Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 157 (2009) 247–261
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Representation of graphs by OBDDs

Robin Nunkesser[a,*], Philipp Woelfel[b]

[a] *TU Dortmund, Department of Computer Science, 44221 Dortmund, Germany*
[b] *University of Toronto, Department of Computer Science, 10 King's College Road, Toronto M5S 3G4, Canada*

Received 24 March 2006; received in revised form 13 February 2008; accepted 19 February 2008
Available online 14 April 2008

Abstract

Recently, it has been shown in a series of works that the representation of graphs by Ordered Binary Decision Diagrams (OBDDs) often leads to good algorithmic behavior. However, the question for which graph classes an OBDD representation is advantageous, has not been investigated, yet. In this paper, the space requirements for the OBDD representation of certain graph classes, specifically cographs, several types of graphs with few P_4s, unit interval graphs, interval graphs and bipartite graphs are investigated. Upper and lower bounds are proven for all these graph classes and it is shown that in most (but not all) cases a representation of the graphs by OBDDs is advantageous with respect to space requirements.

Keywords: Graph representation; Interval graph; Graphs with few P_4s; OBDD

1. Introduction

Some modern applications require such huge graphs that the usual, explicit representation by adjacency lists or adjacency matrices is infeasible. For example, a typical state transition graph arising in the process of verification and synthesis of sequential circuits may consist of 10^{27} vertices and 10^{36} edges. Graphs modeling street or railway networks are usually not so large, but even those graphs become huge if they are interlinked with other components, as for example amount of traffic, time slots, etc.

When dealing with huge graphs, one faces two problems:

(1) Algorithms with polynomial or even linear running times, may not be feasible.
(2) The space needed to store the adjacency matrices or adjacency lists may exceed the available amount of memory.

In order to solve the second problem, one can sometimes use an *implicit* representation of a graph. The typical approach, as introduced by Kannan, Naor and Rudich [13], is to store the vertices in such a way that adjacency is uniquely determined by the vertex labels (a prominent example is the representation of interval graphs, where the nodes are labeled by intervals and two nodes are adjacent if and only if the corresponding intervals intersect). While such implicit representations can lead to good graph compression, they do not solve Problem (1), concerning running times of algorithms. Moreover, such implicit graph representations are ad hoc constructions, i.e., in order to use such implicit graph representations, the algorithm has to "know" with which graph class it has to deal. Hence, for general applications that have to work with arbitrary input graphs, such implicit representations cannot be used in a uniform way.

* Corresponding author. Fax: +49 2317552047.
E-mail addresses: robin.nunkesser@udo.edu (R. Nunkesser), pwoelfel@cs.toronto.edu (P. Woelfel).

0166-218X/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2008.02.012

Another approach to deal with Problem (2), is to use a *generic* data structure, that can represent any graph, but where sufficiently structured graphs may be compressed, i.e., their representations requires only sublinear space (with respect to the number of vertices or edges). Since the adjacency matrix can be considered as a Boolean function, data structures that store Boolean functions can be used for this purpose. One of the most prominent data structures for storing Boolean functions is the *Ordered Binary Decision Diagram* (OBDD), because for all important operations on Boolean functions (e.g. the synthesis operation, substitution by constants or satisfiability test) efficient algorithms are known. These OBDD operations allow to efficiently convert *any* graph represented by, e.g., an adjacency list or an adjacency matrix to a corresponding OBDD.

Another advantage of using OBDDs is, that they may help to cope with Problem (1). Several graph problems can be solved with a polylogarithmic number of OBDD operations, if the input graphs are represented by OBDDs. Examples are network flow maximization [9,18], topological sorting [23] or finding shortest paths in networks [19]. It is important to emphasize that the number of OBDD operations is not directly proportional to the running time of the algorithm, as the running time for one OBDD operation depends on the sizes of the OBDDs on which these operations are performed. Hence, such OBDD graph algorithms are highly heuristic, and we can only expect to obtain advantages with respect to the running times, if all OBDDs encountered during the run of such an algorithm remain small in size. (For example, on a random graph we would expect no speedup from OBDD algorithms.) However, in practice this heuristical approach has been very successful (see for example [4,9]). Moreover, it has been proven for very structured graphs, such as the grid graph, that some OBDD algorithms such as topological sorting [23] and network flow maximization [18] have only polylogarithmic running times (with respect to the number of grid vertices).

It is clear that we cannot achieve sublinear running times, if the representation of the input graph already requires OBDDs of linear size. Therefore, in this paper we start to investigate the question, for which natural graph classes a compression of input graphs can be achieved by using OBDDs. There are several parameters that may have significant influence on the size of an OBDD for one specific graph (we discuss these after defining OBDDs). But in this paper we work under the premise that an OBDD representation is used in general OBDD graph algorithms, i.e., OBDD algorithms that work with any input graph, but where (naturally) the running time depends on the structure of that input graph.

Specifically, we show that cographs, several types of graphs with few P_4s, unit interval graphs, and interval graphs can be compressed significantly, if represented by OBDDs. On the other hand, we show that for bipartite graphs no significant compression can be achieved.

Note that following the conference version of the present paper, Meer and Rautenbach [14] have improved some of our upper bounds by being less restrictive with respect to some of the parameters in the underlying OBDD model. While this relaxed model seems less suitable for OBDD algorithms, their result shows that even better compression factors can be achieved with OBDD representations, if one is only interested in solving Problem (2).

It should be emphasized, that our upper bound results, indicating that graphs from the graph classes mentioned above may be compressed significantly, do not automatically imply that OBDD algorithms are fast on these graphs. But they provide evidence that OBDD algorithms at least have a chance of being superior for sufficiently structured input graphs. On the other hand, our negative results (for example for bipartite graphs) show that OBDD algorithms on such graphs will not have an advantage over non-symbolic algorithms.

1.1. Graph representation by OBDDs

In the following, let B_n denote the class of Boolean functions $f : \{0, 1\}^n \to \{0, 1\}$. OBDDs have been introduced by Bryant in 1986 [3] as a data structure for Boolean functions.

Definition 1. Let $X_n = \{x_1, \ldots, x_n\}$ be a set of Boolean variables.

(1) A *variable ordering* π on X_n is a bijection $\pi : \{1, \ldots, n\} \to X_n$, leading to the ordered list $\pi(1), \ldots, \pi(n)$ of the variables.
(2) A π-OBDD on X_n for a variable ordering π is a directed acyclic graph with one root, two sinks labeled with 0 and 1, respectively, and the following properties: Each inner node is labeled by a variable from X_n and has two outgoing edges, one of them labeled by 0 (or drawn as a dashed line), the other by 1 (or drawn as a solid line). If an edge leads from a node labeled by x_i to a node labeled by x_j, then $\pi^{-1}(x_i) < \pi^{-1}(x_j)$.

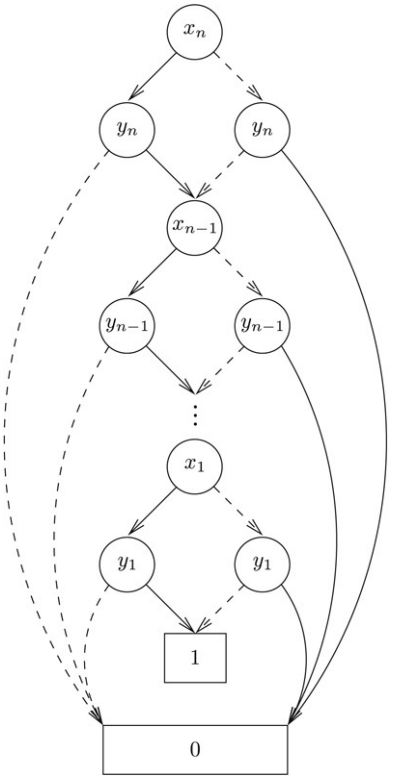

Fig. 1. Example of an OBDD representing $f(x, y) \in B_{2n}$ with $f(x, y) = 1 \Leftrightarrow \forall i \in 1, \ldots, n : x_i = y_i$.

(3) For any input $a = (a_1, \ldots, a_n) \in \{0, 1\}^n$ the *computation path* of a is the unique root-to-sink path leading from any x_i-node over the edge labeled by the value of a_i. A π-OBDD *represents* the Boolean function $f \in B_n$, where for all $a \in \{0, 1\}^n$ the function value $f(a)$ is the label of the sink reached on the computation path of a.
(4) The *size* of a π-OBDD G is the number of its nodes and is denoted by $|G|$.
(5) The π-OBDD *size* of a Boolean function f (π-OBDD(f)) is the minimal size of a π-OBDD computing f. The OBDD *size* of a Boolean function f (OBDD(f)) is the minimal size of a π-OBDD computing f for some variable ordering π.

Fig. 1 shows a π-OBDD with $(\pi(1), \ldots, \pi(2n)) = (x_n, y_n, \ldots, x_1, y_1)$ representing $f(x, y) \in B_{2n}$ with $f(x, y) = 1 \Leftrightarrow \forall i \in 1, \ldots, n : x_i = y_i$.

We denote the binary value represented by the n-bit string $z_{n-1} \ldots z_0 \in \{0, 1\}^n$ by $|z| := \sum_{i=0}^{n-1} z_i 2^i$. Conversely we denote by $[k]_n$, $n \geq \lceil \log(k + 1) \rceil$, the n-bit string representing k, i.e. the string $z_{n-1} \ldots z_0 \in \{0, 1\}^n$ with $k = |z|$.

