Representation of Graphs by OBDDs

Robin Nunkesser[1] and Philipp Woelfel[2,⋆]

[1] FB Informatik, LS2, Universität Dortmund, 44221 Dortmund, Germany
robin.nunkesser@udo.edu
[2] University of Toronto, 10 King's College Road, Toronto M5S 3G4, Canada
pwoelfel@cs.toronto.edu

Abstract. 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.

1 Introduction

Some modern applications require such huge graphs that the usual, explicit representation by adjacency lists or adjacency matrices is infeasible. E.g., 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. Such huge graphs appear also if the basic graph as e.g. the street network of a city is interlinked with other components as e.g. traffic amount and time slots.

In order to be able to store huge graphs, implicit representations of graphs can be used. In the standard implicit representation the vertices of a graph are labeled in such a way that adjacency of two vertices is uniquely determined by their 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). However, such an implicit representation is not a generic graph representation because by each representation only a small number of graphs can be described (e.g. most graphs are not interval graphs). Hence, for different graph classes different implicit representations and different algorithms are needed.

Another approach is to use a generic graph representation (i.e. a representation which can represent *all* graphs) which requires less space for sufficiently structured graphs than for unstructured graphs. One idea is to store the characteristic function of the vertex and the edge set by a generic data structure for boolean functions. A data structure which is well suited for this task is the *Ordered Binary Decision Diagram* (OBDD), because for all important operations on boolean functions (e.g. the synthesis operation, substitution by constants or

⋆ Supported in part by DFG grants We 1066/10-1 and Wo 1232/1-1.

X. Deng and D. Du (Eds.): ISAAC 2005, LNCS 3827, pp. 1132–1142, 2005.

satisfiability test) efficient algorithms are known. Such OBDD operations allow to efficiently convert any graph represented by e.g. an adjacency list or an adjacency matrix to a corresponding OBDD representation. Moreover, it has turned out that several graph problems can be solved with a polylogarithmic number of OBDD operations, if the input graphs are represented by OBDDs. Recently, several OBDD algorithms for fundamental graph problems have been devised, as e.g. for network flow maximization [8, 15], topological sorting [19] or for finding shortest paths in networks [16].

Since an OBDD representation is a generic representation of graphs, most graphs cannot be represented in less space by OBDDs than by adjacency matrices or adjacency lists. But one hopes (and we show in this paper) that for sufficiently structured inputs as they may appear in practical applications, a good compression can be achieved. Experimental studies have already shown that in many practical relevant cases OBDD representations of graphs may be advantageous. However, it has not yet been theoretically investigated for which general graph classes this is the case. While it is obvious that very simply structured graphs as e.g. grid networks have a small OBDD representation, this is not clear for more complicated graph classes. In this paper, we start filling this gap by investigating several important graph classes with respect to their space requirements in an OBDD representation.

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 representation type for boolean functions.

Definition 1. Let $X_n = \{x_1, \ldots, x_n\}$ be a set of boolean variables. 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. 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, the other by 1. If an edge leads from a node labeled by x_i to a node labeled by x_j, then $\pi^{-1}(x_i) < \pi^{-1}(x_j)$.

A π-OBDD is said to *represent* a boolean function $f \in B_n$, if for any $a = (a_1, \ldots, a_n) \in \{0,1\}^n$, the path starting at the root and leading from any x_i node over the edge labeled by the value of a_i, ends at a sink with label $f(a)$. The *size* of a π-OBDD G is the number of its nodes and is denoted by $|G|$. The π-OBDD *size* of a boolean function f (π-OBDD(f)) is the size of the minimum π-OBDD computing f. The OBDD size of a boolean function f (OBDD(f)) is the size of the minimum π-OBDD computing f for some variable ordering π.

