Symbolic Topological Sorting with OBDDs

Philipp Woelfel
University Dortmund
FB Informatik 2
D-44221 Dortmund
Germany.
E-mail: `philipp.woelfel@cs.uni-dortmund.de`

Abstract

We present a symbolic OBDD algorithm for topological sorting which requires $O(\log^2 |V|)$ OBDD operations. Then we analyze its true runtime for the directed grid graph and show an upper bound of $O(\log^4 |V| \cdot \log\log |V|)$. This is the first true runtime analysis of a symbolic OBDD algorithm for a fundamental graph problem, and it demonstrates that one can hope that the algorithm behaves well for sufficiently structured inputs.

1 Introduction

Algorithms on graphs is one of the best studied areas in computer science. Usually, a graph $G = (V, E)$ is given by an adjacency list or by an adjacency matrix. Such an explicit representation of a graph requires space $\Theta(|V| + |E|)$ or $\Theta(|V|^2)$, and for many graph problems efficient algorithms are known. However, there are several application areas where typical problem instances have such a large size that a linear or even polynomial runtime is not feasible, or where even the explicit representation of the problem instance itself may not fit into memory anymore. Examples where this is the case occur if large graphs like the Internet graph or the street network of a major city are interlinked with other components such as e.g. traffic amount and time slots.

Areas where researchers have been dealing with the problem of large input instances in the last decade are e. g. symbolic model checking or circuit verification. 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. In order to deal with such large graphs, *symbolic* (or *implicit*) graph algorithms have been devised, where the vertex and edge sets representing the involved graphs are stored symbolically, i.e., in terms of their characteristic functions. A very popular data structure for representing these characteristic functions are the so-called Ordered Binary Decision Diagrams (OBDDs), which we define in the next section. Symbolic algorithms using OBDDs have been successfully applied in the areas of model checking, circuit verification, finite state machine verification, integer linear programming, and logic minimization (see e.g. [3, 4, 5, 11, 6]).

Most of these applications can be viewed as particular cases of graph problems. This raises the question whether it is possible to devise symbolic graph algorithms with a good behavior for fundamental graph theoretical problems. One approach in this direction was undertaken by Hachtel and Somenzi [9] who introduced a symbolic OBDD algorithm for the maximum flow problem in 0-1 networks. The promising experimental studies demonstrated that the algorithm is able to handle graphs with over 10^{36} edges and that it is competitive with traditional algorithms on dense random graphs. The paper lacks however, a theoretical analysis of its performance with respect to runtime. Recently, Sawitzki [13] has analyzed the number of OBDD operations (i.e., the number of required synthesis operations of characteristic functions) required by the flow algorithm of Hachtel and Somenzi and has proposed an improved algorithm. But there is only a weak relation between the number of OBDD operations and the true runtime of a symbolic OBDD algorithm. The time required for one synthesis step is mainly influenced by the sizes of the involved OBDDs which may range from linear to exponential (in the number of variables of the represented characteristic functions).

However, true runtime analyses of OBDD algorithms are very rare (there are some examples, though, as e.g. in [7]) and in fact, we are not aware of any true runtime analysis of a symbolic OBDD algorithm for a general, fundamental graph problem. A reason for this may be that in most cases a worst-case or average-case analysis cannot yield results with a better expressiveness than the analysis of the number of required OBDD operations, because one has to expect that the representation of most of the characteristic functions obtained during the computation has asymptotically the maximum possible size, which is at least the size of an explicit representation. Another reason why the worst-case analysis of fundamental graph algorithms is hopeless in most cases is that even such a simple decision problem as a reachability test is PSPACE complete if the input graph is represented by OBDDs [8].

But the attractivity of implicit algorithms stems from the prospect that they may have superior performance for well structured problem instances as they arise in typical situations. For example, the street network of an American city like Manhattan resembles more a grid graph than a random graph. Hence, research should focus on developing implicit OBDD algorithms for fundamental graph problems and on analyzing their true runtimes for inputs chosen from certain typical graph classes.

The results and techniques presented here aim to be a first step into this direction. First, we present a new OBDD algorithm for topological sorting which requires only $O(\log^2 |V|)$ OBDD operations on OBDDs for functions with at most $4\lceil \log |V| \rceil$ variables. Then we analyze its true runtime for the directed grid graph and show an upper bound of $O(\log^4 |V| \cdot \log\log |V|)$. This demonstrates that one can in fact hope that such a fundamental graph algorithm behaves well for sufficiently structured inputs. For the analysis, we generalize the notion of threshold functions to multivariate threshold functions. We investigate the OBDD size of multivariate threshold (and modulo) functions and obtain strong results about the effect of OBDD operations such as quantification on such functions. We expect the framework developed here to be strong enough to allow true runtime analyses of other algorithms, where the input is a grid or a grid-like network. In fact, Sawitzki [14] refined our framework in order to analyze his 0-1 network flow algorithm for the grid network.

Clearly, our analysis is a "good-case" analysis which is only valid for one particular input instance, and we are not yet able to make a theoretically profound statement about the performance of the algorithm for other problem instances. But our analysis shows that even for such a well-structured input graph as the directed grid a quite detailed knowledge on the behavior of OBDDs for certain function types is required in order to obtain an expressive upper bound on the true runtime. We hope that the techniques presented here are a good starting point for developing a framework which allows to design and analyze general fundamental graph algorithms for larger classes of input instances.

2 OBDDs and Implicit Graph Representation

In the following, let B_n denote the class of boolean functions $\{0,1\}^n \to \{0,1\}$. Let $f \in B_n$ be a function defined by the variables $x_1, \ldots, x_n$. The subfunction of f, where k variables $x_{i_1}, \ldots, x_{i_k}$ are fixed to k constants $c_1, \ldots, c_k \in \{0,1\}$ is denoted by $f_{|x_{i_1}=c_1,\ldots,x_{i_k}=c_k}$. Ordered Binary Decision Diagrams have been introduced by Bryant in 1986 [2] as a representation type 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 permutation of the indices $\{1, \ldots, n\}$, leading to the ordered list $x_{\pi(1)}, \ldots, x_{\pi(n)}$ 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, resp., 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 $x_{\pi^{-1}(i)} < x_{\pi^{-1}(j)}$. This means that any directed path passes the nodes in an order respecting the variable ordering π.*
3. *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 $f(a)$.*
4. *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 (short: π-*OBDD*(f)) is the size of the minimum π-*OBDD *computing f.*

The restriction of the variable ordering implies that each variable may appear on each source-to-sink path at most once. Note that in some papers the term Binary Decision

Diagram (or BDD) is used interchangeably for OBDD, while in others the term BDD denotes a decision diagram without the restrictions imposed by the variable ordering.

It is important to note that the π-OBDD of minimal size for a given function f and a fixed variable ordering π is unique up to isomorphism. A π-OBDD is called *reduced*, if it is the minimal π-OBDD. It is well-known that the maximum size of any reduced π-OBDD for a function in n variables is bounded by $O(2^n/n)$ (see [1] for the upper bound with the best constants known).

In order to be a representation type for boolean functions which is suitable for graph algorithms, it is necessary that several operations can be performed efficiently. In the following, we summarize the operations on OBDDs to which we will refer in this text. For a more detailed discussion on OBDDs and their operations we refer to the monograph [15].

Let f and g be functions in B_n and let G_f and G_g be π-OBDDs representing f and g, respectively, for an arbitrary variable ordering π.

- **Evaluation:** Given $x \in \{0,1\}^n$ compute $f(x)$. This can trivially be done in time $O(n)$.
- **Minimization:** Compute the reduced π-OBDD for f. This is possible in time $O(|G_f|)$.
- **Binary synthesis:** Given a boolean operation $\otimes \in B_2$ compute a reduced π-OBDD G_h representing the function $h = f \otimes g$. This can be done in time $O(|G_h^*| \log |G_h^*|)$, where G_h^* is the graph which consists of all nodes in the product graph of G_f and G_g reachable from the root. The size of G_h is at most $O(|G_h^*|) = O(|G_f| \cdot |G_g|)$.
- **Replacement by constants:** Given a sequence of variables $x_{i_1}, \ldots, x_{i_k} \in X_n$ and a sequence of constants $c_1, \ldots, c_k$, compute a reduced π-OBDD G_h for the subfunction $h := f_{|x_{i_1}=c_1,\ldots,x_{i_k}=c_k} \in B_{n-k}$. This is possible in time $O(|G_f|)$ and the reduced π-OBDD G_h is of smaller size than G_f.
- **Quantification:** Given a variable $x_i \in X_n$ and a quantifier $Q \in \{\exists, \forall\}$, compute a reduced π-OBDD for the function $h \in B_{n-1}$ with $h := (Qx_i)f$, where $(\exists x_i)f :=$

$f_{|x_i=0} \vee f_{|x_i=1}$ and $(\forall x_i) f := f_{|x_i=0} \wedge f_{|x_i=1}$. The time for computing this π-OBDD is determined by the time for determining the π-OBDDs for $f_{|x_i=0}$ and $f_{x_i=1}$ and the time required for the binary synthesis of the two. Hence, it is bounded by $O\big(|G_f|^2 \log |G_f|\big)$.

