OBDD-Based Representation of Interval Graphs

Marc Gillé*

TU Dortmund, LS2 Informatik, Germany

Abstract. A graph $G = (V, E)$ can be described by the characteristic function of the edge set χ_E which maps a pair of binary encoded nodes to 1 iff the nodes are adjacent. Using *Ordered Binary Decision Diagrams* (OBDDs) to store χ_E can lead to a (hopefully) compact representation. Given the OBDD as an input, symbolic/implicit OBDD-based graph algorithms can solve optimization problems by mainly using functional operations, e. g., quantification or binary synthesis. While the OBDD representation size can not be small in general, it can be provable small for special graph classes and then also lead to fast algorithms. In this paper, we show that the OBDD size of unit interval graphs is $O(|V|/\log|V|)$ and the OBDD size of interval graphs is $O(|V|\log|V|)$ which both improve a known result from Nunkesser and Woelfel (2009). Furthermore, we can show that using our variable order and node labeling for interval graphs the worst-case OBDD size is $\Omega(|V|\log|V|)$. We use the structure of the adjacency matrices to prove these bounds. This method may be of independent interest and can be applied to other graph classes. We also develop a maximum matching algorithm on unit interval graphs using $O(\log|V|)$ operations and evaluate the algorithm empirically.

1 Introduction

The development of graph algorithms is a classic and intensively studied area of computer science. But the requirements on graph algorithms have changed by the emergence of massive graphs, e. g., the internet graph or social networks. There are applications, e. g., dealing with a state transition graphs in circuit verification, where even polynomial running time may not be feasible or the input does not fit into the main memory. In order to deal with such massive graphs, *implicit* graph algorithms have been investigated, where the input is represented by the characteristic function χ_E of the edge set and the nodes are encoded by binary numbers. Implicit representations can be significantly smaller than explicit representations on structured graphs. χ_E can be represented by *Ordered Binary Decision Diagrams* (OBDDs) introduced by Bryant [8] which are a commonly used data structure for Boolean functions since they support many important functional operations efficiently. A research area came up concerning the design and analysis of implicit (graph) algorithms on OBDD represented inputs ([12,13,16,25,26,27,30]). Implicit algorithms are successful in many practical applications, e. g., model checking [9], integer linear programming [18] and logic

* Supported by Deutsche Forschungsgemeinschaft, grant BO 2755/1-1.

A. Brandstädt, K. Jansen, and R. Reischuk (Eds.): WG 2013, LNCS 8165, pp. 286–297, 2013.

minimization [11]. One of the first implicit graph algorithm was the maximum flow algorithm on 0-1-networks presented by Hachtel and Somenzi [16] which was able to solve instances up to 10^{36} edges and 10^{27} nodes in reasonable time.

The number of functional operations of an implicit algorithm is an important measure of difficulty [2]. Algorithms using a polylogarithmic number of operations were designed for instance for topological sorting [30], maximal matching [6] and minimum spanning tree [3] where a matching M, i. e., a set of edges without a common vertex, is called maximal if M is no proper subset of another matching. The actual running time depends on the OBDD sizes which are used for these operations which are hard to determine in general. So the practical performance is often evaluated experimentally, e. g., for the maximum matching problem in bipartite graphs [4] or for the maximum flow problem [16,25].

For a good running time of an implicit algorithm the size of the OBDD representing the input graph should be small. Nunkesser and Woelfel [22] investigated the OBDD size of restricted graph classes such as interval graphs. An interval graph is an intersection graph of intervals on the real line, i. e., two intervals (nodes) are adjacent iff they have a nonempty intersection. If the intervals have a length of 1, then the graph is called unit interval graph. (Unit) Interval graphs were extensively studied and have many applications, e. g., in genetics, archaeology, scheduling, and much more [15]. Nunkesser and Woelfel [22] proved that general interval graphs with N nodes can be represented by OBDDs of size $O(N^{3/2}\log^{3/4} N)$ while the OBDD size of unit interval graphs is $O(N/\sqrt{\log N})$. They also proved a lower bound of $\Omega(N)$ for general interval graphs and $\Omega(N/\log N)$ for unit interval graphs which means that the worst-case OBDD size of a graph from these classes is bounded below by these values.

As in [22], we use $n = \lceil \log N \rceil$ bits, i. e., the minimal number of bits, to encode the nodes of a graph. Since the worst-case OBDD size is exponentially large in the number of input bits, using χ_E in an implicit algorithm motivates to use a minimal amount of input bits to avoid a large worst-case OBDD size. Aiming for a good compression of χ_E, Meer and Rautenbach [19] investigated graphs with bounded clique-width or tree-width and increases the number of bits used for the node labeling to $c \cdot \log N$ with constant c and were able to improve for instance the OBDD size of cographs from $O(N \log N)$ [22] to $O(N)$.

Our Contribution. In Section 3 we present a new method to show upper and lower bounds of the size of an OBDD representing a graph. Using some known structure of the adjacency matrix of interval graphs [21], we improve the bound on general interval graphs to $O(N \log N)$ while using a more convenient way to label the nodes than in [22]. Using a probabilistic argument, we prove a lower bound of $\Omega(N \log N)$ if we use the same labeling and variable order as for our upper bound. We can also close the gap of the upper bound and the lower bound in the case of unit interval graphs and show that the OBDD size is $\Theta(N/\log N)$.