Let $G = (V, E)$ be a graph. We can use an OBDD for representing the graph G by letting it represent the characteristic function $\chi_E : E \to \{0,1\}$ of the edge set, where $\chi_E(v_1, v_2) = 1 \Leftrightarrow \{v_1, v_2\} \in E$. 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 the integer $k \geq 0$, i.e. the string $z_{n-1} \ldots z_0 \in \{0,1\}^n$ with $k = |z|$. In order to encode the vertices by boolean variables, we use the convention that $V = \{[0]_n, \ldots, [N-1]_n\}$, where

$N \geq 0$ and $n = \lceil \log N \rceil$. Thus, we can store V using n bits or by a π-OBDD for the characteristic function χ_V of V, and the minimal π-OBDD for χ_V has a size of $\mathcal{O}(n \log n) = \mathcal{O}(\log N \log \log N)$ for all variable orderings π.

Definition 2. The π-OBDD *size* of a graph $G = (V, E)$ (π-OBDD(G)) is the size of the minimum π-OBDD computing χ_E for some labeling of the vertices. Analogously, the OBDD *size* of 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 $G \in \mathcal{G}$.

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 $n \cdot |f^{-1}(1)|$. This yields the following straight forward proposition.

Proposition 1. *Let $G = (V, E)$ be a graph with $N := |V|$ and $M := |E|$. The* OBDD *size of G is bounded above by*

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

Note that if a graph G with N edges and M vertices 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 $\mathcal{O}\left(|B|\,(\log |B| + \log n)\right)$ 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 at most a constant factor larger than the representation by adjacency matrices. Since $\mathcal{O}\left(M \log N(\log M + \log \log N)\right)$ 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 $\mathcal{O}\left((N + M) \log N\right)$ 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 1 covers only the worst case. In the following we will show that several very natural and large graph classes have OBDD sizes which yield a much better space behaviour than that of explicit representations. We start in Sect. 2 with the investigation of several types of graph classes which do not

contain many P_4s, most prominently cographs. We show that the worst-case OBDD size of these graphs is between $\Omega(N/\log N)$ and $\mathcal{O}(N \log N)$. In Sect. 3 we investigate interval graphs. We show that the worst-case OBDD size of unit interval graphs is between $\Omega(N/\log N)$ and $\mathcal{O}(N/\sqrt{\log N})$, while that of general interval graphs is between $\Omega(N)$ and $\mathcal{O}(N^{3/2}/\log^{3/4} N)$. Finally, in Sect. 4 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 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 [4] 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* [9], P_4-*extendible graphs* [10] and P_4-*sparse graphs* [11]. 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 P_4 p that induces a different P_4 together with three vertices from p and 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.

All these graph classes have in common that they can be constructed from single vertex graphs by graph operations which join 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 $\cup$ and $+$ operations.

Now consider a set Ω of operations, each of them joining several vertex disjoint graphs together. 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).

In order to obtain upper bounds for the OBDD-size of graph classes which have a tree representation, we devise an algorithm which has the following property: In each step of the algorithm a variable is queried (in the order determined by the variable ordering π). 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 previ-

ous stored state value and the result of the variable query. Each possible stored state value q_i of the algorithm corresponds to an OBDD node labeled with the variable $\pi(i+1)$ and thus the sum of the number of possible state values q_i over all $0 \leq i \leq n$ 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 $q_{n+1} \in \{0,1\}$ corresponding to the sinks of the OBDD). It is obvious how to construct the π-OBDD corresponding to such an algorithm.

A tree representation of a graph is helpful if we want to devise an algorithm deciding adjacency which can then be turned into an OBDD for the graph. E.g. 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 $+$. 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, 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.

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 1. *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*

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

The proof of part (a) follows right away from the definition of cographs. The proofs of parts (b) and (c) are omitted due to space restrictions and can be found in the full version of the paper.