- **SAT enumeration:** Enumerate all inputs $x \in f^{-1}(1)$. Using simple DFS techniques, this can be done in optimal time $O(|G_f| + n\big|f^{-1}(1)\big|)$.

We can use OBDDs for an implicit graph representation by letting them represent the characteristic functions of the vertex and edge sets. For practical reasons, though, we assume throughout this text that the vertex set is $V = \{0,1\}^n$ for some $n \in \mathbb{N}$, so that a representation of V is not needed. It is easy to accommodate the algorithm for other vertex sets. We delay the discussion of this matter until we have described the algorithm. Note also that in contrast to the standard notation, we denote with n not the number of vertices but the number of bits required for the description of a vertex.

For an arbitrary relation R over $\{0,1\}^n$, we say that a π-OBDD G_R represents R, if the boolean function represented by G is the characteristic function χ_R of R, that is $x\,R\,y$ if and only if $\chi_R(x,y) = 1$. This way, the edge relation $E \subseteq V \times V$ of a directed graph can be represented by the π-OBDD for its characteristic function. For the ease of notation we write $E(x,y)$ instead of $\chi_E(x,y)$.

If we want to encode integers, we use the standard binary notation. Let $\mathrm{val}_n : \{0,1\}^n \to \mathbb{N}$ be the mapping

$$x_{n-1} \ldots x_0 \mapsto 2^{n-1} x_{n-1} + \cdots + 2^0 x_0.$$

For the ease of notation, we write $|x|$ instead of $\mathrm{val}_n(x)$ if the length of the string x is clear from the context.

3 The Topological Sorting Algorithm

Let $G = (V,E)$, $V = \{0,1\}^n$, be a directed acyclic graph represented by a π-OBDD as described in the former section. The edge relation E defines in a natural way a partial order on V, where v precedes w if and only if there exists a path from v to w.

We use the symbol $\rightsquigarrow$ for the corresponding relation, i.e., $v \rightsquigarrow w$ if v precedes w. In the following we describe a symbolic OBDD-algorithm which topologically sorts the vertices according to the relation $\rightsquigarrow$.

It is necessary, though, to discuss the possible outputs of such an algorithm. In the explicit case a topological sorting algorithm would enumerate all vertices in such a way that if u is enumerated before v, then $v \not\rightsquigarrow u$. In the implicit case, we hope for runtimes in the order of $o(|V|)$ in which the enumeration of all vertices is not possible. Hence, a goal might be to obtain a complete order $\prec$ which inherits the properties of $\rightsquigarrow$ (i.e., $u \prec v$ implies $v \not\rightsquigarrow u$). Unless $\rightsquigarrow$ is a complete order, $\prec$ is not uniquely defined by $\rightsquigarrow$, and thus we assume that an arbitrary complete order $\lessdot$ on the vertex set V is given (this may be fixed in advance for the algorithm or may be given as an additional parameter), which determines the order of the elements which are incomparable with respect to $\rightsquigarrow$ (i.e., those with $u \not\rightsquigarrow v$ and $v \not\rightsquigarrow u$).

An alternative is to compute an OBDD which allows to enumerate the elements in their topological order by simple SAT enumeration operations. For any two vertices u, v we denote by $\Delta(u, v)$ the length of the longest path leading from u to v. (The length of a path is the number of its edges.) If no such path exists, then $\Delta(u, v) := -\infty$. Note that $\Delta(v, v) = 0$, since the graph is acyclic. Furthermore, let $\Delta(v) := \max\{\Delta(u, v) \mid u \in V\}$. We call $\Delta(v)$ the length of the *longest path to* the vertex v. Let now $\mathrm{DIST} \in B_{2n}$, be defined to be 1 for an input $(d, v) \in \{0,1\}^n \times \{0,1\}^n$, if and only if $\Delta(v) = |d|$. Clearly, $|d_u| < |d_v|$ implies $v \not\rightsquigarrow u$, where d_u, d_v are the unique values with $\mathrm{DIST}(d_u, u) = 1$ and $\mathrm{DIST}(d_v, v) = 1$. Hence, if we have a π-OBDD G_{DIST} for the function DIST, we can use it to enumerate the vertices in an order respecting $\rightsquigarrow$ by computing the π-OBDDs for $\mathrm{DIST}_{|d=a}$ for $|a| = 0, 1, \ldots$ and enumerating their satisfying inputs using the SAT enumeration procedure. We will see below how the OBDD G_{DIST} can in addition be used to obtain a complete order respecting $\rightsquigarrow$.

In order to compute the function DIST, we use a method which is similar to that of computing the transitive closure by matrix squaring. For $i \in \{1, \ldots, n\}$ and $u, v \in V$ let $T_i(u, v)$ be the boolean function with function value 1 if and only if there exists a path from u to v which has length exactly 2^i. We can compute OBDDs for all T_i as follows.

$$T_0(u,v) \;=\; E(u,v). \tag{S1}$$

$$T_{i+1}(u,v) \;=\; \exists w:\; T_i(u,w) \wedge T_i(w,v). \tag{S2}$$

We now define the function $\mathrm{DIST}_j \in B_{2n-j}$ for $0 \le j \le n$. It takes as input an $(n-j)$-bit value $d^* = d_{n-1} \ldots d_j$ and a vertex v (for $j = n$, d^* is the empty string ϵ). The function value $\mathrm{DIST}_j(d^*, v)$ is defined as

$$\mathrm{DIST}_j(d^*,v) \;=\; 1 \quad \Leftrightarrow \quad 2^j|d^*| \le \Delta(v) < 2^j(|d^*|+1). \tag{$*$}$$

I.e., $\mathrm{DIST}_j(d^*, v)$ is true if the bits $d_{n-1} \ldots d_j$ are exactly the $n-j$ most significant bits of the integer $\Delta(v)$. Clearly, $\mathrm{DIST} = \mathrm{DIST}_0$. As we show below, the functions DIST_j can be computed by

$$\mathrm{DIST}_n(v) := 1 \tag{S3}$$

and for $j = n-1, \ldots, 0$

$$\begin{aligned}\mathrm{DIST}_j(d_{n-1}\ldots d_j, v) \;=\; & \mathrm{DIST}_{j+1}(d_{n-1}\ldots d_{j+1}, v) \quad \wedge \\ & \Big(d_j \Leftrightarrow \exists u\ \big(T_j(u,v)\ \wedge\ \mathrm{DIST}_{j+1}(d_{n-1}\ldots d_{j+1}, u)\big)\Big).\end{aligned} \tag{S4}$$

Before we prove the correctness of step (S4), we demonstrate the computation of the functions T_i and DIST_j by an example.

Example 1 Consider the graph $G = (V, E)$ with $V = \{a, b, c, d, e, f, g, h\}$ shown as the first graph in Figure 3. The relations T_1 and T_2 obtained by the step (S2) of the algorithm are represented by the edges shown in the second and third graph, respectively. It can be seen that in the graph (V, T_{i+1}) there is an edge from u to v if and only if there is a path of length 2 from u to v in the graph (V, T_i). Hence, the relation T_2 connects those vertex pairs which have in G a path of length 4 between them.

In the first graph in Figure 3, the three bits beneath each vertex v of G denote the unique value $d = d_2d_1d_0$ such that $\mathrm{DIST}(d, v) = 1$. For example, the longest path to g leads from a over the vertices b and f to g. This path has length 3 and hence $\Delta(g) = 3$

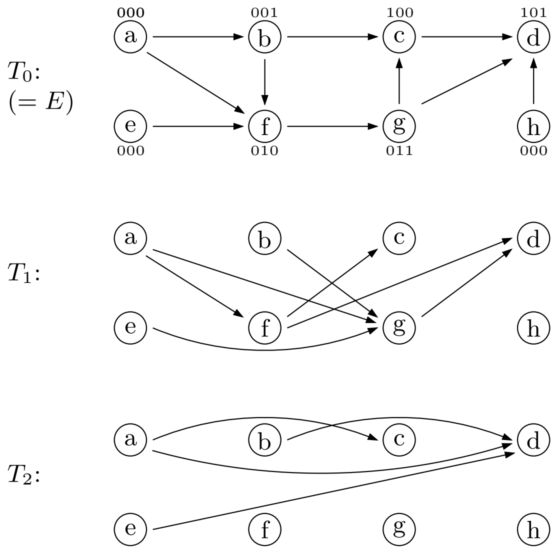

Figure 1: Example for the relations $T_0 = E$, T_1, T_2 and DIST

and $\mathrm{DIST}(011, g) = 1$. We demonstrate the computation of the functions DIST_i in the steps (S3) and (S4). The function DIST_3 is initialized in step (S3) such that it is true for all vertices. For $j = 2$, (S4) now simplifies to