In Section 4 we present a maximum matching algorithm for unit interval graphs using only $O(\log N)$ functional operations. We were able to compute the transitive closure of a unit interval graph using only $O(\log N)$ operations instead of $O(\log^2 N)$ operations, which are needed in general. To the best of the

author's knowledge, this is the first time that the labeling of nodes is used to speed up an implicit algorithm for a large graph class as unit interval graphs and to improve the number of functional operations. In order to implement the algorithm efficiently, we have to extend a known result due to Woelfel [30] to a different variable order for constructing OBDDs representing multivariate threshold functions. In Section 5 we evaluate the implicit matching algorithm experimentally and see that it is both very fast and space efficient.

A simple implicit representation of an interval graph is a list of N intervals using $\Theta(\log N)$ bits for each endpoint summing up to $\Theta(N \log N)$ space. Our results show that in the worst case the OBDD representation is almost as good as the interval representation with the advantage that it is possible to use $o(N \log N)$ space for some instances. Together with our implicit algorithm, this shows that the representation of at least unit interval graphs with OBDDs enables a good compression without loosing the usability in algorithms.

2 Preliminaries

Omitted proofs and a more detailed version of this paper can be found in [14].

OBDDs. We denote the set of Boolean functions $f : \{0,1\}^n \to \{0,1\}$ by B_n. Let $(x_0, \ldots, x_{n-1}) = x \in \{0,1\}^n$ be a binary number of length n and $|x| := \sum_{i=0}^{n-1} x_i \cdot 2^i$ the value of x. Further, let $l \in \mathbb{N}$ be a natural number then we denote by $[l]_2$ the corresponding binary number of l, i. e., $|[l]_2| = l$. Let $G = (V, E)$ be a directed graph with node set $V = \{v_0, \ldots, v_{N-1}\}$ and edge set $E \subseteq V \times V$. Here, an undirected graph is interpreted as a directed symmetric graph. Implicit algorithms are working on the characteristic function $\chi_E \in B_{2n}$ of E where $n = \lceil \log N \rceil$ is the number of bits needed to encode a node of V and $\chi_E(x, y) = 1$ if and only if $(v_{|x|}, v_{|y|}) \in E$. In order to deal with Boolean functions, OBDDs were introduced by Bryant [8] to get a compact representation, which supports a bunch of functional operations efficiently.

Definition 1 (Ordered Binary Decision Diagram (OBDD)).

Order. *A* variable order π *on the input variables* $X = \{x_0, \ldots, x_{n-1}\}$ *of a Boolean function* $f \in B_n$ *is a permutation of the index set* $I = \{0, \ldots, n-1\}$.

Representation. *A* π-OBDD *is a directed, acyclic and rooted graph* G *with two sinks labeled by the constants* 0 *and* 1. *Each inner node is labeled by an input variable from* X *and has exactly two outgoing edges labeled by* 0 *and* 1. *Each edge* (x_i, x_j) *has to respect the variable order* π, *i. e.,* $\pi(i) < \pi(j)$.

Evaluation. *An assignment* $a \in \{0,1\}^n$ *of the variables defines a path from the root to a sink by leaving each* x_i*-node via the* a_i*-edge. A* π*-OBDD* G_f *represents* f *iff for every* $a \in \{0,1\}^n$ *the defined path ends in a sink with label* $f(a)$.

Complexity. *The* size *of a* π*-OBDD* G, *denoted by* $|G|$, *is the number of nodes in* G. *The* π*-OBDD size of a function* f *is the minimum size of a* π*-OBDD representing* f. *The OBDD size of* f *is the minimum* π*-OBDD size over all variable orders* π. *The* width *of* G *is the maximum number of nodes labeled by the same input variable.*

In the following we describe some important operations on Boolean functions which we will use in this paper (see, e. g., Section 3.3 in [29] for a detailed list). Let f and g be Boolean functions in B_n on the variable set $X = \{x_0, \ldots, x_{n-1}\}$ and G_f and G_g be OBDDs representing f and g which are also the inputs for the operations. The *negation* $\overline{f} \in B_n$ of f can be computed in time $O(1)$, i. e., given the OBDD G_f it is possible to compute the OBDD $G_{\overline{f}}$ in $O(1)$ time. Let $i \in \{0, \ldots, n-1\}$ be an index and a $c_i \in \{0, 1\}$. The *replacement by constant*, i. e., the subfunction $f_{|x_i=c_i}$ can be computed in time $O(|G_f|)$. Let $\otimes \in B_2$ be a binary Boolean operation. The *synthesis* of f and g w.r.t. $\otimes$, i. e., the function $h \in B_n$ with $h := f \otimes g$, can be computed in time $O(|G_f| \cdot |G_g|)$. Finally, the *quantification* $h := Qx_i : f$ of f with quantifier $Q \in \{\exists, \forall\}$ is defined by $\exists x_i : f := f_{|x_i=0} \vee f_{|x_i=1}$ and $\forall x_i : f := f_{|x_i=0} \wedge f_{|x_i=1}$. The time needed to compute the quantification is determined by the computation time of two replacements by constant and one synthesis. In the rest of the paper quantifications over k Boolean variables $Qx_1, \ldots, x_k : f$ are denoted by $Qx : f$, where $x = (x_1, \ldots, x_k)$.

