T'$  corresponds uniquely to a subtree of  $T(G)$  that is contracted to this node. Therefore, the label of the root of that subtree in  $T(G)$  is naturally related to the node of  $T'$  (this is not yet the information we are looking for!). We denote this original label by  $ol(X)$  and use those labels in order to compute bottom up a further label  $lab(i, \ell)$  for each box  $X_\ell$  in  $T'$  that occurs along  $path(i)$ . These new labels will be included as part of the encoding of vertex  $i$ . The first operation that involves leaf  $i$  in  $T'$  is a RAKE operation that contracts leaf  $i$  with an internal node  $v$  of  $T(G)$ . If  $X_1$  is the corresponding node on level 1 in  $T'$  (including the nodes  $i$  and  $v$  with  $v$  marked) we take the original label  $ol(X_1)$  as the value for

$lab(i, 1)$ . Now recursively we define each component of the string  $label(i) \in \{0, 1\}^{r+1}$  in relation with the unique box along  $path(i)$  on the corresponding level of  $T'$ . If the label  $lab(i, \ell)$  on level  $\ell$  is determined and the corresponding box  $X_{\ell+1}$  on level  $\ell + 1$  was obtained from a RAKE operation, then we put  $lab(i, \ell + 1) = ol(X_{\ell+1})$ . This is justified because if  $path(i)$  and  $path(j)$  have  $X_{\ell+1}$  as the final common component, then  $ol(X_{\ell+1})$  is the label of their least common ancestor in  $T(G)$ . If label  $lab(i, \ell)$  on level  $\ell$  was determined and the corresponding box  $X_{\ell+1}$  on level  $\ell + 1$  was obtained from a COMPRESS operation, then we put  $lab(i, \ell + 1) = lab(i, \ell)$ . In this situation, we add one additional bit of information denoted by  $upper(i, \ell + 1)$ . It gets value 1 if box  $X_\ell$  was the upper box among the two boxes compressed, otherwise we put  $upper(i, \ell + 1) = 0$ . Again, if for two vertices  $i, j$  of  $G$   $path(i)$  and  $path(j)$  split in node  $X_{\ell+1}$ , then the labeling of the upper node is the decisive one for deciding adjacency. If for example  $upper(i, \ell + 1) = 1$ ,  $upper(j, \ell + 1) = 0$ , then  $lab(i, \ell + 1)$  is the label of the least common ancestor of  $i, j$  in  $T(G)$ .

This finishes the description of the encoding of the vertices. The entire encoding  $name(i)$  is the concatenation of  $path(i), lab(i, \ell)$  for all  $1 \leq \ell \leq 2 \lceil \log_{\frac{5}{4}}(n) \rceil$  and of  $upper(i, \ell)$  for the corresponding levels  $\ell$ . For two inputted vertices  $i, j$  the OBDD reads the encodings of both alternately top-down until the splitting box  $X_k$  of  $path(i)$  and  $path(j)$  is found. Then it inspects  $lab(i, k)$  and  $lab(j, k)$ . If both are equal the label corresponds to the correct answer. If both are different  $X_k$  must have arisen from a COMPRESS operation. Then the OBDD tests whether  $upper(i, k) = 1$ . If yes it returns  $lab(i, k)$ , else it returns  $lab(j, k)$ .

The length of the used encodings is at most  $6 \cdot \lceil \log_{\frac{5}{4}}(n) \rceil = O(\log n)$ . The size of the OBDD is basically determined by the number of splitting points between two paths  $path(i)$  and  $path(j)$ , which in turn corresponds to the number  $n$  of vertices of  $G$ . Then, some additional gates are necessary to decide the label and to stop the computation for inputs that do not properly encode a vertex of the original graph. Clearly, the size of the OBDD thus is of order  $O(\log n)$ . This finishes the proof.  $\square$

## 5 Acknowledgement

This work was done while K. Meer spent a sabbatical at the Forschungsinstitut für Diskrete Mathematik at the University of Bonn, Germany. The hospitality during the stay is gratefully acknowledged.