$$\mathrm{DIST}_2(d_3, v) \;\; = \;\; \exists u \, T_j(u, v).$$

Hence, DIST_2 is true for exactly those pairs $(1, v)$ for which there is in the graph $G(V, T_2)$ an edge pointing to v, and for the pairs $(0, v)$ where v has no edge pointing to it. I.e. DIST_2 is true for the inputs in $\{(0, a), (0, b), (1, c), (1, d), (0, e), (0, f), (0, g), (0, h)\}$. In order to demonstrate the computation of DIST_1 we consider the vertex g. It is $\mathrm{DIST}_2(0, b) = \mathrm{DIST}_2(0, g) = 1$, hence the third bit (the most significant one) of $\Delta(b)$ is the same as that of $\Delta(g)$. Since $(b, g) \in T_1$, it is $\Delta(g) \geq \Delta(b) + 2$ and thus the second bit of $\Delta(g)$ is set. Consequently, step (S4) of the algorithm yields $\mathrm{DIST}_1(01, g) = 1$. For another example consider the vertex d. It is $\mathrm{DIST}_2(1, d) = 1$, hence the most significant bit of $\Delta(d)$ is set (which means that the length of the longest path to d is at least 4). The other vertices for which this is the case are a, c and e. But none of these vertices has an outgoing edge pointing to d in the graph (V, T_1) and thus $\Delta(d) < \Delta(v) + 2$ for all $v \in \{a, c, e\}$. Hence, the second bit of $\Delta(d)$ is not set and consequently step (S4) yields $\mathrm{DIST}_2(10, d) = 1$.

In order to see that the steps (S3) and (S4) of the algorithm are correct, we have to verify $(*)$.

Claim 1 *The algorithm described by* (S3) *and* (S4) *computes* DIST_j *correctly.*

Proof: The proof is by induction on j. For $j = n$ the claim is obvious because in this case d^* is the empty string ϵ and $|\epsilon| = 0$. Let now $j < n$ and $d^* = d_{n-1} \dots d_{j+1}$. We consider two cases.

Case 1: $\Delta(v) < 2^{j+1}|d^*|$ or $\Delta(v) \geq 2^{j+1}(|d^*| + 1)$.

Due to the induction hypothesis, $\mathrm{DIST}_{j+1}(d^*, v) = 0$, and according to (S4) $\mathrm{DIST}_j(d^* d_j, v) = 0$. But on the other hand we always have

$$2^{j+1}|d^*| \;\leq\; 2^j|d^* d_j| \;\leq\; 2^{j+1}(|d^*| + 1/2),$$

which together with the case assumption implies either

$$\Delta(v) \;<\; 2^{j+1}|d^*| \;\leq\; 2^j|d^* d_j|$$

or

$$\Delta(v) \;\geq\; 2^{j+1}\big(|d^*| + 1\big) \;=\; 2^{j+1}\big(|d^*| + 1/2\big) + 2^j \;\geq\; 2^j\big(|d^* d_j| + 1\big).$$

Hence, invariant $(*)$ is fulfilled.

Case 2: $2^{j+1}|d^*| \leq \Delta(v) < 2^{j+1}(|d^*| + 1)$.

Due to the induction hypothesis, the first term of (S4), namely $\mathrm{DIST}_{j+1}(d_{n-1} \dots d_{j+1}, v)$, equals 1. We have to show that $d_j = 1$ is equivalent to

$$\exists u \;\big(T_j(u, v) \wedge \mathrm{DIST}_{j+1}(d_{n-1} \dots d_{j+1}, u)\big). \tag{1}$$

First of all, $d_j = 1$ is equivalent to $\Delta(v) \geq 2^{j+1}|d^*| + 2^j$. Hence, it suffices to show that (1) is equivalent to $\Delta(v) \geq 2^{j+1}|d^*| + 2^j$.

According to the definition of T_j and the induction hypothesis, (1) implies that there exists a path of length 2^j from a vertex u to v and $2^{j+1}|d^*| \leq \Delta(u) < 2^{j+1}(|d^*| + 1)$. If this is the case, then

$$\Delta(v) \;\geq\; \Delta(u) + 2^j \;\geq\; 2^{j+1}|d^*| + 2^j.$$

Now assume that (1) is not satisfied and consider the longest path p leading to v. If $\Delta(v) < 2^j$, then trivially $\Delta(v) < 2^{j+1}|d^*| + 2^j$ and we are done. Hence assume $\Delta(v) \geq 2^j$. Then there exists a vertex u on p such that a longest path from u to v is entirely on p and has length exactly 2^j. Now we know that

$$\Delta(v) = \Delta(u) + 2^j. \tag{2}$$

Furthermore, $T_j(u,v) = 1$ and thus $\mathrm{DIST}_{j+1}(d^*, u) = 0$ due to the assumption that (1) is not satisfied. Then according to the induction hypothesis, either $\Delta(u) < 2^{j+1}|d^*|$ or $\Delta(u) \geq 2^{j+1}(|d^*|+1)$. But the latter is clearly not possible, because by (2) it would imply $\Delta(v) \geq 2^{j+1}(|d^*|+1) + 2^j$ in contrary to the case assumption. Therefore, again by (2) we can conclude $\Delta(v) < 2^{j+1}|d^*| + 2^j$. ∎

Once we have computed the function DIST, we can use it together with an arbitrary given complete order $\lessdot$ to compute a complete order $\prec$ by letting

$$u \prec v \;\Leftrightarrow\; \exists d_u, d_v : \\ \mathrm{DIST}(d_u, u) \;\wedge\; \mathrm{DIST}(d_v, v) \;\wedge\; \big(|d_u| < |d_v| \vee (|d_u| = |d_v| \wedge u \lessdot v)\big). \tag{S5}$$

By definition, for each vertex v it is $\mathrm{DIST}(d_v, v) = 1$ if and only if $|d_v| = \Delta(v)$. Hence, by (S5) we obtain $u \prec v$ if and only if $\Delta(u) < \Delta(v)$ or if $\Delta(u) = \Delta(v)$ and $u \lessdot v$. Therefore, all pairs of vertices $u, v \in V$ are comparable by $\prec$ (i.e., $u \prec v$ or $v \prec u$). Furthermore, due to the input graph being acyclic and due to the transitivity of $\lessdot$, it can be easily checked that $\prec$ is transitive, too. Finally, if $u \leadsto v$, then a path in G leads from u to v. Hence, $\Delta(u) < \Delta(v)$ (due to G being acyclic) and therefore $u \prec v$. Note also that $\prec$ inherits the antisymmetry of $\leadsto$. All in all, $\prec$ defines a complete order on V respecting $\leadsto$.

The following theorem follows easily from the description above and from simply counting the number of OBDD operations.

Theorem 1 *Let $V = \{0,1\}^n$ and $G = (V, E)$ be an acyclic directed graph represented by* OBDD*s. Applying the* OBDD *operations as described in* (S1)–(S5) *yields an* OBDD *for a relation $\prec$ which defines a complete order on V such that $v \prec w$ for all $v, w \in V$ with $(v,w) \in E$. The number of* OBDD *operations required is $O(\log^2 |V|)$, where each* OBDD *represents a function on at most $4n$ variables.*

Note that we are not restricted to vertex sets V whose cardinality is a power of two. Assume that the input for our algorithm is the characteristic function for a vertex set $V \subseteq \{0, \dots, 2^n - 1\}$ and an edge relation $E \subseteq \{0, \dots, 2^n - 1\} \times \{0, \dots, 2^n - 1\}$. If we are not sure that the edge relation is consistent with V (with consistent we mean $E \subseteq V \times V$), then we can simply use two OBDD synthesis operations to obtain a consistent edge relation E':

$$E'(u, v) \;=\; E(u, v) \wedge V(u) \wedge V(v).$$

We then apply the topological sorting algorithm on the edge relation E'. While the complete order $\prec$ returned by the algorithm is defined on $\{0,1\}^n \times \{0,1\}^n$, its restriction to $V \times V$ is obviously a correct complete order.

Since any (not necessarily reduced) OBDD in n variables has $O(2^n)$ nodes, the theorem shows that the true worst-case runtime of our algorithm is $O(|V|^4 \log^2 |V|)$. Clearly, this is much worse than the $O(|V| + |E|)$ upper bound obtained by a well-known explicit algorithm. On the other hand, if all OBDDs obtained during the execution of the algorithm have a subexponential size, its runtime is sublinear with respect to the number of vertices. In the following sections we show that it is justifiable to hope that this is the case for very structured input graphs.

4 Runtime Analysis for the Grid Graph

We analyze the behavior of the topological sorting algorithm for a grid graph with directed edges. We consider a $2^n \times 2^n$-grid, where all edges are directed from left to right and from bottom to up. The directed grid graph consists of the vertex set $V = \{0,1\}^n \times \{0,1\}^n$ and edge set E, where $\big((x, y), (x', y')\big) \in E$ if and only if either $|x| = |x'|$ and $|y'| - |y| = 1$ or $|y| = |y'|$ and $|x'| - |x| = 1$.