In implicit graph algorithms, the following operation (see, e. g., [27]) is useful to reverse the edges of a given graph.

Definition 2. *Let $k \in \mathbb{N}$, ρ be a permutation of $\{1, \ldots, k\}$ and $f \in B_{kn}$ with input vectors $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^n$. The argument reordering $\mathcal{R}_\rho(f) \in B_{kn}$ with respect to ρ is defined by $\mathcal{R}_\rho(f)(x^{(1)}, \ldots, x^{(k)}) := f(x^{(\rho(1))}, \ldots, x^{(\rho(k))})$.*

This operation can be computed by just renaming the variables and repairing the variable order using $3(k-1)n$ functional operations (see [5]).

An important variable order is the interleaved variable order which is defined on vectors of length n where the variables with the same significance are tested one after another.

Definition 3. *Let $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^n$ be input vectors and π be a permutation of $\{0, \ldots, n-1\}$. Then $\pi_{k,n} = (x^{(1)}_{\pi(0)}, x^{(2)}_{\pi(0)}, \ldots, x^{(k)}_{\pi(0)}, \ldots, x^{(1)}_{\pi(n-1)}, \ldots, x^{(k)}_{\pi(n-1)})$ is called* k-interleaved *variable order for $x^{(1)}, \ldots, x^{(k)}$. If $\pi = (n-1, \ldots, 0)$ then we say that the variables are tested with decreasing significance.*

An OBDD-based graph algorithm computes an output χ_O represented as an OBDD given a characteristic function χ_E as an input by mainly using functional operations. The running time depends on the actual size of the OBDDs used for the operations during the computation which is difficult to bound in general.

However, if the input OBDD size representing a graph is large, any algorithm using this OBDD is likely to have an inadequate running time. Beside the variable order, the labeling of the nodes is another optimization parameter with huge influence on the input size. For OBDDs representing state transitions in finite state machines, Meinel and Theobald [20] showed that there can be an exponential blowup of the OBDD size from a good labeling to a worst-case labeling. Nevertheless, a small input OBDD size does not guarantee a good running time since the sizes of the intermediate OBDDs do not have to be small. Indeed, an exponential blowup from input to output size is possible [27,3].

We denote by $f_{|x_{\pi(0)}=a_{\pi(0)},\ldots,x_{\pi(i-1)}=a_{\pi(i-1)}}$ the subfunction where $x_{\pi(j)}$ is replaced by the constant $a_{\pi(j)}$ for $0 \leq j \leq i-1$. The function f *depends essentially* on a variable x_i iff $f_{|x_i=0} \neq f_{|x_i=1}$. A characterization of minimal π-OBDDs due to Sieling and Wegener [28] can be often used to bound the OBDD size.

Theorem 1 ([28]). *Let $f \in B_n$ and for all $i = 0, \ldots, n-1$ let s_i be the number of different subfunctions which result from replacing all variables $x_{\pi(j)}$ with $0 \leq j \leq i-1$ by constants and which essentially depend on $x_{\pi(i)}$. Then the minimal π-OBDD representing f has s_i nodes labeled by $x_{\pi(i)}$.*

Basic Functions and Implicit Algorithms. The OBDD size of the equality $EQ(x, y)$ and greater than function $GT(x, y)$ with $EQ(x, y) = 1 \Leftrightarrow |x| = |y|$ and $GT(x, y) = 1 \Leftrightarrow |x| > |y|$ is linear in the number of input bits for an interleaved variable order (see, e. g., [29]). For the sake of code readability, we use $|x| = |y|$ and $|x| > |y|$ to denote $EQ(x, y)$ and $GT(x, y)$ in our algorithms. Furthermore, by $|x| > c$ ($|x| = c$) for some constant c we denote the function $GT(x, y)_{|y=[c]_2}$ ($EQ(x, y)_{|y=[c]_2}$). Every function $R(x, y) \in B_{2n}$ can be seen as a binary relation R on the set $\{0, 1\}^n$ with $x\, R\, y \Leftrightarrow R(x, y) = 1$. The transitive closure of $R(x, y)$ can be computed implicitly by $O(n^2)$ functional operations using the so called iterative squaring or path doubling technique (see, e. g., [14]).