Let $G = (V, E)$ be a graph and let $N = |V|$. We can represent the edge set E of G by an OBDD for the characteristic function $\chi_E : E \to \{0, 1\}$ of E, where $\chi_E(v_1, v_2) = 1 \Leftrightarrow \{v_1, v_2\} \in E$. In order to encode the vertices by Boolean variables, we use the convention that $V = \{[0]_n, \ldots, [N - 1]_n\}$, where $N > 0$ and $n = \lceil \log N \rceil$. If N is not a power of two we may need to store N by some n-bit integer. Another possibility is to use an OBDD for storing the characteristic function χ_V of V. It is easy to see that for all variable orderings π, the π-OBDD size of the function χ_V is $\mathcal{O}(n \log n) = \mathcal{O}(\log N \log \log N)$.

Note that given a variable ordering π, and a π-OBDD for a function f, it is possible to efficiently (in almost linear time) find the minimal π-OBDD for f (see for example [22]). Therefore, we are only interested in the minimal π-OBDD for some graph.

Definition 2. The π-OBDD *size* of a graph $G = (V, E)$ (π-OBDD(G)) is the minimal size of a π-OBDD computing χ_E for some labeling of the vertices. Analogously, the OBDD *size* of G (OBDD(G)) is the minimum of π-OBDD(G) for all variable orderings π. The *worst-case OBDD size* of a graph class $\mathcal{G}$ is the maximum of OBDD(G) over all $G \in \mathcal{G}$.

Several parameters can influence the OBDD size of a graph. First of all, the π-OBDD size of any Boolean function is highly sensitive to the variable ordering π. Finding the optimal variable ordering is a hard problem [20]. Nevertheless in practice heuristics are often used successfully to find variable orderings with small OBDD sizes (see [22] for a description of several approaches).

Secondly, the vertex labeling has a significant influence on the OBDD size. Even very structured graphs, such as lines or grids, will require large OBDDs if the vertices are labeled at random. However, OBDDs can only be advantageous, if the inputs are not random, so the hope is that if the input is a sufficiently structured graph, as for example a grid-like graph, then the vertex labeling is also "nice". We hope that future research will lead to heuristics, or even efficient approximation algorithms that compute good vertex labelings. Therefore, our upper bound results mainly indicate a principal feasibility of OBDD algorithms.

Finally, increasing the domain from which the vertex labelings are chosen can lead to a reduction of the OBDD size. For example, it is not hard to see that the vertices for any graph G can be labeled with $O(N)$ bits, each, such that $\mathrm{OBDD}(G) = O(N)$. In fact, following our conference version of this paper, Meer and Rautenbach [14] showed that our upper bounds on the OBDD sizes of cographs can be improved, if one allows $c \cdot \log N$ bits for the vertex encoding, for some sufficiently large constant c. Similarly, significantly better upper bounds than those derived in this paper can easily be obtained for interval graphs, by simply using $2\lceil \log N \rceil$ bits for each vertex label.

While such an approach can be valid, if graph compression is the only goal (see Problem (2) above), it causes problems with OBDD algorithms. One reason is that if a domain of cardinality more than N is used for the vertex labels, then the vertex labels have to be stored in some additional data structure. This causes incompatibility with the existing OBDD algorithms. It also seems reasonable to assume (although we do not have any hard evidence) that the algorithmic problems of finding a good vertex labeling and a good variable ordering is harder for larger domain sizes. Moreover, the OBDD size of a Boolean function in n bits can be as large as $\Omega(2^n/n)$. Hence, the worst-case running time of any OBDD graph algorithm is essentially a term with an exponent proportional to the number of bits used for the vertex labelings. Although we think of OBDD algorithms as heuristics, that are hopefully very efficient for reasonably structured inputs, it still make sense to keep the worst-case running time as small as possible. But even just for the graph compression problem, a "tight" domain size of N, where no additional data structure for storing the vertex labels is needed, makes sense in many circumstances: In a pure heuristical setting one might hope for space bounds of $o(N \log N)$, but an explicit data structure for identifying the vertices in a graph that uses labels from a domain of cardinality M, would require at least $\Omega\left(\log\binom{M}{N}\right)$ bits.

1.2. General upper bounds

Breitbart, Hunt III and Rosenkrantz have shown in [2] that any function $f \in B_n$ can be represented by a π-OBDD of size $(2 + \mathcal{O}(1))\, 2^n/n$ (for any variable ordering π). Hence, the π-OBDD size of any graph with N vertices is bounded by $(4 + \mathcal{O}(1))\, N^2/\log N$, because $n < 2 + 2\log N$ for appropriately chosen vertex labels. In order to obtain a general bound which is better for not very dense graphs, we may use the fact that the π-OBDD size of any function $f \in B_n$ is bounded by roughly $n \cdot |f^{-1}(1)|$.

Proposition 3. *Let $G = (V, E)$ be a graph, $N = |V|$ and $M = |E|$. The OBDD size of G is bounded above by*

$$\min\left\{(4 + \mathcal{O}(1))\, N^2/\log N,\ 2M \cdot (2\lceil \log N \rceil - \lfloor \log 2M \rfloor + 1) + 1\right\}.$$

Proof. An OBDD B representing G has depth $2\lceil \log N \rceil$ (because the input consists of two binary coded vertices). Moreover, exactly $2M$ paths lead from the root to the 1-sink. In a minimal π-OBDD, there is no node from which all paths lead to the 0-sink (because otherwise this node can be removed and all edges can be redirected to the 0-sink). Hence, each level of the OBDD has at most $2M$ vertices (the ith level of a π-OBDD is the set of nodes labeled with $\pi(i)$). However, the first $\lfloor \log(2M) \rfloor$ levels cannot have more vertices than a complete binary tree of depth $\lfloor \log(2M) \rfloor$. Therefore, B has at most $2^{\lfloor \log(2M) \rfloor} - 1 \leq 2M - 1$ nodes in the first $\lfloor \log(2M) \rfloor$ levels altogether, and at most $2M$ nodes on each of the following levels, and two sinks. This yields $|B| \leq 2M\,(2\lceil \log N \rceil - \lfloor \log(2M) \rfloor) + 2M + 1$. □

Note that if a graph G with N vertices and M edges is given by an adjacency matrix or adjacency list, then the OBDD for G satisfying the size bound of the above Proposition can be constructed in time $\mathcal{O}(M \log N \log^2 M)$. Hence, OBDDs are a truly generic graph representation.

Obviously, an OBDD B can be uniquely described by $\mathcal{O}(|B| \cdot (\log |B| + \log n))$ bits. Consider a graph G with N vertices and M edges. The above proposition shows that an OBDD representation for dense graphs needs at most $\mathcal{O}(N^2)$ bits and thus is only a constant factor larger than the representation by adjacency matrices. Since $\mathcal{O}(M \log N(\log M + \log\log N))$ bits are sufficient to store G, an OBDD representation is more space efficient than an adjacency matrix for not very dense graphs. An adjacency list representation of G needs $\mathcal{O}((N + M) \log N)$ bits. However, an adjacency test may take linear time in the worst case. Here, OBDDs are more efficient because adjacency can be tested in $\mathcal{O}(\log N)$ time (start at the root of the OBDD and traverse the graph according to the input until the sink is found). Hence, the space requirements of OBDDs are only little worse than that of adjacency lists and much better than that of adjacency matrices for sparse graphs, while the adjacency test is less efficient than that of a matrix representation but much better than that of a list representation in the worst case.

But Proposition 3 covers only the worst case. We mainly use OBDDs because we hope that if graphs are sufficiently structured, then a good "compression" may be achieved and thus the OBDD representation may be more space efficient than any explicit representation. One well-known example where this is the case in the grid graph which can even be represented with OBDDs of logarithmic size. However, we are interested in the representational power of OBDDs for much more general graph classes. In the following we will show that several very natural and large graph classes have OBDD sizes which yield a much better space behavior than that of explicit representations.

In the following Section we derive a general formula which allows us to prove lower bounds for the OBDD size of certain graph classes with counting arguments. In Section 3 we investigate several types of graph classes which do not contain many P_4s, most prominently cographs. We show that they can be represented by OBDDs of size $\mathcal{O}(N \log N)$ and thus an OBDD representation outperforms any explicit representation with respect to space usage if the graphs are not very sparse. On the other hand, our bound is optimal up to a polylogarithmic factor because – as we show – there are cographs which require OBDDs of size $\Omega(N/\log N)$. In Section 4 we investigate interval graphs. First, we show that unit interval graphs have an OBDD size of $\mathcal{O}(N/\sqrt{\log N})$ and give an upper bound for the OBDD size of general interval graphs of $\mathcal{O}(N^{3/2}/\log^{3/4} N)$. We also give lower bounds of $\Omega(N/\log N)$ and $\Omega(N)$, respectively. Finally, in Section 5 we try to find a natural graph class which is very hard to represent for OBDDs. We show that the representation of some bipartite graphs requires OBDDs of size $\Omega(N^2/\log N)$. Hence, the OBDD representation of bipartite graphs is not necessarily more space efficient than that of an adjacency matrix representation.

2. Lower bounds with counting arguments

In order to prove lower bounds we mainly use counting arguments. Wegener has shown in [22] that OBDDs of size s can compute at most $sn^s\,(s+1)^{2s}/s! = 2^{s\log s + s\log n + \mathcal{O}(s)}$ different functions $f \in B_n$. Thus, in order to achieve a lower bound for the OBDD size of graphs from a graph class $\mathcal{G}$ it suffices to count the number $N_{\mathcal{G}}(N)$ of unlabeled graphs with N vertices in $\mathcal{G}$.