Note that there are some parameters we have not fixed in the description of the algorithm and which will most likely affect its performance. First of all, we have not specified the variable ordering (including the ordering of the auxiliary variables) used for the OBDDs. This is a critical point because the π-OBDD size of a function is very sensitive to the variable ordering π. In the analysis to follow, we assume an interleaved variable ordering, that is a variable ordering where e.g. for a function depending on two vertices u, v, the variable v_i precedes the corresponding variable u_i. Note that in practice, heuristics such as sifting algorithms [12] are used to optimize the variable orderings during the execution of an algorithm, and it can be expected that a good variable ordering is found this way. Secondly, we have to fix the order in which variables are quantified during the algorithm. Here we use the fact that all quantifications are over complete n-bit integers. For the analysis to follow we use the convention that if a quantification is to be computed over an n-bit integer $x = x_{n-1} \dots x_0$, then the order of quantification is from the least significant bit to the most significant bit, i.e. we quantify using $Qxf = Qx_{n-1}Qx_{n-1} \dots Qx_0 f$, where $Q \in \{\forall, \exists\}$.

The idea for proving that the topological sorting algorithm is very efficient for the grid graph is that all functions represented by OBDDs after each step of the algorithm belong to a class of functions which have a small OBDD representation. The functions we consider are compositions of certain threshold and modulo functions, which we define and investigate in the next sections.

4.1 Multivariate Threshold Functions and Modulo Functions

In the following, we denote by $X_{k,n}$ the set of variables x^i_j with $1 \le i \le k$ and $0 \le j < n$. By x^i we denote the vector of n variables $(x^i_{n-1}, \dots, x^i_0)$.

Definition 2

1. *A boolean function $f \in B_{kn}$ defined on the variable set $X_{k,n}$ is called k-*variate threshold function*, if there exist a* threshold $T \in \mathbb{Z}$ *and* weights $w_1, \ldots, w_k \in \mathbb{Z}$ *such that*
$$f(x^1, \ldots, x^k) \ = \ 1 \quad \Leftrightarrow \quad \sum_{i=1}^{k} w_i \cdot \left|x^i\right| \ \geq \ T.$$
The maximum absolute weight *of f is defined as $w(f) := \max\left\{|w_1|, \ldots, |w_k|\right\}$. The set of k-variate threshold functions with maximum absolute weight w defined on the set of variables $X_{k,n}$ is denoted by $\mathbb{T}^w_{k,n}$*

2. *A boolean function $g \in B_{kn}$ defined on the variable set $X_{k,n}$ is called k-*variate modulo M function*, if there exists a constant $C \in \mathbb{Z}$ and $w_1, \ldots, w_k \in \mathbb{Z}$ such that*
$$g(x^1, \ldots, x^k) \ = \ 1 \quad \Leftrightarrow \quad \sum_{i=1}^{k} w_i \cdot \left|x^i\right| \equiv C \pmod{M}.$$
The set of k-variate modulo M functions defined on the set of variables $X_{k,n}$ is denoted by $\mathbb{M}^M_{k,n}$.

Remark 1 *Since all weights as well as the threshold describing f are integers, the term $\sum_{i=1}^{k} w_i \cdot \left|x^i\right| < T$ is equivalent to $\sum_{i=1}^{k} (-w_i) \cdot \left|x^i\right| \geq -T + 1$. Hence, if f is in $\mathbb{T}^w_{k,n}$, then so is $\overline{f}$.*

Definition 3 *Let $f \in B_n$ and $\mathcal{C}$ be a class of functions defined on the variable set X_n. We say that f can be* decomposed *into m functions in $\mathcal{C}$, if there exist a formula F on m variables and $f_1, \ldots, f_m \in \mathcal{C}$ such that $f = F(f_1, \ldots, f_m)$. The set of functions decomposable into m functions in $\mathcal{C}$ is denoted by $D[\mathcal{C}, m]$.*

The main idea in our proof is based on two observations. Firstly, any function decomposable into a constant number of threshold and modulo functions has a small OBDD size. Secondly, all intermediate OBDDs obtained during the execution of the topological sorting algorithm on the directed grid graph represent functions which are decomposable into threshold and modulo functions.

Assume that we have a threshold function $f \in B_{kn}$ defined on the variable set $X_{k,n}$ given by an OBDD G_f. Quantifying over one complete integer, e.g. computing the function $(Qx^i)f$ for some quantifier $Q \in \{\forall, \exists\}$, requires n OBDD quantification operations. Since each quantification operation may yield a quadratic blow-up of the input OBDD, during n quantification steps there may appear OBDDs with exponential size. In this case, the true runtime would be exponential in n. The main trick is to prove that all intermediate functions obtained during the computation of $(Qx^1)f$ are decomposable into threshold and modulo functions, too. Note though, that the order in which the variables are quantified is important.

Let in the following lcm denote the least common multiple.

Lemma 1 *Let $f \in D\left[\mathbb{T}^w_{k,n}, m\right]$, and $Q \in \{\exists, \forall\}$ and $q \in \{1, \dots, k\}$. Then $(Qx^q)f \in D\left[\mathbb{T}^{2w \cdot w^*}_{k-1,n} \cup \mathbb{M}^{w^*}_{k-1,n}, m'\right]$, where $w^* \le \mathrm{lcm}\{1, 2, \dots, w\}$ and $m' = O(2^m w^* m^2)$.*

We prove the lemma at the end of this section. The special case $w = 1$ and $m = O(1)$ is most important for the analysis of the grid graph.

Definition 4 *For any $k \in \mathbb{N}$ we denote by D_k the set of function sequences $(f_n)_{n\in\mathbb{N}}$ such that $\exists m \in \mathbb{N}\, \forall n \in \mathbb{N} : f_n \in D\left[\mathbb{T}^1_{k,n}, m\right]$.*

If we start with $(f_n)_{n\in\mathbb{N}} \in D_k$ (i.e., f_n is decomposable into a constant number of threshold functions with maximum absolute weight 1), then according to Lemma 1 $(Qx^i)f_n$ is decomposable into a constant number of threshold functions with maximum absolute weight 2 and modulo functions in $\mathbb{M}^1_{k-1,n}$. Since the functions in $\mathbb{M}^1_{k-1,n}$ are in fact constant functions (and thus are even threshold functions), we obtain the following corollary.

Corollary 1 *Let $k \in \mathbb{N}$ and let $(f_n)_{n\in\mathbb{N}} \in D_k$ be a sequence of functions. For any $Q \in \{\exists, \forall\}$ and $i \in \{1, \dots, k\}$ it is $(Qx^i)f_n \in D\left[\mathbb{T}^2_{k-1,n}, m\right]$, where $m = O(1)$.*

In order to prove Lemma 1, we need the following proposition.

Proposition 1 *Let $S_i : \{0,1\}^{kn} \to \mathbb{N}$ for $i = 1, 2$ be defined on the variable set $X_{k,n}$ by*

$$S_i(x^1, \ldots, x^k) \; = \; \sum_{j=1}^{k} w_{i,j} \left| x^j \right| + T_i,$$

where $w_{i,j}$ and T_i are integers. Let $M \in \mathbb{N}$ and $f_M \in B_{kn}$ defined by

$$\begin{aligned} &f_M(x^1, \ldots, x^k) = 1 \quad \Leftrightarrow \\ &\qquad\qquad \exists \ell \in \{0, \ldots, 2^n - 1\} : \quad S_1(x^1, \ldots, x^k) \; \le \; \ell \cdot M \; \le \; S_2(x^1, \ldots, x^k). \end{aligned}$$

Then $f_M \in D\Big[\mathbb{T}^w_{k,n} \cup \mathbb{M}^M_{k,n}, 2M + 4\Big]$, where $w = 2 \cdot \max\{w_{i,j}\}$.

Proof: Abusing notation, we write S_i instead of $S_i(x^1, \ldots, x^k)$ and f_M instead of $f_M(x^1, \ldots, x^k)$. If $S_1 < 0$, then $f_M = 1$ if and only if $S_2 \ge 0$, and if $S_2 > M(2^n - 1)$, then $f_M = 1$ if and only if $S_1 \le M(2^n - 1)$. To see this, choose $\ell = 0$ in the case $S_1 < 0 \le S_2$ and $\ell = 2^n - 1$ in the case $S_1 \le M(2^n - 1) < S_2$.