Interval Graphs. Let $\mathcal{I} = \{[a_i, b_i] \mid a_i < b_i \text{ and } 0 \leq i \leq N-1\}$ be a set of N intervals on the real line. The *interval graph* $G_{\mathcal{I}} = (V, E)$ has one node for each interval in $\mathcal{I}$ and two nodes $v \neq w$ are adjacent iff the corresponding intervals intersect. If no interval is properly contained in another interval, $G_{\mathcal{I}}$ is called *proper interval graph*. If the length of every interval in $\mathcal{I}$ is equal to 1 then $G_{\mathcal{I}}$ is called *unit interval graph*. Notice that the set of all interval graphs does not change if we restrict ourselves to sets $\mathcal{I}$ where all endpoints are different. The definitions of proper and unit interval graphs are equivalent in the sense that they generate the same class of interval graphs [23]. Hence, in the following we only use the term of unit interval graphs. An undirected graph H is a (unit) interval graph iff there is a set of (unit) intervals $\mathcal{I}$ such that $H = G_{\mathcal{I}}$. Due to the one-to-one correspondence of the nodes of $G_{\mathcal{I}}$ and the elements of $\mathcal{I}$, we use the notion of node and interval synonymously.

3 OBDD Size of Interval Graphs

We present a way to count the subfunctions of the characteristic function χ_E of the edge set of a graph using the adjacency matrix of the graph which can give us a more graph theoretic approach to subfunctions.

The rows (columns) of an adjacency matrix correspond to the x-variables (y-variables) of $\chi_E(x, y)$. We can sort the rows of the adjacency matrix according to a variable order π by connecting the i-th row to the input x with $\sum_{l=0}^{n-1} x_{\pi(n-l-1)} \cdot 2^l = i$, i. e., we let the l-th x-variable in π have significance 2^{n-l-1} to sort the rows. This can be done analogously to sort the columns. Thus, the variable order π defines a permutation of the rows and columns of the adjacency matrix resulting in a new matrix which we call π-ordered adjacency matrix.

Definition 4. *Let $G = (V, E)$ be a graph and $\pi := \pi_{2,n}$ be a 2-interleaved variable order for the characteristic function $f := \chi_E$. The* π-ordered *adjacency matrix A_π of G is defined as follows: $a_{ij} = 1$ iff $f(x, y) = 1$ with $\sum_{l=0}^{n-1} x_{\pi(n-l-1)} \cdot 2^l = i$ and $\sum_{l=0}^{n-1} y_{\pi(n-l-1)} \cdot 2^l = j$.*

Notice that the π-ordered adjacency matrix is equal to the "normal" adjacency matrix where the rows and columns are sorted by the node labels iff the variables in π are tested with decreasing significance. The π-ordered adjacency matrix gives us a visualization of the subfunctions in terms of *blocks* of the matrix.

Definition 5. *Let $n \in \mathbb{N}$ and A be a $2^n \times 2^n$ matrix. For $0 \leq k \leq n$ and $0 \leq i, j \leq 2^k - 1$ the* block B_{ij}^k *of A is defined by the quadratic submatrix of size $2^n/2^k \times 2^n/2^k$ which is formed by the intersection of the rows $i \cdot 2^n/2^k, \ldots, (i + 1) \cdot 2^n/2^k - 1$ and the columns $j \cdot 2^n/2^k, \ldots, (j + 1) \cdot 2^n/2^k - 1$.*

Let i be even. Then the block $B_{|a|,|b|}^{i/2}$ represents the function table of the subfunction which results from replacing the first $i/2$ x-variables w.r.t π by $a \in \{0,1\}^{i/2}$ and the first $i/2$ y-variables by $b \in \{0,1\}^{i/2}$. Thus, counting the number of different blocks $B_{|a|,|b|}^{i/2}$ is equivalent to counting the number of different subfunctions.

Bollig and Wegener [7] use a similar approach to visualize subfunctions of a storage access function by building a matrix whose columns and rows are sorted according to the variable order and correspond to variables (not assignments as in our π-ordered matrix). Notice that A_π is not the communication matrix which is often used to show lower bounds of the OBDD size.

Theorem 2. *Let $\pi := \pi_{2,n}$ be the interleaved variable order with decreasing significance and $G = (V, E)$ be an interval graph with $N := |V|$ nodes. The π-OBDD size of χ_E can be bounded above by $O(N \log N)$.*

Proof. Let $f := \chi_E$, $1 \leq k \leq n$ and s_k be the number of different subfunctions $f_{|\alpha,\beta}$ of f where $\alpha \in \{0,1\}^k$ is an assignment to the variables $x_{n-1}, \ldots, x_{n-k}$ and $\beta \in \{0,1\}^k$ is an assignment to the variables $y_{n-1}, \ldots, y_{n-k}$, respectively. It is enough to bound s_k by above because replacing an additional variable by a constant can at most double the number of subfunctions.

We label the nodes according to their position in the sorted sequence of interval starting points. Recall that $a_{i,j}$ is one if and only if interval i intersects interval j. Now, notice that if $a_{j,i}$ is zero for $j > i$ then no interval $j' > j$ with a larger starting point can cut interval i. Thus, for every column $i \in \{0, \ldots, N-1\}$, the sequence $(a_{i+1,i}, \ldots, a_{N-1,i})$ is zero or starts with a continuous sequence of ones followed by only zeros (see also [21]).