Corollary 4. *Let $\mathcal{G}$ be a graph class that allows the addition of isolated vertices and let $s : \mathbb{N} \to \mathbb{R}$ be a function. If*

$$\lim_{N\to\infty} \left(2^{s(N)\log s(N) + s(N)\log\log N + \mathcal{O}(s(N))} \cdot \left(N_{\mathcal{G}}(N)\right)^{-1}\right) < 1,$$

then there are $N \in \mathbb{N}$ such that there are graphs with N vertices that cannot be represented by OBDDs of size $s(N)$ or less.

Proof. If the limit above is less than 1, then for large enough N there are more graphs than OBDDs for functions in $n = 2 \cdot \lceil \log N \rceil$ variables. □

3. Graphs with few induced P_4s

Hereinafter P_4 denotes a chordless path with four vertices and three edges. Many graphs with few induced P_4s have common properties such as a unique (up to isomorphism) tree representation. Starting from the tree representation developed by Lerchs (see e.g. [5]) of the well-known class of *cographs* (graphs with no induced P_4), Jamison and Olariu have developed and studied tree representations for various graph classes with few induced P_4s such as P_4-*reducible graphs* [10], P_4-*extendible graphs*, [11] and P_4-*sparse graphs* [12]. P_4-*reducible graphs* contain no vertex, that belongs to more than one induced P_4. P_4-*extendible graphs* contain at most one additional vertex for each induced

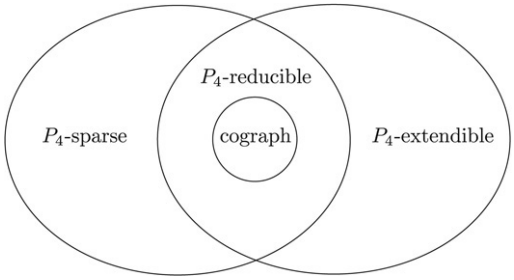

Fig. 2. A graph class inclusion diagram for the considered graph classes.

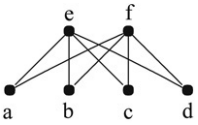

Fig. 3. Example of a cograph (constructed by $+(\cup(a, b, c, d), \cup(e, f))$).

P_4p that induces a different P_4 together with three vertices from p. In *P_4-sparse graphs* every set of five vertices induces one P_4 at most. In the following discussion we omit P_4-reducible graphs, because their class is the intersection of the classes of P_4-extendible and P_4-sparse graphs. The relation between the considered graph classes is depicted in Fig. 2.

All these graph classes have in common that they can be constructed from single vertex graphs by graph operations, each of them joining several vertex disjoint graphs together. Consider for example two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ on two disjoint vertex sets. Then the *union* of G_1 and G_2 is $G_1 \cup G_2 := (V_1 \cup V_2, E_1 \cup E_2)$ and the *join* is $G_1 + G_2 := (V_1 \cup V_2, E_1 \cup E_2 \cup E')$, where E' contains all edges $\{v_1, v_2\}$ with $v_1 \in V_1$ and $v_2 \in V_2$. It is well known that any *cograph* can be obtained from single vertex graphs by a sequence of union and join operations (see e.g. [5]; Fig. 3 shows an example).

Now consider a set Ω of operations, each of them joining several vertex disjoint graphs together. Note, that the operations need not be commutative like union and join. Further assume that a graph G can be constructed from single vertex graphs by using only the joining operations in Ω. Then G has a natural representation as a rooted tree $T(G)$ which can be constructed recursively as follows: If G is a single vertex graph, consisting of the vertex v, then $T(G) = v$. If $G = \omega(G_1, \ldots, G_k)$ for an operation $\omega \in \Omega$, then the root of $T(G)$ is a vertex labeled with ω whose children are the roots of the trees $T(G_1), \ldots, T(G_k)$ (note that the order of the children may be important for noncommutative operations).

Such a tree representation of a graph is very helpful if we want to devise an algorithm deciding adjacency which can then be turned into an OBDD for the graph. For example, if G is a cograph, then two vertices v_1 and v_2 are adjacent if and only if the least common ancestor of v_1 and v_2 in $T(G)$, $\mathrm{lca}(v_1, v_2)$, is labeled with $+$. (The least common ancestor of two nodes v_1 and v_2 is unique and is defined as the deepest node that is an ancestor of both v_1 and v_2.)

Hence, for the algorithm it suffices to determine the lca of v_1 and v_2. For P_4-extendible graphs and P_4-sparse graphs adjacency is not so simple to determine. However, in the following we develop a new tree representation for these graphs such that adjacency of two vertices can be determined by computing the lca of two vertices and some additional information. Later, we show how to compute the lca and the additional information with OBDDs.

Recall that $V = \{[0]_n, \ldots, [N-1]_n\}$. We label the vertices for our representation in such a way that $|v_1|$ is less than $|v_2|$ for two vertices v_1, v_2, if a preorder traversal of $T(G)$ traverses the leaf corresponding to v_1 first. Furthermore, for two vertices v_1, v_2 of a graph $G = (V, E)$ with a tree representation $T(G)$ let $\delta_d(v_1, v_2)$ be $||v_1| - |v_2||$, if $||v_1| - |v_2|| \leq d$ and 0 otherwise. Let $c : V \to \mathbb{N}$ be the function with $c(v) = i$ if the vertex v is the ith child of its parent in $T(G)$.

Lemma 5. *Let $\mathcal{G}$ be the class of either cographs, P_4-sparse graphs or P_4-extendible graphs. Then there is a tree representation $T(G)$ for all graphs $G = (V, E) \in \mathcal{G}$ such that for any two vertices $v_1, v_2 \in V$ the characteristic function $\chi_E(v_1, v_2)$ is uniquely determined by*

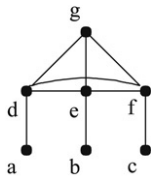

Fig. 4. Example of a P_4-sparse graph (constructed by $\circledast(\leftrightarrow(d, a, e, b, f, c), g)$).

(a) $\mathrm{lca}(v_1, v_2)$, *in the case of cographs,*
(b) $\mathrm{lca}(v_1, v_2)$, $\delta_1(v_1, v_2)$, $|v_1| \bmod 2$, $|v_2| \bmod 2$ *and the information whether* $|v_1| < |v_2|$, *in the case of* P_4*-sparse graphs,*
(c) $\mathrm{lca}(v_1, v_2)$, $\delta_4(v_1, v_2)$, $c(v_1)$, $|v_1| \bmod 2$, $|v_2| \bmod 2$ *and the information whether* $|v_1| < |v_2|$, *in the case of* P_4*-extendible graphs.*

Proof. For cographs, the claim is obvious.

P_4-sparse graphs: Jamison and Olariu have defined an operation ② such that any P_4-sparse graph can be obtained from single vertex graphs with the operations $+$, $\cup$ and ②. Consider disjoint graphs $G_1 = (V_1, \emptyset)$ and $G_2 = (V_2, E_2)$ with $V_2 = \{v\} \cup K \cup R$ such that all vertices in K are adjacent to each other, $|K| = |V_1| + 1 \geq 2$, every vertex in R is adjacent to all vertices in K and nonadjacent to v, and there exists a vertex v' in K such that v is either adjacent only to v' or adjacent to all vertices in K except for v'. Note, that G_1 and G_2 are P_4-sparse graphs. Choose a bijection $f : V_1 \to K \setminus \{v'\}$ and define $G_1 ② G_2 = \left(V_1 \cup V_2, E_2 \cup E'\right)$, where

$$E' = \begin{cases} \{\{v_1, f(v_1)\} \mid v_1 \in V_1\}, & \text{if } \{v, v'\} \in E_2 \\ \{\{v_1, v_2\} \mid v_1 \in V_1, v_2 \in K \setminus \{f(v_1)\}\}, & \text{otherwise.} \end{cases}$$

For our purposes, we need to substitute the operation ② by three operations that help us obtaining the same graphs as when using ②. Now consider an arbitrary P_4-sparse graph $G = G_1 ② G_2$, where G_1 and G_2 are as above. Clearly, $G_{|R}$ and $G_{|V'}$, where $V' = K \cup V_1 \cup \{v\}$, are P_4-sparse, too, and may thus be obtained by a sequence of $\cup$, $+$ and ② operations. We now define the mentioned three operations, $\leftrightarrow$, $\nleftrightarrow$ and $\circledast$ such that

$$G = G_{|V'} \circledast G_{|R} \tag{1}$$

and $G_{|V'}$ is obtained from applying either the operation $\leftrightarrow$ or the operation $\nleftrightarrow$ on all the single vertex graphs $(\{v\}, \emptyset)$ with $v \in V'$.

In the following, if we use a joining operation on a single vertex v, then we mean that the joining operation is on the single vertex graph consisting only of v. Consider two disjoint vertex sets $V = \{v_1, \ldots, v_k\}$ and $U = \{u_1, \ldots, u_k\}$ with $k \geq 2$. Then $\leftrightarrow (v_1, u_1, \ldots, v_k, u_k)$ yields the graph $G(V \cup U, E_{\leftrightarrow})$, where $E_{\leftrightarrow}$ contains the edges $\{v_i, u_i\}$ and $\{v_i, v_j\}$ for $1 \leq i, j \leq k$ and $i \neq j$. The operation $\nleftrightarrow (v_1, u_1, \ldots, v_k, u_k)$ yields the graph $G(V \cup U, E_{\nleftrightarrow})$, where $E_{\nleftrightarrow}$ contains the edges $\{v_i, u_j\}$ and $\{v_i, v_j\}$ for $1 \leq i, j \leq k$ and $i \neq j$. Now consider a graph $G' = (V', E')$, $V' = V \cup U$, obtained by one of the operations $\leftrightarrow$ or $\nleftrightarrow$ on $(v_1, u_1, \ldots, v_k, u_k)$ and let $G^* = (V^*, E^*)$ be another arbitrary P_4-sparse graph. The operation $\circledast$ is defined as:

$$G' \circledast G^* := \left(V \cup U \cup V^*,\ E' \cup E^* \cup \left\{\{v, v^*\} \mid v \in V, v^* \in V^*\right\}\right).$$

It is easy to see that (1) is in fact true. An example of a P_4-sparse graph obtained by these operations is given in Fig. 4.