Assume now $S_1 \ge 0$ and $S_2 \le M(2^n - 1)$. There exists $\ell \in \{0, \ldots, 2^n - 1\}$ with $S_1 \le \ell \cdot M \le S_2$ if and only if $S_2 - S_1 \ge S_2 \bmod M$. Hence, under the assumption $S_1 \ge 0$ and $S_2 \le M(2^n - 1)$

$$\begin{aligned} f_M(x^1, \ldots, x^k) = 1 \quad &\Leftrightarrow \quad S_2 - S_1 \; \ge \; (S_2 \bmod M) \\ &\Leftrightarrow \quad \exists 0 \le i < M : \; (S_2 \bmod M = i \; \wedge \; S_2 - S_1 \ge i) \\ &\Leftrightarrow \quad \bigvee_{i=0}^{M-1} (S_2 \bmod M = i \; \wedge \; S_2 - S_1 \ge i) \end{aligned}$$

Altogether, we obtain

$$\begin{aligned} f_M \quad \equiv \quad & (S_1 < 0 \wedge S_2 \ge 0) \; \vee \; (S_1 \le M(2^n - 1) \wedge S_2 > M(2^n - 1)) \; \vee \\ & (S_1 \ge 0 \wedge S_2 \le M(2^n - 1)) \wedge \left(\bigvee_{i=0}^{M-1} (S_2 \bmod M = i \; \wedge \; S_2 - S_1 \ge i) \right) \end{aligned}$$

Now it is obvious that f_M can be decomposed into M modulo functions and $M + 4$ threshold functions (note that we do not need to count the functions $S_1 \ge 0$ and $S_2 \le M(2^n - 1)$ since these are the negations of $S_1 < 0$ and $S_2 > M(2^n - 1)$.) Clearly, the modulo functions $S_2 \bmod M = i$ are in $\mathbb{M}^M_{k,n}$. Since the maximum absolute weight of the threshold functions $S_2 - S_1 \ge i$ is bounded by $w = 2 \cdot \max\{w_{i,j}\}$, all threshold functions are in $\mathbb{T}^w_{k,n}$. Hence, $f_M \in D[\mathbb{T}^w_{k,n} \cup \mathbb{M}^M_{k,m}, \, 2M + 4]$. ■

Proof of Lemma 1: We consider only the case $Q = \exists$; the case $Q = \forall$ then follows right away from the De Morgan rules.

Let w.l.o.g. $q = 1$. Assume that f can be decomposed into m functions $f_1, \ldots, f_m$, i.e., there is a formula F such that $f = F(f_1, \ldots, f_m)$. Let f_i, $1 \leq i \leq m$ be a function with weights $w_{i,1}, \ldots, w_{i,k} \leq w$ and threshold T_i, hence f_i decides whether $\sum_{j=1}^{k} w_{i,j}|x^i| \geq T_i$. Let further

$$w^* \;=\; \mathrm{lcm}\{w_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq k\} \;\leq\; \mathrm{lcm}\{1, \ldots, w\},$$

and for $1 \leq i \leq m$ let $z_i = 1$ if $w_{i,1} = 0$ and $z_i = |w^*/w_{i,1}|$, otherwise. Note that $z_i \in \mathbb{N} \cup \{0\}$. Then f_i is equivalent to the function which decides whether $\sum_{j=1}^{k} z_i \cdot w_{i,j} \geq z_i T_i$. Hence, f_i is a threshold function with a maximum absolute weight of at most $w \cdot z_i \leq w \cdot w^*$, and the weight of x^1 is equal to $z_i \cdot w_{i,1} \in \{-w^*, 0, w^*\}$. This means that f can be decomposed into threshold functions $f_1, \ldots, f_m$ with a maximum absolute weight of at most $w \cdot w^*$ and where the weights of x^1 are in $\{-w^*, 0, w^*\}$. Turning the formula F into its DNF, we can write

$$f \;=\; \bigvee_{i=1}^{2^m} \Big(\bigwedge_{j=1}^{\ell_i} g_{i,j} \Big),$$

where $\ell_i \leq m$ and $g_{i,j}$ is one of the functions $f_1, \ldots, f_m$ or their negations. According to Remark 1 and the discussion above $g_{i,j} \in \mathbb{T}_{k,n}^{w \cdot w^*}$ for all i, j, and the weight of x^1 appearing in $g_{i,j}$ is in $\{-w^*, 0, w^*\}$.

Assume that $g_{i,j}(x^1, \ldots, x^k)$ equals the term $\sum_{t=1}^{k} w_{i,j}^t |x^t| \;\geq\; T_{i,j}$. Then let $S_{i,j}(x^2, \ldots, x^k) = T_{i,j} - \sum_{t=2}^{k} w_{i,j}^t |x^t|$. Abusing notation for the sake of readability, we write in the following $g_{i,j}$ for $g_{i,j}(x^1, \ldots, x^k)$ and $S_{i,j}$ for $S_{i,j}(x^2, \ldots, x^k)$. Obviously, $g_{i,j}$ equals 1 if and only if $w_{i,j}^1 \cdot |x^1| \geq S_{i,j}$. Since $w_{i,j}^1 \in \{-w^*, 0, w^*\}$, we have

$$g_{i,j} = 1 \quad \Leftrightarrow \quad \begin{cases} 0 \geq S_{i,j} & \text{if } w_{i,j}^1 = 0 \\ w^*|x^1| \geq S_{i,j} & \text{if } w_{i,j}^1 = w^* \\ w^*|x^1| \leq -S_{i,j} & \text{if } w_{i,j}^1 = -w^*. \end{cases} \tag{3}$$

Let now for $1 \leq i \leq 2^m$

$$C_i \;:=\; C_i(x^2, \ldots, x^k) \;:=\; (\exists x^1) \bigwedge_{j=1}^{\ell_i} g_{i,j}.$$

Then the function $(\exists x^1)f$ is the disjunction of all C_i, $1 \le i \le m$. Further, let $S_i^0 = \{S_{i,j} \mid w_{i,j}^1 = 0\}$, $S_i^+ = \{S_{i,j} \mid w_{i,j}^1 = w^*\}$, and $S_i^- = \{-S_{i,j} \mid w_{i,j}^1 = -w^*\}$. According to (3),

$$\begin{aligned} C_i &= (\exists x^1)\Big(\big(\bigwedge_{S\in S_i^+} w^*|x^1| \ge S\big) \wedge \big(\bigwedge_{S\in S_i^-} w^*|x^1| \le S\big) \wedge \big(\bigwedge_{S\in S_i^0} -S \ge 0\big)\Big) \\ &= (\exists x^1)\Big(\max\{S \in S_i^+\} \le w^*|x^1| \le \min\{S \in S_i^-\} \wedge \big(\bigwedge_{S\in S_i^0} -S \ge 0\big)\Big) \\ &= \Big(\bigwedge_{\substack{S\in S_i^+\\ S'\in S_i^-}} \exists \ell \in \{0,\dots,2^n-1\} : S \le \ell w^* \le S'\Big) \quad \wedge \quad \Big(\bigwedge_{S\in S_i^0} -S \ge 0\Big). \end{aligned}$$

The last part of this formula $(\wedge_{S\in S_{i,j}^0} - S \ge 0)$ is the conjunction of functions in $\mathbb{T}_{k-1,n}^{w\cdot w^*}$. Furthermore, according to Proposition 1, the first part of this formula is the conjunction of functions in $D\Big[\mathbb{T}_{k-1,n}^{2w\cdot w^*} \cup \mathbb{M}_{k-1,n}^{w^*},\, 2w^*+4\Big]$.

To summarize, C_i can be written as the conjunction of $O(m^2)$ functions in $D\Big[\mathbb{T}_{k-1,n}^{2w\cdot w^*} \cup \mathbb{M}_{k-1,n}^{w^*},\, 2w^*+4\Big]$. Since $(\exists x^1 f)$ is the disjunction of all C_i, $1 \le i \le 2^m$, it can be decomposed into at most $O(w^* 2^m m^2)$ functions in $\mathbb{T}_{k-1,n}^{2w\cdot w^*} \cup \mathbb{M}_{k-1,n}^{w^*}$. ∎

4.2 The OBDD Representation of Threshold and Modulo Functions

We now show that functions which are decomposable into threshold functions and modulo functions have small reduced OBDDs, if the variable ordering is chosen appropriately. Let $\pi_{k,n}$ be the variable ordering which orders the variables in $X_{k,n}$ as follows:

$$x_0^1, x_0^2, \dots, x_0^k, x_1^1, \dots, x_1^k, \dots, x_{n-1}^k.$$

I.e., a $\pi_{k,n}$-OBDD tests all bits of the input integers in an interleaved order with increasing significance of the bits.

An OBDD is called *complete*, if all variables appear on each source-to-sink path. It is well known that the reduced π-OBDD of some boolean function f has at most as many nodes on each level as any OBDD, and thus also as any complete OBDD. (By level we mean a maximal set of nodes labeled by the same variable).

Lemma 2

(a) For any function $f \in \mathbb{T}^w_{k,n}$ there exists a complete $\pi_{k,n}$-OBDD which has at most $4kw+5$ nodes on each level, and thus the $\pi_{k,n}$-OBDD size of f is bounded by $O(k^2nw)$.

(b) For any function $g \in \mathbb{M}^M_{k,n}$ and any variable ordering π there exists a complete π-OBDD which has at most M nodes on each level, and thus the π-OBDD size of g is bounded by $O(knM)$.

The result for threshold functions is a simple generalization of [10, Proposition 4]. Similar results about Modulo Functions appear in literature in several variants.