Every subfunction $f_{|\alpha,\beta}$ corresponds to a block $B_{|\alpha|,|\beta|}^k$. Let $\beta = 0^k$ and $|\alpha| \geq 1$, i. e., we consider the blocks $B_{|\alpha|,0}^k$ of size $2^{n-k} \times 2^{n-k}$ (see Fig. 1). As we observed, every column of A_π has (below the diagonal) at most one possible *changing* position k such that $a_{k,i} = 1$ and $a_{k+1,i} = 0$. Looking at the sequence $(B_{1,0}^k, \ldots, B_{2^k-1,0}^k)$ of blocks, this fact implies that a block $B_{i,0}^k$ can only form a new block, i. e., all previous blocks in the sequence are different to this block, if

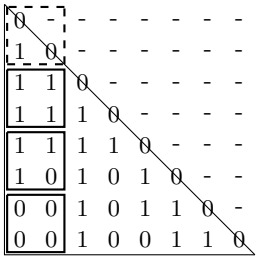

Fig. 1. Possible adjacency matrix with 8 nodes and framed subfunctions $f_{|\alpha,\beta}$ with $\beta = 0^k$, $|\alpha| \geq 1$, and $k = 2$.

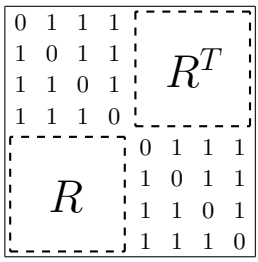

Fig. 2. Random interval graph where only R is generated randomly

there is a changing position in one column inside of $B^k_{i,0}$ or inside the block $B^k_{i-1,0}$ or between these two blocks. Therefore, every changing position can induce at most two different blocks and we can bound the number of different blocks by two times the number of possible changing positions which is at most the number of columns of a block, i. e., $2 \cdot 2^{n-k}$. Since the graph is symmetric and there are 2^k blocks on the diagonal, we can bound the overall number of different blocks by $O(2^{n-k} \cdot 2^k + 2^k) = O(2^n)$ and thus $s_k = O(2^n)$. Summing this up over all possible values of k, the π-OBDD size is at most $O(2^n \cdot n) = O(N \log N)$. □

In [22] it is proved that the OBDD size of unit interval graphs is $\Omega(N/\log N)$ and $O(N/\sqrt{\log N})$ which we can improve by using the π-ordered adjacency matrix.

Theorem 3. *Let π be the interleaved variable order with decreasing significance. The π-OBDD size of χ_E for a unit interval graph $G = (V, E)$ is $O(N/\log N)$.*

The difference between unit and general interval graphs is that in general interval graphs there is no dependence between the columns of the π-ordered adjacency matrix, which is important for our lower bound, while in unit interval graphs, the row number of the last 1 entry in a column is increasing from left to right.

The upper bound proof suggests that the number of blocks $B^k_{i,j}$ with a changing position roughly determines the number of x_{n-k-1}-nodes of the OBDD. However, explicitly constructing a worst-case interval graph with OBDD size of $\Omega(N \log N)$ is difficult because there are many dependencies between blocks for different values of k, since a block $B^k_{i,j}$ results from dividing some block $B^{k-1}_{i',j'}$.

In order to overcome these dependencies, we look at a random interval graphs and compute the expected value of the number of different blocks for $\Omega(n)$ values of k. We choose the length of the 1-sequence of column j for all $0 \leq j \leq \frac{N}{2} - 1$ uniformly at random from $\{\frac{N}{2} - j, \ldots, N - 1 - j\}$ and for all $\frac{N}{2} \leq j \leq N - 1$ we set the length to $N - 1 - j$. Thus, the lengths of the 1-sequences within the $\frac{N}{2} \times \frac{N}{2}$ lower left submatrix R are uniform at random in $\{1, \ldots, \frac{N}{2}\}$ (see Fig. 2).

Lemma 1. *Let $G = (V, E)$ be a random interval graph generated by the above process. The probability that a fixed block in R of size $L_1 \times L_2$ with $L_1, L_2 \leq 2^{n/2-1}$ has exactly one changing position is at least $\frac{L_2 \cdot (L_1 - 1)}{2^{n-1}} \cdot e^{-1}$.*

Theorem 4. *The worst-case π-OBDD size of an interval graph is $\Omega(N \log N)$ where the nodes are labeled according to the interval starting points and π is an interleaved variable order with decreasing significance.*