Now consider a P_4-sparse graph G and the corresponding tree representation $T(G)$, where the inner nodes are labeled with the operations in $\Omega = \{\cup, +, \nleftrightarrow, \leftrightarrow, \circledast\}$. Let v_1, v_2 be two vertices of G and let $p = \mathrm{lca}(v_1, v_2)$. If p is labeled with $+$ or $\cup$, then the adjacency of v_1 and v_2 is already uniquely determined. Hence, assume that p is labeled with one of the operations in $\{\nleftrightarrow, \leftrightarrow, \circledast\}$. Recall that we choose the labels of the vertices in such a way that a preorder traversal of $T(G)$ traverses the leafs v in an order with an increasing number $|v|$.

If p is labeled $\leftrightarrow$ or $\nleftrightarrow$, then clearly $c(v)$ is odd if and only if v is in V (if the operation $\leftrightarrow$ or $\nleftrightarrow$ is carried out as described above on vertices from V and U). Hence, two vertices v_1, v_2, $|v_1| < |v_2|$, which are both children of a node p labeled with $\leftrightarrow$ in $T(G)$ are adjacent if and only if either $|v_2| - |v_1| = 1$ and $c(v_1)$ is odd or if $c(v_1)$ and $c(v_2)$ are odd. On the other hand if p is labeled with $\nleftrightarrow$, then v_1 and v_2 are adjacent if either $|v_2| - |v_1| \neq 1$ and $c(v_1)$ is odd or if $c(v_2)$ is odd. Now assume that $p = \mathrm{lca}(v_1, v_2)$ is a node labeled with $\circledast$. Then the left child of p is labeled with

$\nleftrightarrow$ or $\leftrightarrow$ and using the assumption $|v_1| < |v_2|$ we know that v_1 is a child of the left child of p. Hence, v_1 and v_2 are adjacent if and only if v_1 is in V, i.e. if $c(v_1)$ is odd.

To conclude, assuming $|v_1| < |v_2|$, it suffices to determine $\mathrm{lca}(v_1, v_2)$, $\delta_1(v_1, v_2)$ as well as $c(v_1) \bmod 2$ and in the case that p is not labeled with $\circledast$ also $c(v_2) \bmod 2$. But note that if we know $p = \mathrm{lca}(v_1, v_2)$ and p is labeled with either $\leftrightarrow$, $\nleftrightarrow$ or $\circledast$, then the parent of v_1 is uniquely determined. Furthermore, if p is labeled with $\leftrightarrow$ or $\circledast$ also the parent of v_2 is uniquely determined. And if the parent of a vertex v is uniquely determined then $c(v) \bmod 2$ is uniquely determined by $|v| \bmod 2$ (it is either $|v| \bmod 2$ or $(|v|+1) \bmod 2$, depending on whether the first child of the parent has an odd or an even vertex coding). Thus, the adjacency of v_1 and v_2 is uniquely determined by $\mathrm{lca}(v_1, v_2)$, $\delta_1(v_1, v_2)$, $|v_1| \bmod 2$, $|v_2| \bmod 2$ and the information whether $|v_1| < |v_2|$.

P_4-extendible graphs: Similar to P_4-sparse graphs Jamison and Olariu defined additional operations sufficient to construct P_4-extendible graphs. Jamison and Olariu defined an operation named ② in [11] which can lead to eight different graphs (this ②-operation is different from the ②-operation for P_4-sparse graphs). We substitute this operation with eight operations, each of them joining the single vertex graphs with vertices in $V = \{a, b, c, d, e\}$ to one of the eight possible graphs obtained by the ②-operation. Let

$$
\begin{aligned}
&\overset{us1}{\multimap}(a,b,c,d) = (\{a,b,c,d\}, \{\{a,b\},\{b,d\},\{c,d\}\}),\\
&\overset{us2}{\multimap}(a,b,c,d,e) = (V,\ \{\{a,b\},\{a,c\},\{c,d\},\{a,e\},\{d,e\}\}),\\
&\overset{us3}{\multimap}(a,b,c,d,e) = (V,\ \{\{a,b\},\{a,c\},\{c,d\},\{a,e\},\{c,e\},\{d,e\}\}),\\
&\overset{us4}{\multimap}(a,b,c,d,e) = (V,\ \{\{a,b\},\{b,d\},\{c,d\},\{d,e\}\}),\\
&\overset{us5}{\multimap}(a,b,c,d,e) = (V,\ \{\{a,b\},\{b,d\},\{c,d\},\{c,e\},\{d,e\}\}),\\
&\overset{\overline{us}1}{\multimap}(a,b,c,d,e) = (V,\ \{\{a,b\},\{b,c\},\{c,d\},\{a,e\},\{c,e\},\{d,e\}\}),\\
&\overset{\overline{us}2}{\multimap}(a,b,c,d,e) = (V,\ \{\{a,b\},\{b,c\},\{c,d\},\{d,e\}\}),\\
&\overset{\overline{us}3}{\multimap}(a,b,c,d,e) = (V,\ \{\{a,b\},\{b,c\},\{c,d\},\{a,e\},\{d,e\}\}).
\end{aligned}
$$

In addition to their ②-operation they also defined an operation, which we denote as *wing-addition*: Consider a graph $G_1 = (V_1, E_1)$ which is one of the five graphs obtained by one of the $\overset{us}{\multimap}$ operations and an arbitrary disjoint P_4-extendible graph $G_2 = (V_2, E_2)$. The wing addition $\curlyvee$ is defined as $G_1 \curlyvee G_2 := (V_1 \cup V_2, E_1 \cup E_2 \cup E')$, where

$$E' := \{\{v_1, v_2\} \mid v_1 \in V_1 \text{ is inner node of an induced } P_4,\ v_2 \in V_2\}.$$

Now consider a P_4-extendible graph G with a tree representation $T(G)$ where each joining operation is either $\cup$, $+$, $\curlyvee$ or one of the $\overset{us}{\multimap}$ or $\overset{\overline{us}}{\multimap}$ operations. Let v_1, v_2 be two vertices of G and $p = \mathrm{lca}(v_1, v_2)$. If p is labeled with $\cup$ or $+$, then the adjacency of v_1 and v_2 is clear. Now assume that the label of p is one of the $\overset{\overline{us}}{\multimap}$ or $\overset{us}{\multimap}$ operations. Then obviously, the adjacency is uniquely determined by $c(v_1)$ and $c(v_2)$ which is uniquely determined by $\delta_4(v_1, v_2)$ and the relation ($<$ or $\geq$) between $|v_1|$ and $|v_2|$. Finally consider the case that the label of p is $\curlyvee$ and assume w.l.o.g. that $|v_1| < |v_2|$. Then v_1 is a child of a vertex labeled with an $\overset{us}{\multimap}$ operation. Hence, v_1 and v_2 are adjacent if and only if v_1 is an inner node. But due to the ordering of the children of such a vertex it is easy to see that v_1 is an inner node if and only if $c(v_1) \bmod 2 = 1$ in the case that there are three inner vertices and $c(v_1) \bmod 2 = 0$ in the case that there are two inner vertices. But since we know the parent of v_1, $c(v_1) \bmod 2$ can be determined by $|v_1| \bmod 2$. Hence, the adjacency of v_1 and v_2 is uniquely determined by p, $|v_1| \bmod 2$, $|v_2| \bmod 2$ and the information whether $|v_1| < |v_2|$ in the case that p is labeled with $\curlyvee$. □

Any OBDD can be viewed as a non-uniform algorithm which queries all variables of a function in an order determined by the variable ordering, and then outputs the function value. The state space of the algorithm after each variable query corresponds to the set of nodes in the OBDD labeled with the variable to be queried next. Hence, in order to upper bound the OBDD size for a graph we can devise a corresponding algorithm which decides adjacency with a small state space. The previous lemma shows that for cographs such an algorithm merely needs to compute the lca of two leaves in the tree representation of that cograph.

The following algorithm determines the lowest common ancestor (lca) of two leaves in a tree representation. The idea of the algorithm is to search the lowest common ancestor starting from the leaf corresponding to v_1 and ascending successively while reading the vertex coding of v_2.

Algorithm 1. The algorithm is defined for a fixed tree T with N leaves labeled with values from $\{0,1\}^n, n = \lceil \log N \rceil$, in such a way that if $v_0, \ldots, v_{N-1}$ are the leaves found in a preorder traversal then $|v_i| = i$. Note, that we do not need such a labeling for the inner nodes. We only need to store them in such a way, that they are uniquely identifiable. The inputs of the algorithm are $x, y \in \{0,1\}^n$ and the output is the lca of x and y if x and y are both leaves in the tree. If either x or y is not a leaf of T, then the output is $-\infty$. The algorithm queries all input variables once in the order $x_{n-1}, \ldots, x_0, y_{n-1}, \ldots, y_0$ and after each query two values b and c are stored. The value of c is one of the relations "$<$", "$>$" or "$=$" and b is a node in T.