Proof of Lemma 2: *Part (a):* Let $f \in \mathbb{T}^w_{k,n}$ with weights $w_1, \ldots, w_k$ and threshold T. We describe an algorithm which tests whether $S \geq 0$, where

$$S \;:=\; S(x^1, \ldots, x^k) \;:=\; -T + \sum_{i=0}^{k} w_i |x^i|.$$

The algorithm reads the variables in the order defined by $\pi_{k,n}$. After each variable test, one out of $4kw+5$ possible "states" is stored (in addition to the label of the most recently read variable), and the next state is determined by the outcome of the next variable test and the former state. It is obvious how our algorithm can be transformed into a complete $\pi_{k,n}$-OBDD where each node on a level corresponds to exactly one of the possible states.

Assume that $S_0, \ldots, S_n$ is a representation of S similar to the two's complement. More precisely, let $S_i \in \{0,1\}$ for $0 \leq i \leq n-1$, and $S_n \in \mathbb{Z}$ such that $S = \sum_{i=0}^{n} S_i \cdot 2^i$. Obviously, the sign of S_n equals the sign of S. Our algorithm computes the sign of S_n by the school method of addition as follows. Let $T_0, \ldots, T_n$ be the unique integers with $T_i \in \{0,1\}$ for $0 \leq i \leq n-1$ and $T_n \in \mathbb{Z}$, such that $-T = \sum_{i=0}^{n} T_i \cdot 2^i$. In the ith step of the school method the value S_{i-1} as well as the carry value c_i is computed. Thus, $c_{-1} = 0$ and for $0 \leq i \leq n-1$

$$S_i = \left(c_{i-1} + T_i + \sum_{j=1}^{k} w_j \cdot x^j_i\right) \bmod 2, \quad \text{and}$$

$$c_i = \left\lfloor \left(c_{i-1} + T_i + \sum_{j=1}^{k} w_j \cdot x^j_i\right)/2 \right\rfloor$$

Finally, $S_n = c_n$. It is easy to see that $S = \sum_{i=0}^{n} S_i \cdot 2^i$ as required. Note that it is not necessary for the algorithm to compute S_i for $i \le n-1$ in order to determine $S_n = c_n$.

We describe an algorithm consisting of n steps and computing c_{i-1} in the ith step, $1 \le i \le n$, and the sign of c_n in the nth step. The state the algorithm stores in each step is an integer $Q \in \mathbb{Z}$. Before the ith step, $Q = c_{i-1}$, and thus Q is initialized to 0 when the algorithm starts. During the ith step, the variables $x_{i-1}^0, \dots, x_{i-1}^k$ are tested and Q is updated after each variable test in the obvious way such that after all these variable tests $Q = c_{i-1} + \sum_{j=1}^{k} w_j x_{i-1}^j$. Once this value is computed, $c_i = \lfloor (T_i + Q)/2 \rfloor$ is already uniquely determined by Q because T_i is fixed in advance.

Note that in the last step it is not necessary to compute c_n, but instead the sum $c_{n-1} + \sum_{j=1}^{k} w_j x_{n-1}^j$ already uniquely determines the sign of c_n. Hence, the maximum absolute value Q takes during the algorithm is bounded by

$$\max_{0 \le i \le n} \left\{ |c_{i-1}| + \sum_{j=1}^{k} |w_j| x_{i-1}^j \right\} \;\le\; \max_{0 \le i \le n} \{ |c_{i-1}| + kw \}.$$

Since $c_{-1} = 0$ and for $1 \le i \le n-1$

$$|c_i| \;\le\; |c_{i-1} + T_i + kw|/2 \;\le\; |c_{i-1} + kw + 1|/2,$$

a simple induction shows that $|c_i| \le kw + 1$ for all $0 \le i \le n-1$. Hence, $|Q|$ is bounded by $2(kw+1)$ during all steps of the algorithm. This shows that the sign of c_n (and thus the sign of S) can be computed this way by an algorithm storing after each variable test one out of $4kw + 5$ states.

Part (b): Let $g(x^1, \dots, x^k)$ defined to be 1 if and only if $\sum_{i=1}^{k} w_i |x^i| \equiv C \pmod{M}$. Similar as in the proof of part (a) it suffices to describe an algorithm storing one out of M states after each variable test. This is straightforward – the variable ordering even does not matter. The algorithm stores a state $Q \in \{0, \dots, M-1\}$, which is in the beginning initialized to 0. Upon reading an arbitrary variable x_i^j, the value of Q is replaced by $(Q + w_j \cdot |x_i^j|) \bmod M$. When the last variable is read, the result of $g(x^1, \dots, x^k)$ is uniquely defined by the state Q. Since only M possible states have to be stored, the claimed size for the corresponding OBDD follows. ∎

By the *size* of a formula we denote the number of its leaves (which is one more than the number of its gates if all gates have fan-in 2).

Lemma 3 *Let $f_1, \ldots, f_m \in \mathbb{T}^w_{k,n} \cup \mathbb{M}^M_{k,n}$ be given by reduced $\pi_{k,n}$-OBDDs for f_i, $1 \le i \le m$. Further, let $f = F(f_1, \ldots, f_m)$ for a formula F of size s and let $L = L(k, m, M) = \max\{4kw + 5, M\}$. The minimal $\pi_{k,n}$-OBDD for f has at most $L^s kn$ nodes and can be computed in time and space $O\big((kns)^2 L^s \log(knL)\big)$.*

The lemma follows from the bounds on the OBDD size of threshold and modulo functions given in Lemma 2 and the following lemma which is known by folklore.

Lemma 4 *Let $f_1, f_2 \in B_n$ and let $\otimes$ be a boolean operation. If there exist complete π-OBDDs for f_1 and f_2 which have on level i ($1 \le i \le n$) at most $s_{1,i}$ and $s_{2,i}$ nodes, respectively, then there exists a complete π-OBDD for $f_1 \otimes f_2$ which has on level i at most $s_{1,i} \cdot s_{2,i}$ nodes.*

Proof of Lemma 3: We assume w.l.o.g. that each gate of the formula F has fan-in 2. (Note that the OBDD for the negation $\overline{f_i}$ of one of the input functions can be obtained by simply exchanging the OBDD's 1-sink with its 0-sink.) We first show by induction on s that there exists a complete $\pi_{k,n}$-OBDD for f which has at most L^s nodes on each of its kn levels.

For $s = 1$, the formula F has no gates, i.e., $f \in \mathbb{T}^w_{k,n} \cup \mathbb{M}^M_{k,n}$ is already given as the input OBDD. In this case, the claimed size bound follows directly from Lemma 2.

Let now $s > 1$, i.e., F has at least one gate. Let f_a and f_b be the two functions represented at the inputs of the output gate of F. Then $F = f_a \otimes f_b$ for some binary operation $\otimes \in B_2$, and $f_a = F_a(f_1, \ldots, f_m)$ and $f_b = F_b(f_1, \ldots, f_m)$ for two formulas F_a and F_b of sizes s_a and s_b with $s = s_a + s_b$. By induction hypothesis, the minimal $\pi_{k,n}$-OBDDs G_a and G_b for f_a and f_b have on each level at most L^{s_a} and L^{s_b} nodes, respectively. Hence, $|G_a| \le L^{s_a} kn$ and $|G_b| \le L^{s_b} kn$. According to Lemma 4, there exists a complete OBDD for $f = f_a \otimes f_b$ having at most $L^{s_a} \cdot L^{s_b} = L^s$ nodes on each level.

We now show the claimed time bound. Consider an arbitrary $\otimes$-gate ($\otimes \in B_2$) and let its inputs be sub-formulas for the functions f_a and f_b represented by the OBDDs G_a and G_b. Assume that the sub-formulas for f_a and f_b have a size of s_a and s_b, respectively. Then $s_a + s_b \le s$ and by what we have proven above $|G_a| \le knL^{s_a}$

and $|G_b| \le knL^{s_b}$. Hence, the time required for the binary synthesis of G_a and G_b is bounded by

$$\begin{aligned} O\big(|G_a|\cdot|G_b|\log\left(|G_a|\cdot|G_b|\right)\big) &= O\big((kn)^2L^{s_a+s_b}\cdot\log\left((kn)^2L^{s_a+s_b}\right)\big) \\ &= O\big((kn)^2L^s s\log(knL)\big) \end{aligned}$$

Since the OBDD for the complete formula can be computed with at most s such synthesis operations, the claimed time bound follows ■

We have shown so far, that if a function is given by the $\pi_{k,n}$-OBDDs of threshold and modulo functions into which it can be decomposed, then its $\pi_{k,n}$-OBDD can be computed efficiently. Now we show for functions being decomposable into threshold functions (and no modulo functions), that the quantification over one of its variable blocks $x_0^i,\dots,x_{n-1}^i$, $1 \le i \le k$, can be done efficiently.