Proof. Let $G = (V, E)$ be a random interval graph generated by the above process and $f := \chi_E$. We know that each $n/2+1 \leq k \leq (3/4)n$ induces a grid in R consisting of $2^{k-1} \cdot 2^{k-1}$ blocks $B^k_{i,j}$ of size $2^{n-k} \times 2^{n-k}$. Using Lemma 1, the expected number of blocks with exactly one changing position is at least $\frac{1}{2e} \cdot 2^k \cdot 2^k \cdot \frac{2^{n-k} \cdot (2^{n-k}-1)}{2^n} = \Omega(2^n)$. Now, we have to ensure that these blocks correspond to different subfunctions which are also essentially dependent on x_{n-k-1}. The subfunctions, where, additionally, x_{n-k-1} is replaced by 0 and 1, correspond to a half of the blocks. Thus, a block is symmetric iff the corresponding subfunction is not essentially dependent on x_{n-k-1}. Due to the one changing position in each block, this is not possible. Blocks $B^k_{i,j}$ and $B^k_{i',j}$ with exactly one changing position and $i \neq i'$ clearly correspond to different subfunctions because they are in the same block column. But blocks $B^k_{i,j}$ and $B^k_{i',j'}$ with $j \neq j'$, i. e., from different block columns, do not have to be different. By replacing some columns of the blocks by constants, we ensure that this also holds. Consider the case $k = (3/4)n$. For every $0 \leq j \leq 2^k - 1$ we fix the first k columns of $B^k_{i,j}$ with $0 \leq i \leq 2^k - 1$ such that they represent the binary number $[j]_2$. As a result, the blocks $B^k_{i,j}$ and $B^k_{i',j'}$ with $j \neq j'$ are always different. Since we looked at the finest grid, this also holds for smaller values of k because every larger block is equal to a union of small blocks. For $k = (3/4)n$ the number of fixed columns is $(3/4)n$ and in each $k \to k-1$ step this number is doubled, i. e., for $n/2+1 \leq k \leq (3/4)n$ the number of "free" columns is $2^{n-k} - 2^{(3/4)n-k} \cdot (3/4)n = \Omega(2^{n-k})$ for n large enough. Thus, the expected number of blocks with exactly one changing position remains $\Omega(2^n)$ for every $n/2+1 \leq k \leq (3/4)n$ which means there is an interval graph with π-OBDD size $\Omega(N \log N)$ □

4 Implicit Matching Algorithm on Unit Interval Graphs

In the following, the nodes of the unit interval graphs are labeled according to the sorted sequence of starting points. At first, we have to look into so called multivariate threshold functions.

Definition 6 (see, e. g., [30]). *Let $T \in \mathbb{Z}$ and $W \in \mathbb{N}$ and $w_1, \ldots, w_k \in \{-W, \ldots, W\}$. A Boolean function $f : \{0,1\}^{kn} \to \{0,1\}$ with input vectors $x^{(1)}, \ldots, x^{(k)} \in \{0,1\}^n$ and $f(x^{(1)}, \ldots, x^{(k)}) = 1 \Leftrightarrow \sum_{j=1}^{k} w_j \cdot |x^{(j)}| \geq T$ is called* k-variate threshold function. *The set of k-variate threshold functions $f \in B_{kn}$ with weight parameter W is denoted by $\mathbb{T}^W_{k,n}$.*

Woelfel [30] investigated the OBDD size for the variable order where the variables are tested with increasing significance, i. e., just the reverse of our variable order. Hosaka et al. [17] showed that the difference of the OBDD sizes for this two orders is at most $n-1$. We can show that an OBDD using our variable order is not only small but can also be constructed efficiently which is important in view of

the implementation. The result also implies that we can compute a sequence of $O(1)$ binary synthesis of multivariate threshold functions efficiently using our variable order if k and W are constants.

Theorem 5. *Let $f \in \mathbb{T}^W_{k,n}$ and $\pi_{k,n}$ be the k-interleaved variable order where the variables are tested with decreasing significance. Then we can construct a $\pi_{k,n}$-OBDD representing f with width $O(kW)$ and size $O(k^2Wn)$ in time $O(k^2Wn)$.*

We use the arithmetic notation in our algorithm instead of the functional notation, e. g., we denote by $|x|-|y| = 1$ the conjunction of the multivariate threshold functions $f(x, y) = 1 \Leftrightarrow |x| - |y| \geq 1$ and $g(x, y) = 1 \Leftrightarrow |y| - |x| \geq -1$.

Maximum Matching on Unit Interval Graphs. Let $G = (V, E)$ be a unit interval graph. Then we can make a simple observation (see [10]): W.l.o.g. G is connected. Then we have $\{v_i, v_{i+1}\} \in E$ for all $i = 0, \ldots, N-2$. Assume that there is an i such that $\{v_i, v_{i+1}\} \notin E$, then due to the connectivity there has to be another interval with starting point left of v_i and length greater than 1 (which is not possible) intersecting both intervals.

Algorithm 1 uses the characteristic function of the set of nodes which is here equal to $f(x) = 1 \Leftrightarrow |x| < N$. The algorithm computes a directed subgraph of G, which consists of the vertex disjoint paths visiting all nodes in the connected components. Maximum matchings on arbitrary vertex disjoint paths can be computed with $O(\log^2 N)$ functional operations [6]. Here, we know that every path consists of a consecutive sequence of nodes. We compute the set of starting nodes of the paths and the connected components of the graph: Two nodes x and y are connected iff every node z with $|x| \leq |z| < |y|$ has a successor $(v_{|z|}, v_{|z|+1}) \in E$. This can be computed by $O(\log N)$ operations. Next, we can compute the matching by adding every second edge of a path to the matching beginning with the first edge. While computing this set of edges needs $O(\log^2 N)$ operations in general [6], here we can easily determine the set by comparing the difference of two node labels due to the structure of the paths.