We describe two invariants which are true after each step of the algorithm unless the algorithm terminated. Consider a situation in which all variables up to y_i, $0 \leq i \leq n-1$, have been queried and the algorithm has not terminated. The first invariant is that c is the relation between $|x_{n-1} \ldots x_i|$ and $|y_{n-1} \ldots y_i|$ (e.g., if $c =$ " $<$ ", then $|x_{n-1} \ldots x_i| < |y_{n-1} \ldots y_i|$). Now assume $c =$ "$<$" and let y_i^0 be the leaf $y_{n-1} \ldots y_i 0 \ldots 0$. The second invariant is that $b = \mathrm{lca}(x, y_i^0)$ and $y_i^1 = y_{n-1} \ldots y_i 1 \ldots 1$ is not in the subtree rooted at b. For the case $c =$ "$>$", the invariant is analogous, but the roles of y_i^0 and y_i^1 are exchanged. In the case $c =$ "$=$", we have $b = x$.

Note that if these invariants are true, then by knowing c and b, the value of $y_{n-1} \ldots y_i$ is uniquely determined. For $c =$ "$=$" this is obvious. For $c =$ "$<$", this follows from the enumeration of the leaves: In the subtree rooted at the right child of b, there can only be one leaf $a_{n-1} \ldots a_i 0 \ldots 0$ such that $a_{n-1} \ldots a_i 1 \ldots 1$ is not in this subtree. The case $c =$ "$>$" is analogous. Hence, it suffices to describe an algorithm for which these invariants remain true after each query of a y-variable and which – under the assumption that the invariants remain true – outputs the correct result.

Step 1: Store "$=$" in c. Query all x-variables and let $b = x$. If x is not a leaf, output $-\infty$. Clearly, the invariants remain true after this step unless the algorithm terminates.

Step 2: Query the next y-variable, say y_i. Since the invariants were true before querying y_i, by knowing b and c we now know $y_{n-1} \ldots y_i$. If $c =$ "$=$" and $y_i = b_i$, we can proceed with querying the next variable because the invariants remain true. Hence, we continue with Step 1, again. If $c =$ "$=$" and $y_i \neq b_i$, then we have found the most significant bit in which x and y differ. We can change the value of c to "$<$" or "$>$" such that it reflects the relation between $|x_{n-1} \ldots x_i|$ and $|y_{n-1} \ldots y_i|$.

Once we reach this point it holds $c \neq$ "$=$". Let b' be $\mathrm{lca}(b, y_i^0)$ $(= \mathrm{lca}(x, y_i^0))$ in the case $c =$ "$<$" and $b' = \mathrm{lca}(b, y_i^1)$ $(= \mathrm{lca}(x, y_i^1))$ in the case $c =$ "$>$". Since we know $y_{n-1} \ldots y_i$ and b, b' is uniquely determined, if it exists. However, it may happen that such a b' does not exist. In this case y cannot be a leaf of the tree and thus we output $-\infty$.

Assume that $c =$ "$<$" (the case $c =$ "$>$" is analogous with the roles of y_i^1 and y_i^0 exchanged). If the leaf y_i^1 is not in the subtree rooted at b', then we replace b with b'. Clearly, the invariants are now true again and we proceed with the next y-variable by going to Step 1. If on the other hand, y_i^1 is in the subtree rooted at b', then obviously all leaves $y_{n-1} \ldots y_i a_{i-1} \ldots a_0$ for $a_{i-1} \ldots a_0 \in \{0,1\}^i$ are in this subtree. Hence, b' is the lca of x and y and we output b'.

Note that after querying the last y-variable, the algorithm terminates in Step 1, because either an appropriate b' is not found (and the algorithm outputs $-\infty$) or the found b' is in fact the lca of x and y.

In the following we consider π-OBDDs with more than two possible output values. Such an OBDD has for each possible output z a sink marked with z. The semantics is defined analogously to that of binary OBDDs: For an input $a = (a_1, \ldots, a_n) \in \{0,1\}^n$, the function value $f(a)$ of the function f represented by the OBDD equals z if and only if the computation path of a reaches the sink labeled z.

Algorithm 1 in fact defines such a π-OBDD, where $(\pi(1), \ldots, \pi(2n)) = (x_{n-1}, \ldots, x_0, y_{n-1}, \ldots, y_0)$: In the ith step the variable $\pi(i)$ is queried and after the query the algorithm stores a state value q_i which depends only on the previous stored state value and the result of the variable query. Each possible stored state value q_i of the algorithm corresponds to a node labeled with the variable $\pi(i+1)$ and thus the sum of the number of possible state values q_i over all $0 \leq i \leq 2n$ is the number of OBDD nodes (q_0 is the unique starting state corresponding to the root of the OBDD and the two possible final state values $q_{2n} \in \{0,1\}$ corresponding to the sinks of the OBDD). It is obvious how to construct the π-OBDD corresponding to such an algorithm. The following lemma bounds the number of possible states between which the algorithm has to distinguish and thus the OBDD size.

Lemma 6. *Let T be a tree as in Algorithm 1 and let π be the variable ordering with $(\pi(1), \ldots, \pi(2n)) = (x_{n-1}, \ldots, x_0, y_{n-1}, \ldots, y_0)$. There is a π-*OBDD *computing for $x, y \in \{0,1\}^n$ the lca of x and y, if it exists, and $-\infty$, otherwise. The* OBDD *has at most $2N-2$ x-nodes and at most $3N\lceil \log N\rceil - 2N$ y-nodes.*

Proof. For Step 1 it suffices to store $x_{n-1} \ldots x_i$ when these variables have been queried. Moreover, if it turns out that $|x_{n-1} \ldots x_i 0 \ldots 0|$ is at least N, the algorithm can output $-\infty$. Hence, it is easy to see that the π-OBDD has at most as many nodes as a complete binary tree in the first n levels and that the $(n+1)$th level has at most N nodes. Hence, there are at most $2^n - 1 < 2N - 1$ x-nodes and at most N y_{n-1}-nodes.

Now consider Step 2. As long as b is a leaf, the value of c is "=" and once b is not a leaf anymore, the value of c is either "<" or ">". Hence, by knowing c, we can conclude on whether b is a leaf or not. Since there are N leaves and at most $N-1$ inner nodes, $3N$ states suffice for storing c and b. Therefore, there are at most $3N$ y_i-nodes for $0 \le i < n-1$. To conclude, the total number of y-nodes of the OBDD is bounded by $N + (n-1) \cdot 3N = 3N \cdot \lceil \log N\rceil - 2N$. □

Now assume that we want to devise an OBDD for a cograph $G = (V, E)$ with N vertices. We use the tree representation $T(G)$, where each inner node is either labeled with $+$ or $\cup$. We label each leaf of $T(G)$ corresponding to a vertex v with v. Recall that in order to be able to apply Lemma 6, we have to label the vertices in such a way that if in a preorder traversal of the tree $T(G)$ the leaves appear in the order $v_0, \ldots, v_{N-1}$, then $|v_i| = i$, where $V = \{v_0, \ldots, v_{N-1}\}$. Now the above lemma yields an OBDD computing the lca of the corresponding leaves of two vertices v_1, v_2 in V. We join all sinks labeled with $+$ of this OBDD together to the 1-sink, and all sinks labeled $\cup$ as well as the sink labeled with $-\infty$ together to the 0-sink. This way we obtain an OBDD for χ_E. This already shows that the OBDD size of cographs is in fact at most $3N\lceil \log N\rceil$ (we have to add a value of 2 for the two sinks to the total number of inner nodes obtained in Lemma 6).

Theorem 7. *The* OBDD *size of cographs is at most $3N\lceil \log N\rceil$.*

In order to obtain OBDDs for P_4-sparse and P_4-extendible graphs, our Algorithm 1 has to be modified in such a way that it computes in addition to the lca of two leaves the other information which is needed in order to decide adjacency between the vertices v_1 and v_2, as described in Lemma 5.

Let $x = x_{n-1} \ldots x_0 = v_1$ and $y = y_{n-1} \ldots y_0 = v_2$. First note that in order to obtain the information, whether $|v_1| < |v_2|$, no additional OBDD nodes are necessary, because in our analysis of Algorithm 1 we have already taken into account the variable c, which stores this information once the algorithm terminates. The values $|v_1| \bmod 2 = x_0$ and $|v_2| \bmod 2 = y_0$ can also be obtained very easily: Once our algorithm has queried x_0, it stores its value until it terminates. Since y_0 is the variable queried last, there is nothing to be stored. However, we have to take care that if the algorithm terminates before all variables have been queried, variable y_0 must now be tested additionally. To conclude, if we have an OBDD computing lca (v_1, v_2), then by first doubling the number of y-nodes and then inserting at most two y_0-nodes, we obtain an OBDD which additionally computes $|v_1| \bmod 2$ and $|v_2| \bmod 2$. Hence, such an OBDD has at most $2N + \mathcal{O}(1)$ x-nodes and $6N\lceil \log N\rceil - 4N + \mathcal{O}(1)$ y-nodes.

The additional computation of $c(v_1)$ is also very easy, because right after Step 1, x is known completely. Hence, we can store $c(v_1)$ from there on. This increases the number of y-nodes by a factor of $c_{\max}$, where $c_{\max}$ is the maximum number of leaves a node in $T(G)$ may have as children. Note that $c_{\max} = 5$ in the case of P_4-extendible graphs, but for P_4-sparse graph it may be as large as N.