The following algorithm determines the lowest common ancestor (lca) of two nodes in a tree representation. The idea of the algorithm is to search the lca starting in 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 $|v_i| = i$. 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 $|x_{n-1} \dots x_i|$ and $|y_{n-1} \dots y_i|$ (e.g., if c="<", then $|x_{n-1} \dots x_i| < |y_{n-1} \dots y_i|$). Now assume $c =$ "<" and let y_i^0 be the leaf $y_{n-1} \dots y_i 0 \dots 0$. The second invariant is that $b = \mathrm{lca}(x, y_i^0)$ and $y_i^1 = y_{n-1} \dots y_i 1 \dots 1$ 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 $y_{n-1} \dots y_i$ is uniquely determined. For $c =$ "=" this is obvious. For $c =$ "<", this follows because due to the enumeration of the leaves, in the right subtree of the tree rooted at b, there can only be one leaf $a_{n-1} \dots a_i 0 \dots 0$ such that $a_{n-1} \dots a_i 1 \dots 1$ 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 invariant remains true – outputs the correct result.

Step 1: Store "=" in c. Query all x variables and let b be the corresponding leaf of T. If there is no corresponding 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 $y_{n-1} \dots y_i$. 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 2, 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 $|x_{n-1} \dots x_i|$ and $|y_{n-1} \dots y_i|$. Hence, if we reach this point in any case $c \neq$ "=". 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 $y_{n-1} \dots y_i$ 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 $c =$ "<" (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 2. If on the other hand, y_i^1 is in the subtree rooted at b', then obviously all leaves $y_{n-1} \dots y_i a_{i-1} \dots a_0$ for $a_{i-1} \dots a_0 \in \{0,1\}^i$ 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 2, 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.

Theorem 1. *The* OBDD *size of a cograph with N vertices is at most* $3N \lceil \log N \rceil$.

Proof. Let $V = \{v_0, \ldots, v_{N-1}\}$ and let $G = (V, E)$ be a cograph. Let $T(G)$ be the tree representation of G as described before Lemma 1. We have shown above that Algorithm 1 computes for two vertices $u, v \in V$ (given by n-bit strings x, y) the corresponding $\mathrm{lca}(u, v)$ in $T(G)$ by querying each x- and y-variable at most once in the order $x_{n-1}, \ldots, x_0, y_{n-1}, \ldots, y_0$. According to Lemma 1 the adjacency of u and v in G is uniquely determined by their lca, and thus the algorithm describes a π-OBDD for the variable ordering π with $(\pi(1), \ldots, \pi(2n)) = (x_{n-1}, \ldots, x_0, y_{n-1}, \ldots, y_0)$. In the following we bound the number of possible states the algorithm has to store after each variable query.

For Step 1 it suffices to store $x_{n-1} \ldots x_i$ once theses variables have been queried. If it turns out that $|x_{n-1} \ldots x_i 0 \ldots 0|$ is at least N, the algorithm outputs $-\infty$. Then the π-OBDD has at most as many nodes as a complete binary tree in the first n levels and 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 no 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 $0 \leq i < n - 1$. To conclude, the total number of y nodes of the OBDD is bounded by $N + (n-1) \cdot 3N = 3N \cdot \lceil \log N \rceil - 2N$. □

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 1. The necessary modifications of the algorithm can be found in the full version of the paper.

Theorem 2. *The* OBDD *size of P_4-sparse graphs and P_4-extendible graphs is* $\mathcal{O}(N \log N)$.

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.

Theorem 3. *The worst-case* OBDD *size of cographs is at least* $1.832 \cdot N / \log N - \mathcal{O}(1)$.

We prove this with counting arguments. Let in the following $N_{\mathcal{G}}(N)$ denote the number of graphs with N vertices in a graph class $\mathcal{G}$. Note that unless stated otherwise the graphs in the considered graph classes $\mathcal{G}$ are unlabeled.