Algorithm 1. Implicit maximum matching algorithm for unit interval graphs

Input: Unit interval graph χ_E
Output: Matching χ_M
// Compute path graph
1: $\chi_{\overrightarrow{E}}(x, y) = \chi_E(x, y) \wedge (|y| - |x| = 1)$
// Compute set of starting nodes
2: $First(z) = (|z| < N) \wedge \forall x : \overline{\chi_{\overrightarrow{E}}(x, z)}$
// Compute set of reachable nodes
3: $S(z) = \exists z' : \chi_{\overrightarrow{E}}(z, z')$
4: $Reachable(x, y) = (|x| \leq |y|) \wedge \forall z : (|x| \leq |z| < |y|) \Rightarrow S(z)$
5: $Reachable(x, y) = Reachable(x, y) \wedge (|x| < N) \wedge (|y| < N)$
// Compute matching
6: $F(x) = \exists z, d : First(z) \wedge Reachable(z, x) \wedge (|x| - |z| = 2|d|)$
7: $M(x, y) = \chi_{\overrightarrow{E}}(x, y) \wedge F(x)$
8: $\chi_M(x, y) = M(x, y) \vee M(y, x)$
9: **return** χ_M

Theorem 6. *Algorithm 1 computes a maximum matching for unit interval graphs using $O(\log N)$ functional operations.*

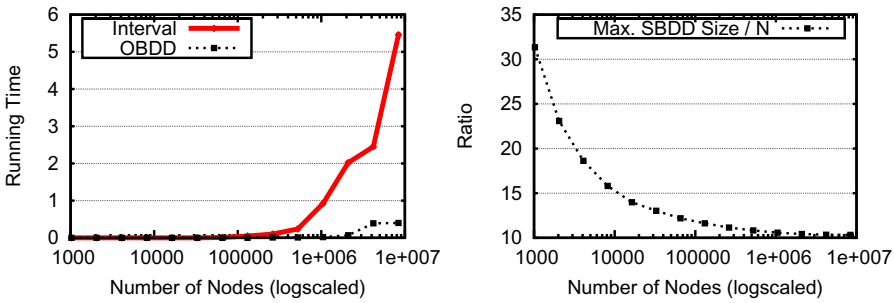

Fig. 3. Runtime and memory of the matching algorithms on random unit interval graphs. Memory plot shows the ratio of $S \log S$ (space usage of the OBDD-based algorithm) and number of nodes.

5 Experimental Evaluation

We evaluated the implicit maximum matching algorithm on unit interval graphs. Unit interval graphs can be represented as balanced nonnegative strings over {'[', ']'} (see, [24]) and such strings are created randomly using the algorithm in [1]. We generated 35 random graphs of size 2^i for $i = 10, \ldots, 23$. The nodes of the graphs are encoded as in Section 3. We compare the OBDD-based algorithm to the algorithm which gets the interval representation as an input, sort the intervals according to their starting point and compute a maximum matching by scanning this sorted sequence with the same idea used in the implicit algorithm.

Experimental Setup. We implemented the implicit algorithm with the BDD framework CUDD 2.5.0[1] by F. Somenzi. The algorithms are implemented in C++ and were compiled with Visual Studio 2012 in the default release configuration. All source files, scripts and random seeds are publicly available[2]. The experiments were performed on a computer with a 2.6 GHz Intel Core i5 processor and 4 GB main memory running Windows 7. The runtime is measured by used processor time in seconds and the space usage of the implicit algorithm is given by the maximum SBDD size which came up during the computation, where a SBDD is a collection of OBDDs which can share nodes. Due to the small variance of these values, we only show the mean in the diagrams.

Results. The implicit matching algorithm outperforms the explicit matching algorithm on unit interval graphs (see Fig. 3). Even on graphs with more than 8 million nodes the implicit algorithm computes a maximum matching within 1 seconds. Storing a SBDD of size S needs $O(S \log S)$ bits. The memory diagram shows that the asymptotic space usage of the implicit algorithm on these instances is close to $O(N)$. Recall that the unit interval representation needs $\Theta(N \log N)$ space since $\log N$ bits are needed to represent the starting points.

[1] `http://vlsi.colorado.edu/~fabio/CUDD/`
[2] `http://ls2-www.cs.uni-dortmund.de/~gille/`

I. e., the implicit algorithm needs less space and can compute a maximum matching on larger instances than the explicit one. An interesting consequence of these results is that the submodules of our maximum matching algorithm, namely computing the connected components, a Hamiltonian path in every connected component and a maximum matching on these paths, are also very fast and space efficient which is surprising, since especially the computation of the transitive closure is often a bottleneck in implicit algorithms.

Open Questions. Using the π-ordered adjacency matrix, we think that it is possible to bound the OBDD size for other graph classes with a well structured adjacency matrix, e. g., convex graphs where the nodes can be ordered such that the neighborhood of every node consists of nodes which are consecutive in this order. The gap between the upper and lower bound of the OBDD size of interval graphs is $O(\log N)$. It is an interesting open question whether there is another labeling and/or variable order such that the OBDD size is $O(N)$ or the general lower bound can be increased to $\Omega(N \log N)$.

Even for a fixed variable order, the complexity of computing a node labeling for a given graph, such that the representing OBDD has minimal size, is unknown. The π-ordered adjacency matrix seems to help to prove upper/lower bounds on the OBDD size for a fixed labeling. Using this matrix to bound the size of OBDDs for every labeling could be object of further research.