It is a little bit more complicated to compute $\delta_d(v_1, v_2)$. Recall that $\delta_d(v_1, v_2) = ||x| - |y||$ if this value is at most d and is 0 otherwise. Let $\ell = \lceil \log(d+1)\rceil$. Consider the situation in which in Step 2 of Algorithm 1 for the first time a variable $y_i \neq b_i$ is found (if this does not occur then $\delta_d(v_1, v_2) = 0$). At this time, we still have $b = x$ and thus $x = x_{n-1} \ldots x_0$ is completely known. Assume first that $y_i > x_i$ (i.e. $y_i = 1$ and $x_i = 0$). Then $\delta_d(v_1, v_2) \neq 0$ only, if $x_{i-1} = \cdots = x_\ell = 1$ and $y_{i-1} = \cdots = y_\ell = 0$. If this is the case, then $|y| - |x|$ is uniquely determined by the least significant ℓ bits of x and y. Hence, the algorithm checks the corresponding x_j-variables, $\ell \le j < i$, as well as the y-variables once they are queried; if they are not all 1 or all 0, respectively, then it stores "$\delta_d = 0$". Additionally, it stores $x_{\ell-1} \ldots x_0$ during its execution as well as $y_{\ell-1} \ldots y_1$, once they have been queried. The other case, $y_i < x_i$ works analogously, because then $\delta_d(v_1, v_2) \neq 0$ only if $x_{i-1} = \cdots = x_\ell = 0$ and $y_{i-1} = \cdots = y_\ell = 1$. Note that in order to distinguish between these two cases, no additional storage is necessary, because the relation of x_i and y_i is uniquely determined by c, the relation between $|x|$ and $|y|$. Hence, it suffices to store additionally $2^\ell + 1$

Fig. 5. Example of an unit interval graph and its interval representation.

possible states (one for the case "$\delta_d = 0$") during the phase in which y_j-variables with $j \geq \ell - 1$ are queried and additionally $2^{2\ell} + 1$ possible states in the remainder of the algorithm. This means that the number of y_j-nodes in an OBDD computing the lca has to be multiplied by a factor of at most $2^\ell + 1$ for $j \geq \ell - 1$ and by a factor of at most $2^{2\ell} + 1$ for $j < \ell - 1$. But since it is well known that a minimal OBDD has on the last k levels, k a constant, only a constant number of nodes, it suffices to multiply the number of y-nodes by $2^{\lceil \log(d+1) \rceil} + 1$ and add an $\mathcal{O}(1)$ term on top of that.

Applying these modifications on the OBDD obtained in Lemma 6 directly yields OBDDs for P_4-sparse graphs and for P_4-extendible graphs that are larger only by a constant factor.

Theorem 8. *The* OBDD *size of P_4-sparse graphs and P_4-extendible graphs is bounded by $\mathcal{O}(N \log N)$.*

We contrast the above results by a lower bound for cographs, which also applies to its superclasses of P_4-reducible, P_4-extendible and P_4-sparse graphs. This result shows that our upper bounds are optimal up to a factor of $\mathcal{O}(\log^2 N)$.

Theorem 9. *The worst-case* OBDD *size of cographs is at least* $1.832 \cdot N / \log N - \mathcal{O}(1)$.

Proof. As cographs can be constructed from single vertex graphs by consecutive application of join and union operations, their number is equal to the number of two-terminal series-parallel networks. We use an asymptotic formula for this number by Finch [7]. It says that the number of graphs in the class of cographs $\mathcal{C}$ satisfies $N_{\mathcal{C}}(N) \sim \lambda \kappa^N N^{-\frac{3}{2}}$ for the constants $\lambda = 0.4127\ldots$ and $\kappa = 3.5608\ldots$. As $\kappa^N = 2^{N \cdot 1.8322\ldots}$, we gain the result directly from Corollary 4. □

4. Interval graphs

An interval graph is defined by a set of closed intervals $I \subseteq \mathbb{R}^2$, each of them corresponding to a vertex in the graph. Two vertices are adjacent if and only if the corresponding two intervals intersect.

4.1. Unit interval graphs

We first analyse *unit interval graphs*, i.e. interval graphs where the underlying intervals have unit length (see Fig. 5 for an example).

Therefore, we can identify the intervals with just the left endpoint. We assume w.l.o.g. that no two intervals have the same endpoints and label the vertices in such a way that if the interval represented by a vertex v_1 starts further left than the interval represented by a vertex v_2, then $|v_1| < |v_2|$.

In order to proof an upper bound we use the characterization of minimal OBDDs due to Sieling and Wegener [21]. We say that a function *essentially depends* on a variable x_i, if the substitution $x_i = 0$ leads to a different *subfunction* than the substitution $x_i = 1$.

Theorem 10 (*[21]*). *Let $X_n = \{x_1, \ldots, x_n\}$ and let $f \in B_n$ be a function depending on x_i, $1 \leq i \leq n$, and π be a variable ordering on X_n. Further, let S_i, $1 \leq i \leq n$, be the set of non-constant subfunctions of f resulting from setting all variables $\pi(j)$, $j < i$ to constants, restricted to those, that essentially depend on $\pi(i)$. Then the minimal π-OBDD for f has $|S_i|$ nodes labeled with the variable $\pi(i)$.*

Let $G = (V, E)$ be a unit interval graph labeled as described above. For $x \in \{0, 1\}^n$ let the interval corresponding to the vertex x be denoted by $I(x) = [a, a + 1]$, where $a \in \mathbb{R}$. Recall that the vertices are labeled in such a way that for $|x| < |y|$ the starting point of the interval $I(x)$ is less than the starting point of $I(y)$.

Let π be the variable ordering where $(\pi(1), \ldots, \pi(2n)) = (x_{n-1}, y_{n-1}, \ldots, x_0, y_0)$. Further, let $f = \chi_E$ and $s_{k,\ell}$, $1 \leq k \leq n$ with $\ell \in \{k-1, k\}$, be the number of non-constant subfunctions $f_{|\alpha,\beta}$ of f, where α is an assignment to the

variables $x_{n-1}, \ldots, x_{n-k}$ and β is an assignment to the variables $y_{n-1}, \ldots, y_{n-\ell}$. Additionally, we set $s_{0,0} := 1$. Then, for $0 \leq k < n$, $s_{k,k}$ is an upper bound on the number of x_{n-k-1}-nodes and $s_{k+1,k}$ is an upper bound on the number of y_{n-k-1}-nodes as stated in Section 1.2. For the sake of simplicity we assume $k = \ell$ using the simple observation that $s_{k+1,k}$ is at most $2s_{k,k}$ and denote $s_{k,k}$ by s_k.

The following upper bound, which we will use if k is large, is true for all Boolean functions. Since there are 2^{2^m} Boolean functions in m variables, we have

$$s_k \leq 2^{2^{2n-2k}}. \tag{2}$$

If k is small, we need a better bound. Therefore, we derive an upper bound for the number of non-constant subfunctions $f_{|\alpha,\beta}$, where α and β are assignments to the variables $x_{n-1} \ldots x_{n-k}$ and $y_{n-1} \ldots y_{n-k}$, respectively, and $|\alpha| \leq |\beta|$. Then s_k is at most twice the result. Let $(\alpha_1, \beta_1), \ldots, (\alpha_p, \beta_p)$ be all different pairs of assignments to the variables $x_{n-1} \ldots x_{n-k}$ and $y_{n-1} \ldots y_{n-k}$ such that $|\alpha_i| \leq |\beta_i|$ and $f_{|\alpha_i,\beta_i}$ is not constant for all $1 \leq i \leq p$. Furthermore, assume that $(|\alpha_1|, |\beta_1|), \ldots, (|\alpha_p|, |\beta_p|)$ are ordered lexicographically. We prove below that

$$\forall\, 1 \leq i \leq p: \quad |\beta_i| \leq |\beta_{i+1}|. \tag{3}$$

Then, using the fact that $(|\alpha_1|, |\beta_1|), \ldots, (|\alpha_p|, |\beta_p|)$ are ordered lexicographically, it is easy to see that the number of these pairs, p, is bounded by $|\alpha_p| + |\beta_p| + 1$. Hence, we obtain $p \leq 2^{k+1} - 1$ and thus $s_k \leq 2^{k+2} - 2$. Using the upper bound of (2) it follows that $s_k \leq \min\{2^{k+2} - 2,\ 2^{2^{2n-2k}}\}$. Summing over all $0 \leq k < n$ and applying Theorem 10 yields the following upper bound for the OBDD size of the unit interval graph G:

$$\sum_{k=0}^{n-1}(s_k + 2s_k) + 2 \leq 3 \sum_{k=0}^{n-\lfloor(\log n)/2\rfloor} 2^{k+2} + 3 \sum_{k=n-\lfloor(\log n)/2\rfloor+1}^{n-1} 2^{2^{2n-2k}} = \mathcal{O}(N/\sqrt{\log N}).$$

This leads to the following result.