Theorem 2 *Let $(f_n)_{n\in\mathbb{N}}$ such that there exist $w, m \in \mathbb{N}$ with $f_n \in D[\mathbb{T}_{k,n}^w, m]$ for all $n \in \mathbb{N}$, and let $Q \in \{\exists,\forall\}$. If f_n is given as a $\pi_{k,n}$-OBDD, then for any $1 \le \ell \le k$ a minimal $\pi_{k,n}$-OBDD for $(Qx^\ell)f_n$ can be computed in time $k^{O(1)}n^3\log n$.*

Proof: Fix $w, m \in \mathbb{N}$ such that $f_n \in D[\mathbb{T}_{k,n}^w, m]$ for all $n \in \mathbb{N}$ and write f instead of f_n. W.l.o.g. we assume $\ell = 1$ and for the sake of readability we write x instead of x^1. We only prove the theorem for the case $Q = \forall$; the proof for $Q = \exists$ works analogously. We can write $(\forall x)f$ as $(\forall x_{n-1}\forall x_{n-2}\dots\forall x_0)f(x^2,\dots,x^k)$. If we apply the OBDD quantification operations to the bits $x_0,\dots,x_{n-1}$ in this order, then after the ith quantification $(0 \le i \le n)$ the resulting OBDD G_i represents the function $g_i = (\forall x_{i-1}\dots\forall x_0)f$ in B_{kn-i}. Since each of the n quantification operations can be done in time $O(|G_i|^2\log|G_i|)$, the total time required is bounded by $\sum_{i=0}^{n-1}|G_i|^2\log|G_i|$. Hence, it suffices to show that G_i has a size of at most $O(nk^{O(1)})$ for all $0 \le i \le n-1$.

Note that g_i does not depend on the variables $x_0,\dots,x_{i-1}$. In the following we introduce n dummy variables $z_0,\dots,z_{n-1}$ and show that g_i can be written as $\big((\forall z_0,\dots,z_{n-1})g_i^*\big)_{|x_0=0,\dots,x_{i-1}=0}$, where g_i^* is a function in $D\Big[\mathbb{T}_{k+1,n}^w, m+1\Big]$. Hence, g_i is obtained from the function $(\forall z_0,\dots,z_{n-1})g_i^*$ by restricting some variables to constants. By Lemma 1, this function is decomposable into a constant number of threshold and modulo functions, and therefore its OBDD size is bounded sufficiently. Note that the variables $z_0, \dots, z_{n-1}$ are merely artifical helper variables, and that none of the functions we "really" deal with (i.e., which are represented by OBDDs) depend on these variables.

Let $f = F(f_1, \dots, f_m)$ for a formula F and $f_1, \dots, f_m \in \mathbb{T}^w_{k,n}$. Since $m = O(1)$, we may assume w.l.o.g. that the size s of F is a constant, too. We introduce n new variables, which we denote by $z_0, \dots, z_{n-1}$. Then we replace the variables x_j with the variables z_j for $0 \le j \le i-1$. This way we obtain

$$\begin{aligned} g_i =\ & (\forall x_{i-1} \dots x_0) f(x_{n-1} \dots x_i x_{i-1} \dots x_0, x^2, \dots, x^k) \\ =\ & (\forall z_{i-1} \dots z_0) f(x_{n-1} \dots x_i z_{i-1} \dots z_0, x^2, \dots, x^k) \\ =\ & (\forall z_{n-1} \dots z_0)\big((z_{n-1}, \dots, z_i) \neq (0, \dots, 0) \ \vee \ f(x_{n-1} \dots x_i z_{i-1} \dots z_0, x^2, \dots, x^k)\big) \\ =\ & (\forall z_{n-1} \dots z_0)\Big(|z| \ge 2^i \ \vee \ f(x_{n-1} \dots x_i z_{i-1} \dots z_0, x^2, \dots, x^k)\Big) \end{aligned} \tag{4}$$

Now consider an arbitrary threshold function f_j, $1 \le j \le m$, i.e.,

$$f_j(x, x^2, \dots, x^k) = 1 \quad \Leftrightarrow \quad w_1|x| + w_2\left|x^2\right| + \dots + w_k\left|x^k\right| \ge T$$

Let $f_j^* \in B_{(k+1)n}$ with

$$f_j^*(z, x, x^2, \dots, x^k) = 1 \quad \Leftrightarrow \quad w_1|z| + w_1\left|x^1\right| + w_2\left|x^2\right| + \dots + w_k\left|x^k\right| \ge T$$

and $f^* = F(f_1^*, \dots, f_m^*)$. Obviously, $f^* \in D[\mathbb{T}^w_{k+1,n}, m]$. If $|z| < 2^i$, then $|x_{n-1} \dots x_i z_{i-1} \dots z_0|$ is the same as $|x_{n-1} \dots x_i 0 \dots 0| + |z|$. Hence, it is easy to conclude from (4) that

$$\begin{aligned} g_i =\ & (\forall z_{n-1} \dots z_0)\Big(|z| \ge 2^i \ \vee \ f^*(z, x_n \dots x_i 0 \dots 0, x^2, \dots, x^k)\Big) \\ =\ & (\forall z_{n-1} \dots z_0)\Big(|z| \ge 2^i \ \vee \ f^*_{|x_{i-1} = \dots = x_0 = 0}(z, x, x^2, \dots, x^k)\Big). \end{aligned}$$

Now let

$$g_i^*(x^1, \dots, x^k) \ = \ |z| \ge 2^i \ \vee \ f^*(z, x^1, x^2, \dots, x^k).$$

Then $g_i^* \in D\Big[\mathbb{T}^w_{k+1,n}, m+1\Big]$ and $g_i = \big((\forall z) g_i^*\big)_{|x_0 = 0, \dots, x_{i-1} = 0}$.

Since $g_i^* \in D\Big[\mathbb{T}^w_{k+1,n}, m+1\Big]$ and k, w, and m are constants, we can conclude from Lemma 1 that $(\forall z) g_i^* \in D\Big[\mathbb{T}^{w'}_{k,n} \cup \mathbb{M}^M_{k,n}, m'\Big]$ for some constants w', M, and m'. Thus, by Lemma 3 the $\pi_{k,n}$-OBDD size of $(\forall z)g_i^*$ is bounded by $O(nk^{O(1)})$. But as we have shown above, the $\pi_{k,n}$-OBDD for g_i can be obtained from the $\pi_{k,n}$-OBDD for $(\forall z)g_i^*$ by simply replacing some variables with the constant 0. Hence, the resulting minimal $\pi_{k,n}$-OBDD for g_i can only be smaller than that for $(\forall z)g_i^*$ and thus its size is also bounded by $O(nk^{O(1)})$. ■

Remark 2 *It is obvious that all the upper bounds in Lemma 3 and Theorem 2 proven for functions decomposable into threshold- and modulo functions hold equivalently for their subfunctions $f_{|\alpha_1\ldots\alpha_i}$, where $\alpha_1\ldots\alpha_i$ is a restriction to arbitrary variables except those being quantified in case of Theorem 2.*

Remark 3 *Let $f \in B_n$ be the function $x_{n-1}\ldots x_0 \mapsto x_i$ for some $0 \le i \le n-1$. Then f is obtained from the function $g \in D\left[\mathbb{T}_{1,n}^1, 2\right]$, where $g(x) = 1 \Leftrightarrow |x| = 2^i$, by replacing all variables except x_i with 0.*

4.3 Runtime Analysis of the Topological Sorting Algorithm for the Grid Graph

In order to show that the topological sorting algorithm runs efficiently on a grid graph, we prove that all functions obtained during the execution of the algorithm can be decomposed into a constant number of threshold and modulo functions, where possibly some variables are fixed by constants.

Since we need only very special cases of the results proven in the former section, we summarize the results for these cases in the following corollary.

Corollary 2 *Fix a constant $k \in \mathbb{N}$ and let $i, j \in \{1, \ldots, k\}$ and $Q, Q' \in \{\exists, \forall\}$. Further, let $(g_n)_{n\in\mathbb{N}} \in D_k$ and $f_n = g_{n|\alpha}$, where α is an assignment of constants to arbitrary variables except to those in $\{x_0^i, \ldots, x_{n-1}^i\}$. If g_n is either given by a reduced $\pi_{k,n}$-OBDD or by the reduced $\pi_{k,n}$-OBDDs for the threshold functions into which it is decomposable, then the reduced $\pi_{k,n}$-OBDDs for $(Qx^i)g_n$, $(Qx^i)f_n$, and $(Qx^iQ'x^j)g_n$ can be computed in time $O(n^3 \log n)$.*

Proof: If g_n is given by a constant number of reduced $\pi_{k,n}$-OBDDs for the threshold functions into which it is decomposable, then according to Lemma 3 the $\pi_{k,n}$-OBDD for g_n can even be computed in time $O(n^2 \log n)$. For the claimed time to compute the $\pi_{k,n}$-OBDD of $(Qx^i)g_n$ apply Theorem 2, for $(Qx^i)f_n$ refer in addition to Remark 2. For $(Qx^iQ'x^j)g_n$ we note that according to Corollary 1 $(Q'x^j)g_n \in D\Big[\mathbb{T}^2_{k-1,n}, m\Big]$, where $m = O(1)$. Hence, Theorem 2 can be applied twice. ∎

Recall that the algorithm consists of the following steps.