Proposition 2. *Consider functions $s_N : \mathbb{N} \to \mathbb{R}$ for $N \in \mathbb{N}$ and let $\mathcal{G}$ be a graph class that allows the addition of isolated vertices. If $\lim_{N\to\infty}(2^{s_N \log s_N + s_N \log\log N + \mathcal{O}(s_N)} \cdot (N_{\mathcal{G}}(N))^{-1}) < 1$, then for large enough N there are graphs with N or more vertices in $\mathcal{G}$ such that the* OBDD *size of these graphs is more than s_N.*

Proof. Wegener has shown in [18] 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 $f \in B_n$. It is easy to see that if the limit in the claim 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. □

An asymptotic formula due to Finch [6] states 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 $\kappa = 3.5608\ldots$. Since $\kappa^N = 2^{N \cdot 1.8322\ldots}$, Theorem 3 follows directly from Proposition 2.

3 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.

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

Theorem 4. *The* OBDD *size of unit interval graphs with N vertices is bounded above by $\mathcal{O}(N/\sqrt{\log N})$.*

In order to proof this result we use the characterization of minimal OBDDs due to Sieling and Wegener [17]: The minimal π-OBDD representing a function f on $x_1, \ldots, x_n$ has as many nodes labeled with the variable x_i, $1 \leq i \leq n$, as there are different subfunctions of f essentially depending on x_i when all variables x_j with $\pi^{-1}(x_j) < \pi^{-1}(x_i)$ are set to constants (a subfunction essentially depends on a variable x_i, if the substitution $x_i = 0$ leads to a different subfunction than the substitution $x_i = 1$).

Proof (of Theorem 4). 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 $a \in \mathbb{R}$. 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 $s_{k,\ell}$, $1 \leq k < n$, $\ell \in \{k-1, k\}$, 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 $y_{n-1}, \ldots, y_{n-\ell}$. Then $s_{k,k}$ is an upper bound on the number of x_k nodes and $s_{k,k-1}$ is an upper bound on the number of y_k nodes as stated in Sect 1. For the sake of simplicity we assume $k = \ell$ using the simple observation that $s_{k,k+1}$ is at most $2s_{k,k}$ and denote $s_{k,k}$ by s_k.

Since there are 2^{2^m} boolean functions in m variables, we have

$$s_k \;\leq\; 2^{2^{2n-2k}} \quad . \tag{1}$$

If k is small, we need a better bound. We derive an upper bound for the number of non-constant subfunctions $f_{|\alpha,\beta}$, where α and β are assignments to the variables $x_n \ldots x_{n-k}$ and $y_n \ldots y_{n-k}$, 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 different pairs of assignments to the x and y variables such that $|\alpha_i| \leq |\beta_i|$ and $f_{|\alpha_i,\beta_i} \notin \{0,1\}$ (i.e. these

subfunctions are 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

$$\forall\, 1 \leq i \leq p: \quad |\beta_i| \leq |\beta_{i+1}| \ . \tag{2}$$

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 (1) it follows that $s_k \leq \min\{2^{k+2} - 2,\, 2^{2^{2n-2k}}\}$. We plug this inequality into the upper bound $2 + \sum_{k=0}^{n-1}(s_k + 2s_k)$ on the OBDD size of G, and using simple algebra obtain an upper bound of $\mathcal{O}(N/\sqrt{\log N})$.

It remains to prove the claim (2). 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 (2) 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 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. □

Using a similar idea yields an upper bound for general interval graphs stated in the following theorem (the proof can be found in the full version of the paper).

Theorem 5. *Interval graphs with N vertices have* OBDD*s of size $\mathcal{O}(N^{3/2}/\log^{3/4} N)$.*

Finch [5] provided an 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 [7] obtained an asymptotic formula for the size of the class of general interval graphs $\mathcal{I}$: $N_{\mathcal{I}}(N) \geq 2^{N \log N - \mathcal{O}(N)}$. Thus, the following lower bounds follow directly from Proposition 2.

Theorem 6. *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 $(1-\varepsilon)N - \mathcal{O}(1)$.*

4 Bipartite Graphs

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

Theorem 7. *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. 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 [13] 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 [20, 14] 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 [12] 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 Proposition 2. □

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 function on lower bounds for the OBDD size of the corresponding bipartite graphs.