Theorem 11. *The OBDD size of unit interval graphs with N vertices is bounded above by $\mathcal{O}(N/\sqrt{\log N})$.*

Proof. It remains to prove claim (3). If $|\alpha_i| = |\alpha_{i+1}|$, this follows right away from the lexicographical ordering of the pairs $(|\alpha_j|, |\beta_j|)$. Hence, assume $|\alpha_i| < |\alpha_{i+1}|$. If (3) is not true, i.e. $|\beta_{i+1}| < |\beta_i|$, then we have

$$|\alpha_i| < |\alpha_{i+1}| \leq |\beta_{i+1}| < |\beta_i|$$

(recall that we only count the pairs (α_j, β_j) where $|\alpha_j| \leq |\beta_j|$). Since $f_{|\alpha_i,\beta_i}$ is not the constant 0-function, there is an assignment c to the remaining x-variables $x_{n-k-1}, \ldots, x_0$ and an assignment d to the remaining y-variables $y_{n-k-1}, \ldots, y_0$ such that $f_{|\alpha_i,\beta_i}(c, d) = 1$. Hence, $\chi_E(\alpha_i c, \beta_i d) = 1$ and thus the intervals $I(\alpha_i c)$ and $I(\beta_i d)$ intersect. Now consider additional arbitrary assignments c' to the remaining x-variables and d' to the remaining y-variables. Obviously, then $|\alpha_i c| < |\alpha_{i+1} c'| < |\beta_i d|$ and $|\alpha_i c| < |\beta_{i+1} d'| < |\beta_i d|$. Hence, the intervals $I(\alpha_{i+1} c')$ and $I(\beta_{i+1} d')$ are neither right of $I(\beta_i d)$ nor left of $I(\alpha_i c)$. But since the latter intervals intersect, obviously $I(\alpha_{i+1} c')$ and $I(\beta_{i+1} d')$ intersect, too. Because this is true for all c' and d' we obtain that $f_{|\alpha_{i+1},\beta_{i+1}} = 1$, which contradicts the assumption that this subfunction is not constant. This completes the proof of (3). □

4.2. General interval graphs

We now consider the more general class of arbitrary interval graphs. Here, we have to care about both start and end points of the intervals and the upper bound technique for unit interval graphs does not work anymore. However, using a similar idea yields the upper bound stated in the following theorem.

Theorem 12. *Interval graphs with N vertices have OBDDs of size $\mathcal{O}(N^{3/2}/\log^{3/4} N)$.*

Proof. We can assume w.l.o.g. that no two intervals have the same endpoints. It is clear that we can label the vertices for our representation in such a way, that the most significant different bit of v_1 and v_2 for two vertices v_1, v_2 indicates the relation of the start points of the intervals representing v_1 and v_2, if the index of the most significant different bit is odd, and the relation of the end points otherwise (the value 0 for this bit should indicate that start and end point are to the left of the start and end point of the other interval, respectively). We just have to separate the vertices iteratively in

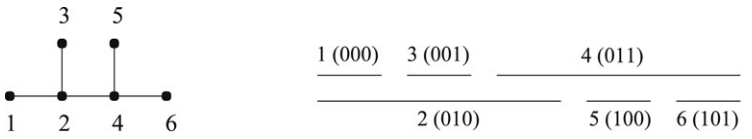

Fig. 6. Example of an interval graph and its labeled interval representation.

two halves alternately distinguishing between start points and end points to attain such a labeling. It is clear, that we can label our vertices in such a way, that $V = \{[0]_n, \ldots, [N-1]_n\}$ where N is the number of vertices of the graph that is to be represented and n is $\lceil \log N \rceil$ (see Fig. 6 for an example).

Let $G = (V, E)$ be an interval graph labeled as described above. For $x \in \{0,1\}^n$ let the interval corresponding to the vertex x be denoted by $I(x) = [a, b]$, where $a, b \in \mathbb{R}$.

Let π be the variable ordering where $(\pi(1), \ldots, \pi(2n)) = (x_{n-1}, y_{n-1}, \ldots, x_0, y_0)$. Further, let $f := \chi_E$ and let s_k be defined as in the proof of Theorem 11.

For large k we also need the upper bound (2): $s_k \leq 2^{2^{2n-2k}}$. If k is small we devise an upper bound for the number of non-constant subfunctions $f_{|\alpha,\beta}$ with $|\alpha| \leq |\beta|$. Then s_k is at most twice the result. Let α^e denote the substring of α which consists only of the bits with even index in α. Similarly, α^o is the substring of α consisting of the bits with odd index. We define β^e and β^o analogously. Further, let $(\beta_1^o, \alpha_1^e, \alpha_1^o, \beta_1^e), \ldots, (\beta_p^o, \alpha_p^e, \alpha_p^o, \beta_p^e)$ be all different quadruples of assignments to the variables $x_{n-1} \ldots x_{n-k}$ and $y_{n-1} \ldots y_{n-k}$ such that $|\alpha_i| \leq |\beta_i|$ and $f_{|\alpha_i,\beta_i}$ is not constant for all $1 \leq i \leq p$. Furthermore, assume that $(|\beta_1^o|, |\alpha_1^e|, |\alpha_1^o|, |\beta_1^e|), \ldots, (|\beta_p^o|, |\alpha_p^e|, |\alpha_p^o|, |\beta_p^e|)$ are ordered lexicographically. We prove below that

$$\forall\, 1 \leq i < j \leq p: \quad \left(\left(|\alpha_i^o| = |\alpha_j^o|\right) \wedge \left(|\beta_i^e| = |\beta_j^e|\right)\right) \Rightarrow \left(|\alpha_i^e| \leq |\alpha_j^e|\right). \tag{4}$$

Then, using the fact that $(|\beta_1^o|, |\alpha_1^e|, |\alpha_1^o|, |\beta_1^e|), \ldots, (|\beta_p^o|, |\alpha_p^e|, |\alpha_p^o|, |\beta_p^e|)$ are ordered lexicographically, it is easy to see that the number of these pairs, p, is bounded by $(|\alpha_p^e| + |\beta_p^o| + 1)(|\alpha_p^o| + 1)(|\beta_p^e| + 1)$. Hence, we obtain $p \leq \frac{3}{2}\sqrt{2} \cdot 2^{\frac{3}{2}k} - 2^k$ and thus $s_k \leq 3\sqrt{2} \cdot 2^{\frac{3}{2}k} - 2^{k+1}$. Using the upper bound of (2) it follows that

$$s_k \leq \min\{3\sqrt{2} \cdot 2^{\frac{3}{2}k} - 2^{k+1}, 2^{2^{2n-2k}}\}.$$

Summing over all $0 \leq k < n$ and applying Theorem 10 yields the following upper bound for the OBDD size of the interval graph G:

$$\begin{aligned} 3\sum_{k=0}^{n-1} s_k + 2 &\leq 3 \sum_{k=0}^{n-\left\lfloor \frac{2\log n-1}{4} \right\rfloor} \left(3\sqrt{2} \cdot 2^{\frac{3}{2}k} - 2^{k+1}\right) + 3 \sum_{k=n-\left\lfloor \frac{2\log n-1}{4} \right\rfloor+1}^{n-1} 2^{2^{2n-2k}} \\ &\leq 16 \cdot 2^{\frac{3}{2}n - \frac{3}{4}\log n} - 10 \cdot 2^{n - \frac{1}{2}\log n} + \left(\frac{3}{2}\log n - \frac{3}{4}\right) \cdot 2^{2^{-2,5}n} + 2 \\ &= \mathcal{O}(N^{\frac{3}{2}}/(\log N)^{\frac{3}{4}}). \end{aligned}$$

It remains to prove the claim (4). If $|\beta_i^o| = |\beta_j^o|$, this follows right away from the lexicographical ordering of the quadruples $\left(|\beta_k^o|, |\alpha_k^e|, |\alpha_k^o|, |\beta_k^e|\right)$. Hence, assume $|\beta_i^o| < |\beta_j^o|$. We assume that (4) is not true, i.e. $|\alpha_i^e| > |\alpha_j^e|$, $|\alpha_i^o| = |\alpha_j^o|$ and $|\beta_i^e| = |\beta_j^e|$. Since $f_{|\alpha_i,\beta_i}$ is not the constant 1-function, there is an assignment c to the remaining x-variables $x_{n-k-1}, \ldots, x_0$ and an assignment d to the remaining y-variables $y_{n-k-1}, \ldots, y_0$ such that $f_{|\alpha_i,\beta_i}(c, d) = 0$. Since the graph is undirected, we have $f(x, y) = f(y, x)$ for all x and y and thus we may assume w.l.o.g. that $|c| < |d|$ in the case that $|\alpha_i| = |\beta_i|$. We obtain $\chi_E(\alpha_i c, \beta_i d) = 0$ and thus the intervals $I(\alpha_i c)$ and $I(\beta_i d)$ do not intersect and $I(\alpha_i c)$ is left of $I(\beta_i d)$ (recall that we assumed $|\alpha_i| \leq |\beta_i|$ and have $|c| < |d|$ if $|\alpha_i| = |\beta_i|$). Now consider additional arbitrary assignments c' to the remaining x-variables and d' to the remaining y-variables. Obviously, then $|\beta_i^o c| < |\beta_j^o c'|$ and $|\alpha_i^e d| > |\alpha_j^e d'|$ and as $|\alpha_i^o| = |\alpha_j^o|$ and $|\beta_i^e| = |\beta_j^e|$, the interval $I(\alpha_j d')$ ends earlier than $I(\alpha_i d)$ and $I(\beta_j c')$ starts later than $I(\beta_i c)$. But since $I(\alpha_i d)$ and $I(\beta_i c)$ do not intersect, obviously $I(\alpha_j c')$ and $I(\beta_j d')$ do not intersect either. Because this is true for all c' and d' we obtain that $f_{|\alpha_j,\beta_j} = 0$, which contradicts the assumption that this subfunction is not constant. This completes the proof of (4). □

4.3. Lower bounds

We finally state a lower bound for unit interval graphs and general interval graphs obtained by counting arguments.

Theorem 13. *For all* $\varepsilon > 0$*, the worst-case* OBDD *size of unit interval graphs with* N *vertices is at least* $(2-\varepsilon)N/\log N - \mathcal{O}(1)$ *and the worst-case* OBDD *size of interval graphs with* N *vertices is at least* $(1-\varepsilon)N - \mathcal{O}(1)$.