The investigation of implicit algorithms on special graph classes seems quite promising and it would be interesting if the good performance can also be achieved for other large graph classes.

Acknowledgements. I thank Beate Bollig, Melanie Schmidt and Chris Schwiegelshohn for the valuable discussions and, together with the anonymous referees, for their comments on the presentation of the paper.

References

1. Arnold, D.B., Sleep, M.R.: Uniform random generation of balanced parenthesis strings. ACM Trans. Program. Lang. Syst. 2(1), 122–128 (1980)
2. Bloem, R., Gabow, H.N., Somenzi, F.: An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. Formal Methods in System Design 28(1), 37–56 (2006)
3. Bollig, B.: On symbolic OBDD-based algorithms for the minimum spanning tree problem. TCS 447, 2–12 (2012)
4. Bollig, B., Gillé, M., Pröger, T.: Implicit computation of maximum bipartite matchings by sublinear functional operations. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 473–486. Springer, Heidelberg (2012)
5. Bollig, B., Löbbing, M., Wegener, I.: On the effect of local changes in the variable ordering of ordered decision diagrams. IPL 59(5), 233–239 (1996)
6. Bollig, B., Pröger, T.: An efficient implicit OBDD-based algorithm for maximal matchings. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 143–154. Springer, Heidelberg (2012)
7. Bollig, B., Wegener, I.: Asymptotically optimal bounds for OBDDs and the solution of some basic OBDD problems. Journal of Computer and System Sciences 61(3), 558–579 (2000)

8. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers 35(8), 677–691 (1986)
9. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98(2), 142–170 (1992)
10. Chung, Y., Park, K., Cho, Y.: Parallel maximum matching algorithms in interval graphs. In: Proc. of the 4th ICPADS, pp. 602–609 (1997)
11. Coudert, O.: Doing two-level logic minimization 100 times faster. In: Proc. of the 6th SODA, pp. 112–121 (1995)
12. Gentilini, R., Piazza, C., Policriti, A.: Computing strongly connected components in a linear number of symbolic steps. In: Proc. of the 14th SODA, pp. 573–582 (2003)
13. Gentilini, R., Piazza, C., Policriti, A.: Symbolic graphs: Linear solutions to connectivity related problems. Algorithmica 50(1), 120–158 (2008)
14. Gillé, M.: OBDD-Based Representation of Interval Graphs. ArXiv e-prints (2013), http://arxiv.org/abs/1305.2772
15. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Discrete Mathematics, vol. 57. North-Holland Publishing Co. (2004)
16. Hachtel, G.D., Somenzi, F.: A symbolic algorithms for maximum flow in 0-1 networks. Formal Methods in System Design 10(2/3), 207–219 (1997)
17. Hosaka, K., Takenaga, Y., Kaneda, T., Yajima, S.: Size of ordered binary decision diagrams representing threshold functions. Theor. Comput. Sci. 180(1-2), 47–60 (1997)
18. Lai, Y.-T., Pedram, M., Vrudhula, S.B.K.: EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition. IEEE Transactions on CAD of Integrated Circuits and Systems 13(8), 959–975 (1994)
19. Meer, K., Rautenbach, D.: On the OBDD size for graphs of bounded tree- and clique-width. Discrete Mathematics 309(4), 843–851 (2009)
20. Meinel, C., Theobald, T.: On the influence of the state encoding on OBDD-representations of finite state machines. ITA 33(1), 21–32 (1999)
21. Mertzios, G.B.: A matrix characterization of interval and proper interval graphs. Applied Mathematics Letters 21(4), 332–337 (2008)
22. Nunkesser, R., Woelfel, P.: Representation of graphs by OBDDs. Discrete Applied Mathematics 157(2), 247–261 (2009)
23. Roberts, F.S.: Indifference graphs. In: Harary, F. (ed.) Proof Techniques in Graph Theory, pp. 139–146 (1969)
24. Saitoh, T., Yamanaka, K., Kiyomi, M., Uehara, R.: Random generation and enumeration of proper interval graphs. IEICE Transactions 93-D(7), 1816–1823 (2010)
25. Sawitzki, D.: Implicit flow maximization by iterative squaring. In: Van Emde Boas, P., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2004. LNCS, vol. 2932, pp. 301–313. Springer, Heidelberg (2004)
26. Sawitzki, D.: The complexity of problems on implicitly represented inputs. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 471–482. Springer, Heidelberg (2006)
27. Sawitzki, D.: Exponential lower bounds on the space complexity of OBDD-based graph algorithms. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 781–792. Springer, Heidelberg (2006)
28. Sieling, D., Wegener, I.: NC-algorithms for operations on binary decision diagrams. Parallel Processing Letters 3, 3–12 (1993)
29. Wegener, I.: Branching programs and binary decision diagrams. SIAM Monographs on Discrete Mathematics and Applications (2000)
30. Woelfel, P.: Symbolic topological sorting with OBDDs. Journal of Discrete Algorithms 4, 51–71 (2006)