Definition 3. 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 the vertex sets $V_1 := \{v_1 \in \{0,1\}^{\frac{n}{2}} \mid |v_1| < 2^{\frac{n}{2}}\}$ and $V_2 := \{v_2 \in \{0,1\}^{\frac{n}{2}+1} \mid 2^{\frac{n}{2}} \leq |v_2| < 2^{\frac{n}{2}+1}\}$ and the edge set $E := \{\{v_1, v_2\} | v_1 \in V_1, v_2 \in V_2, f(v_1[|v_2| - 2^{\frac{n}{2}}]_{\frac{n}{2}}) = 1\}$.

Theorem 8. *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.*

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 3. 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 1. *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 $N^2/(\log N)^{\mathcal{O}(\log\log N)}$.*

Acknowledgment. The authors thank Ingo Wegener for helpful comments on the paper.

References

1. A. Andreev, J. Baskakov, A. Clementi, and J. Rolim. Small pseudo-random sets yield hard functions: New tight explicit lower bounds for branching programs. In *26th ICALP*, volume 1644 of *LNCS*, pp. 179–189. 1999.
2. Y. Breitbart, H. Hunt III, and D. Rosenkrantz. On the size of binary decision diagrams representing boolean functions. *Theor. Comp. Sci.*, 145:45–69, 1995.
3. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, C-35:677–691, 1986.
4. D. G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs. *Discrete Appl. Math.*, 3:163–174, 1981.
5. S. R. Finch. *Mathematical Constants*. Cambridge University Press, 2003.
6. S. R. Finch. Series-parallel networks, 2003. Supplementary Material for [5].
7. C. Gavoille and C. Paul. Optimal distance labeling schemes for interval and circular-arc graphs. In G. Di Battista and U. Zwick, editors, *11th ESA*, volume 2832 of *LNCS*. Springer, 2003.
8. G. D. Hachtel and F. Somenzi. A symbolic algorithm for maximum flow in 0-1 networks. *Formal Methods in System Design*, 10:207–219, 1997.
9. B. Jamison and S. Olariu. p_4-reducible graphs - a class of uniquely tree representable graphs. *Stud. Appl. Math.*, 81:79–87, 1989.
10. B. Jamison and S. Olariu. On a unique tree representation for p_4-extendible graphs. *Discrete Appl. Math.*, 34:151–164, 1991.
11. B. Jamison and S. Olariu. A tree representation for p_4-sparse graphs. *Discrete Appl. Math.*, 35:115–129, 1992.
12. H. J. Prömel. Counting unlabeled structures. *J. Combin. Theory Ser. A*, 44:83–93, 1987.
13. H. J. Prömel and A. Steger. Random l-colorable graphs. *Random Structures Algorithms*, 6:21–37, 1995.
14. R. C. Read and E. M. Wright. Coloured graphs: A correction and extension. *Canad. J. Math.*, 22:594–596, 1970.
15. D. Sawitzki. Implicit flow maximization by iterative squaring. In P. Van Emde Boas, J. Pokorny, M. Bielikova, and J. Stuller, editors, *30th SOFSEM*, volume 2932 of *LNCS*, pp. 301–313. Springer, 2004.
16. D. Sawitzki. A symbolic approach to the all-pairs shortest-paths problem. In *30th WG*, volume 3353 of *LNCS*, pp. 154 – 116. 2004.
17. D. Sieling and I. Wegener. NC-algorithms for operations on binary decision diagrams. *Parallel Processing Letters*, 3:3–12, 1993.
18. I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.
19. P. Woelfel. Symbolic topological sorting with OBDDs. *Journal of Discrete Algorithms*, to appear.
20. E. M. Wright. Counting coloured graphs. *Canad. J. Math.*, 13:683–693, 1961.