Proof. Finch [6] provided asymptotic formula for the size of the class of unit interval graphs $\mathcal{U}$, $N_{\mathcal{U}}(N) \sim \frac{1}{8e^{\kappa}\sqrt{\pi}} \frac{4^N}{N^{\frac{3}{2}}}$, and Gavoille and Paul [8] obtained asymptotic formula for the size of the class of general interval graphs $\mathcal{I}$: $N_{\mathcal{I}}(N) \geq 2^{N\log N - \mathcal{O}(N)}$. The lower bounds follow directly from Corollary 4. □

5. Bipartite graphs

The goal of this section is to show for some specific graph class that graph representation is not necessarily more space efficient with OBDDs than with adjacency matrices.

Theorem 14. *For all* $\varepsilon > 0$*, the worst-case* OBDD *size of bipartite graphs with* N *vertices is at least* $(\frac{1}{8} - \varepsilon)N^2/\log N - \mathcal{O}(1)$.

Proof. To apply counting arguments, we consider the class of labeled 2-*coloured graphs*, where different colourings of 2-*colourable (bipartite) graphs* lead to different graphs. The asymptotic relation between the size of the class of labeled 2-coloured graphs $\mathcal{C}_\ell$ and the class of labeled 2-colourable graphs $\mathcal{B}_\ell$ has been proven by Prömel and Steger in [16] to be as follows: $\lim_{N\to\infty} N_{\mathcal{C}_\ell}(N)/N_{\mathcal{B}_\ell}(N) = 2$. Therefore, a random bipartite graphs has almost surely only one 2-colouring (and the inverse 2-colouring). Asymptotics for the number of labeled 2-coloured graphs were given by Wright [24,17] as $N_{\mathcal{C}_\ell}(N) \sim \kappa 2^{\frac{N^2}{4}} 2^N \sqrt{\left(\frac{2}{N\ln 2}\right)}$ where $\kappa = 1 \pm 0.0000013097\ldots$ is a constant. According to Prömel [15] the relation between the number of labeled and unlabeled bipartite graphs is bounded by $N!$. Combining all these results leads to the following relation for the size of the class of unlabeled bipartite graphs $\mathcal{B}$: $\lim_{N\to\infty} 2(N!)N_{\mathcal{B}}(N)/2^{\frac{N^2}{4}} 2^N \sqrt{\left(\frac{2}{N\ln 2}\right)} \leq 1$. The lower bound now follows directly from Corollary 4. □

The disadvantage of proving lower bounds with counting arguments is that they only show the existence of graphs which are hard to represent. However, such graphs might for large N never appear in applications because e.g. they are not computable in polynomial time. A statement showing how to construct such a graph or at least telling us that such a graph is computable in polynomial time has much more relevance. In order to achieve such results, we show how any Boolean function can be represented by a bipartite graph. This way, we can conclude from known lower bounds for the OBDD size of Boolean functions on lower bounds for the OBDD size of the corresponding bipartite graphs.

Definition 15. Let $f \in B_n$, n even, be a Boolean function. The bipartite graph $G_f = (V_1 \cup V_2, E)$ is given by

$$V_1 := \left\{ v_1 \in \{0,1\}^{\frac{n}{2}+1} \mid |v_1| < 2^{\frac{n}{2}} \right\}$$
$$V_2 := \left\{ v_2 \in \{0,1\}^{\frac{n}{2}+1} \mid 2^{\frac{n}{2}} \leq |v_2| < 2^{\frac{n}{2}+1} \right\}$$
$$E := \left\{ \{v_1, v_2\} \mid v_1 \in V_1, v_2 \in V_2, f\left([|v_1|]_{\frac{n}{2}} \left[|v_2| - 2^{\frac{n}{2}} \right]_{\frac{n}{2}} \right) = 1 \right\}.$$

Fig. 7 shows an example of a bipartite graph for a function.

Theorem 16. *For each function* $f \in B_n$ *there is a bipartite graph* $G_f = (V, E)$ *such that the OBDD size of* χ_E *is not smaller than the OBDD size of* f.

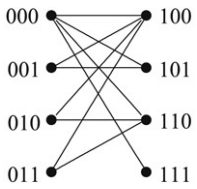

Fig. 7. Example of a bipartite graph for $f = \overline{x_1 x_3} \vee \overline{x_1 x_2} x_3 \vee x_1 \overline{x_4}$.

Proof. Let us assume that an OBDD B with smaller size exists for χ_E. Let $\{x_0, \ldots, x_{\frac{n}{2}}, y_0, \ldots, y_{\frac{n}{2}}\}$ be the set of variables of χ_E, then

$$f\left(x_1, \ldots, x_{\frac{n}{2}}, y_1, \ldots, y_{\frac{n}{2}}\right) = \chi_E\left(0, x_1, \ldots, x_{\frac{n}{2}}, 1, y_1, \ldots, y_{\frac{n}{2}}\right)$$

follows from Definition 15. We therefore can construct an OBDD for f from B by redirecting all edges leading to a node labeled with x_0 or y_0 to the appropriate 0-successor or 1-successor of this node, respectively. We represent f with this OBDD of smaller size, which is a contradiction. □

Andreev, Baskakov, Clementi and Rolim [1] presented a Boolean function which is computable in polynomial time and has an OBDD size of $2^{n-\mathcal{O}(\log^2 n)}$. According to the knowledge of the authors this is the best known lower bound for the OBDD size of a function in P.

Corollary 17. *There is a bipartite graph G_f, $f \in B_{2k}$, with $N = 2^{k+1}$ vertices which is computable in polynomial time and for which the* OBDD *size of χ_E is at least $N^2/(\log N)^{\mathcal{O}(\log\log N)}$.*

Acknowledgements

The authors thank Ingo Wegener and two anonymous reviewers for helpful comments on the paper. The second author was supported in part by DFG grants We 1066/10-1 and Wo 1232/1-1.

References

[1] A. Andreev, J. Baskakov, A. Clementi, J. Rolim, Small pseudo-random sets yield hard functions: New tight explict lower bounds for branching programs, in: 26th ICALP, in: LNCS, vol. 1644, 1999, pp. 179–189.
[2] Y. Breitbart, H. Hunt III, D. Rosenkrantz, On the size of binary decision diagrams representing boolean functions, Theoret. Comput. Sci. 145 (1995) 45–69.
[3] R.E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Trans. Comput. C-35 (8) (1986) 677–691.
[4] J.R. Burch, E.M. Clarke, K.L. McMillan, D.J. Dill, L.J. Hwang, Symbolic model checking: 10^{30} states and beyond, Inform. Comput. 98 (1992) 142–170.
[5] D.G. Corneil, H. Lerchs, L. Stewart Burlingham, Complement reducible graphs, Discrete Appl. Math. 3 (1981) 163–174.
[6] S.R. Finch, Mathematical Constants, Cambridge University Press, 2003.
[7] S.R. Finch, Series-parallel networks, Supplementary material for [6], 2003.
[8] C. Gavoille, C. Paul, Optimal distance labeling schemes for interval and circular-arc graphs, in: G. Di Battista, U. Zwick (Eds.), 11th ESA, in: LNCS, vol. 2832, Springer, 2003.
[9] G.D. Hachtel, F. Somenzi, A symbolic algorithm for maximum flow in 0-1 networks, Formal Methods Syst. Des. 10 (1997) 207–219.
[10] B. Jamison, S. Olariu, p_4-reducible graphs — a class of uniquely tree representable graphs, Stud. Appl. Math. 81 (1989) 79–87.
[11] B. Jamison, S. Olariu, On a unique tree representation for p_4-extendible graphs, Discrete Appl. Math. 34 (1991) 151–164.
[12] B. Jamison, S. Olariu, A tree representation for p_4-sparse graphs, Discrete Appl. Math. 35 (1992) 115–129.
[13] S. Kannan, M. Naor, S. Rudich, Implicit representation of graphs, in: STOC '88, ACM Press, 1988, pp. 334–343.
[14] K. Meer, D. Rautenbach, On the OBDD size for graphs of bounded tree- and clique-width, in: IWPEC '06, in: LNCS, vol. 4169, Springer, 2006.
[15] H.J. Prömel, Counting unlabeled structures, J. Combin. Theory Ser. A 44 (1987) 83–93.
[16] H.J. Prömel, A. Steger, Random l-colorable graphs, Random Structures Algorithms 6 (1995) 21–37.
[17] R.C. Read, E.M. Wright, Coloured graphs: A correction and extension, Canad. J. Math. 22 (1970) 594–596.
[18] D. Sawitzki, Implicit flow maximization by iterative squaring, in: P. Van Emde Boas, J. Pokorny, M. Bielikova, J. Stuller (Eds.), 30th SOFSEM, in: LNCS, vol. 2932, Springer, 2004, pp. 301–313.
[19] D. Sawitzki, A symbolic approach to the all-pairs shortest-paths problem, in: 30th WG, in: LNCS, vol. 3353, 2004, pp. 154–167.
[20] D. Sieling, The nonapproximability of OBDD minimization, Inform. Comput. 172 (2002) 103–138.
[21] D. Sieling, I. Wegener, NC-algorithms for operations on binary decision diagrams, Parallel Processing Lett. 3 (1) (1993) 3–12.
[22] I. Wegener, Branching Programs and Binary Decision Diagrams, SIAM, 2000.
[23] P. Woelfel, Symbolic topological sorting with OBDDs, in: 28th MFCS, 2003, pp. 671–680.
[24] E.M. Wright, Counting coloured graphs, Canad. J. Math. 13 (1961) 683–693.