1. Computing T_i, $0 \leq i \leq n$:

$$T_0(u,v) = E(u,v). \tag{S1}$$

$$T_{i+1}(u,v) = \exists w : \; T_i(u,w) \wedge T_i(w,v). \tag{S2}$$

2. Computing DIST_j, $0 \leq j \leq n$:

$$\mathrm{DIST}_n(v) := 1 \tag{S3}$$

and for $j = n-1, \ldots, 0$

$$\begin{aligned}\mathrm{DIST}_j(d_{n-1} \ldots d_j, v) &= \mathrm{DIST}_{j+1}(d_{n-1} \ldots d_{j+1}, v) \quad \wedge \\ &\Big(d_j \Leftrightarrow \exists u \; \big(T_j(u,v) \; \wedge \; \mathrm{DIST}_{j+1}(d_{n-1} \ldots d_{j+1}, u)\big)\Big).\end{aligned} \tag{S4}$$

3. Computing the complete order $\prec$:

$$\begin{aligned}&u \prec v \;\Leftrightarrow\; \exists d_u, d_v : \\ &\qquad \mathrm{DIST}(d_u, u) \;\wedge\; \mathrm{DIST}(d_v, v) \;\wedge\; \big(|d_u| < |d_v| \vee (|d_u| = |d_v| \wedge u \lessdot v)\big).\end{aligned} \tag{S5}$$

Whenever we talk in the following about an OBDD for some function sequence in D_k, we assume that the variable ordering is $\pi_{k,n}$. We have to specify the complete order $\lessdot$ for the operations in (S5). A very natural order is the lexicographical order

$$(x^1, y^1) \lessdot (x^2, y^2) \quad \Leftrightarrow \quad |x^1| < |x^2| \;\vee\; \big(|x^1| = |x^2| \wedge |y^1| \leq |y^2|\big).$$

We start with the analysis of the edge relation E. By the definition of the grid graph, $\big((x^1, y^1), (x^2, y^2)\big) \in E$ if and only if

$$\big(|x^2| - |x^1| = 0 \;\wedge\; |y^2| - |y^1| = 1\big) \;\vee\; \big(|y^2| - |y^1| = 0 \;\wedge\; |x^2| - |x^1| = 1\big).$$

Clearly, this function is in D_4.

Now we look at the functions T_i obtained by (S1) and (S2). Recall that $T_i(u, v)$ is defined to be 1 if and only if there exists a path from u to v which has length exactly 2^i. Note also that in the directed grid graph all paths from vertex u to vertex v have the same length. Hence, for the directed grid graph $T_i\big((x^1, y^1), (x^2, y^2)\big) = 1$ if and only if

$$|y^2| \geq |y^1| \quad \wedge \quad |x^2| \geq |x^1| \quad \wedge \quad |x^2| - |x^1| + |y^2| - |y^1| = 2^i.$$

Clearly, this function is in D_4 and thus according to Corollary 2, T_{i+1} can be computed from T_i in time $O(n^3 \log n)$. (Note also that the quantification over one vertex in the grid graph is a quantification over two integers.) Hence, computing $T_1, \ldots, T_n$ requires time $O(n^4 \log n)$ in total.

Next, we analyze the construction of the OBDDs for the functions DIST_j in (S3) and (S4). Recall that for any vertex v and any $d^* = d_{n-1} \ldots d_j$, the function $\mathrm{DIST}_j(d^*, v)$ is true if and only if d^* describes the $n-j$ most significant bits of the bitwise representation of $\Delta(v)$. Let $f_j \in B_{3n}$, $0 \leq j \leq n$, be defined by

$$f_j(d, x, y) \;=\; 1 \quad \Leftrightarrow \quad |d| \;\leq\; |x| + |y| \;<\; |d| + 2^j.$$

Hence, f_j is the conjunction of two functions in $\mathbb{T}^1_{3,n}$. Furthermore, it is easy to see that

$$\mathrm{DIST}_j\big(d_{n-1} \ldots d_j, (x, y)\big) \;=\; f_{|d_{j-1} = \cdots = d_0 = 0}(d, x, y).$$

Therefore, DIST_j is obtained from a function in D_3 by replacing some variables with the constant 0. Note also that $\mathrm{DIST} = \mathrm{DIST}_0$ is in fact in D_3. Moreover, due to the analysis of T_j above, it becomes obvious that $T_j(u, v) \wedge \mathrm{DIST}_{j+1}(d_{n-1} \ldots d_{j+1}, u)$ is a function in D_5, where some variables are replaced with the constant 0. Hence, according to Corollary 2, the OBDD for

$$g_j \;:=\; \exists u : \; T_j(u, v) \wedge \mathrm{DIST}_{j+1}(d_{n-1} \ldots d_{j+1}, u)$$

can be computed in time $O(n^3 \log n)$. The function g_j is obtained from a function in $D\big[\mathbb{T}^8_{3,n} \cup \mathbb{M}^2_{3,n}, O(1)\big]$ by replacing some variables with the constant 0 (apply first Corollary 1 and then Lemma 1). Now it is easy to see that the final two synthesis operations of (S4) required in order to compute DIST_j run in time $O(n^2 \log n)$ (apply Lemma 3 and Remarks 2 and 3.) Hence, the total time for computing DIST_j from DIST_{j-1} is $O(n^3 \log n)$ and $\mathrm{DIST}_{n-1}, \ldots, \mathrm{DIST}_0 = \mathrm{DIST}$ can be computed in total time $O(n^4 \log n)$.

Finally, we have to investigate the computation of the complete order $\prec$ using the operations in (S5). Recall that $\mathrm{DIST} \in D_3$ Hence, if one takes the definition of $\lessdot$ into account, the complete term in (S5) before the first quantification describes a function h in D_4. According to Corollary 2 the function $h' = (\exists d_v \exists d_u)h$ can be computed in time and space $O(n^3 \log n)$.

Summing up the time bounds for all OBDD operations, we have obtained the following result.

Theorem 3 *The* OBDD *algorithm for topological sorting takes time* $O(n^4 \log n)$ *on the directed* $2^n \times 2^n$ *grid graph for an appropriate variable ordering* $\pi_{k,n}$ *and the complete order* $\lessdot$ *as defined above.*

5 Conclusion

The analysis of the symbolic topological sorting algorithm has turned out to be quite involved even for such a simple input instance as the directed grid graph. Nevertheless, the results about the threshold and modulo functions are very general, and we hope that they might as well be applicable to the analysis of other symbolic OBDD algorithms. It would be nice to extend the techniques in such a way that not only single input instances but small graph classes can be handled. An interesting example would be grids where some arbitrary or randomly chosen edges have been removed.

Acknowledgments

The author thanks Daniel Sawitzki and Ingo Wegener for helpful comments and discussions.

References

[1] Y. Breitbart, H. B. Hunt III, and D. J. Rosenkrantz. On the size of binary decision diagrams representing boolean functions. *Theoretical Computer Science*, 145:45–69, 1995.

[2] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[4] H. Cho, G. Hachtel, S.-W. Jeong, B. Plessier, E. Schwarz, and F. Somenzi. ATPG aspects of FSM verification. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 134–137. 1990.

[5] H. Cho, S.-W. Jeong, F. Somenzi, and C. Pixley. Synchronizing sequences and symbolic traversal techniques in test generation. *Journal of Electronic Testing: Theory and Applications*, 4:19–31, 1993.

[6] O. Coudert. Doing two-level logic minimization 100 times faster. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 112–121. 1995.

[7] R. Drechsler. Pseudo-kronecker expressions for symmetric functions. *IEEE Transactions on Computers*, 48:987–990, 1999.

[8] J. Feigenbaum, S. Kannan, M. Y. Vardi, and M. Viswanathan. Complexity of problems on graphs represented as OBDDs. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pp. 216–226. 1998.

[9] 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.

[10] S. Jukna. The graph of integer multiplication is hard for read-k-times networks. Technical Report 95–10, Universität Trier, 1995.

[11] Y. T. Lai, S. B. K. Vrudhula, and M. Pedram. EVBDD-based algorithms for linear integer programming, spectral transformation and function decomposition. *IEEE Transactions on Computer Aided Design*, 13:959–975, 1994.

[12] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 42–47. 1993.

[13] D. Sawitzki. Implizite Algorithmen für Graphprobleme. Diploma thesis, Univ. Dortmund, 2002.

[14] D. Sawitzki. Implicit flow maximization by iterative squaring. In *30th Conference on Current Trends in Theory and Practice of Informatics*, volume 2932 of *Lecture Notes in Computer Science*, pp. 301–313. 2004.

[15] I. Wegener. *Branching Programs and Binary Decision Diagrams - Theory and Applications*. SIAM, 2000.