K. Meer is partially supported by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778 and by the Danish Natural Science Research Council SNF. This publication only reflects the authors' views.

## References

1. S. Arnborg, J. Lagergren, D. Seese: *Easy problems for tree decomposable graphs*, Journal of Algorithms 12, pp. 308–340 (1991).
2. H.L. Bodlaender: *NC-algorithms for graphs with small tree-width*, In: Proceedings Graph-theoretic concepts in computer science, Lecture Notes in Comput. Sci., vol. 344, Springer, pp. 1–10 (1989).
3. Y. Breitbart, H. Hunt, D. Rosenkrantz: *On the size of binary decision diagrams representing Boolean functions*, Theoretical Computer Science, Vol. 145, Nr. 1, pp. 45–69 (1995).

4. D.G. Corneil, H. Lerchs, L. Stewart Burlingham: *Complement reducible graphs*, Discrete Applied Mathematics 3, pp. 163–174 (1981).
5. B. Courcelle: *Graph grammars, monadic second-order logic and the theory of graph minors*, Contemporary Mathematics 147, pp. 565–590 (1993).
6. B. Courcelle, J.A. Makowsky, U. Rotics: *Linear Time Solvable Optimization Problems on Graphs of Bounded Clique Width*, Theory of Computing Systems, vol. 33(2), pp. 125–150 (2000).
7. B. Courcelle, M. Mosbah: *Monadic second-order evaluations on tree decomposable graphs*, Theoretical Computer Science 109, pp. 49–82 (1993).
8. B. Courcelle, S. Olariu: *Upper bounds to the clique-width of graphs*, Discrete Applied Mathematics 101, pp. 77–114 (2000).
9. R. Diestel: *Graph Theory*, Springer Graduate Texts in Mathematics, 2nd edition (2000).
10. J. Feigenbaum, S. Kannan, M.Y. Vardi, M. Viswanathan: *Complexity of Problems on Graphs Represented as OBDDs*, Chicago Journal of Theoretical Computer Science, vol. 1999, issue 5/6, pp. 1–25 (1999).
11. G.D. Hachtel, F. Somenzi: *A symbolic algorithm for maximum flow in 0-1 networks*, Formal Methods in System Design 10, pp. 207–219 (1997).
12. V. Lozin, D. Rautenbach: *The Relative Clique-Width of a graph*, Rutcor Research Report RRR 16-2004 (2004).
13. G. Miller, J. Reif: *Parallel tree contraction and its application*, Proc. 26th Foundations of Computer Science FOCS 1985, IEEE, pp. 478–489 (1985).
14. R. Nunkesser, P. Woelfel: *Representation of Graphs by OBDDs*, Proceedings of ISAAC 2005, X. Deng and D. Du (eds.), Lecture Notes in Computer Science 3827, Springer, pp. 1132–1142 (2005).
15. D. Sawitzki: *Implicit flow maximization by iterative squaring*, Proceedings of 30th SOFSEM, P. van Emde Boas, J. Pokorny, M. Bielikova, and J. Stuller (eds.), Lecture Notes in Computer Science 2932, Springer, pp. 301–313 (2004).
16. D. Sawitzki: *A symbolic approach to the all-pairs shortest paths problem*, Proceedings of 30th Graph-Theoretic Concepts in Computer Science WG 2004, J. Hromkovic, M. Nagl, B. Westfechtel (eds.), Lecture Notes in Computer Science 3353, Springer, pp. 154 – 167 (2004).
17. I. Wegener: *Branching Programs and Binary Decision Diagrams: Theory and Applications*, SIAM Monographs on Discrete Mathematics and Applications, SIAM (2000).
18. I. Wegener: *BDDs – design, analysis, complexity, and applications*, Discrete Applied Mathematics 138, pp. 229–251 (2004).
19. P. Woelfel: *Symbolic topological sorting with OBDDs*, Journal of Discrete Algorithms, to appear.