The Complexity of Problems on Implicitly Represented Inputs*

Daniel Sawitzki**

University of Dortmund, Computer Science 2,
D-44221 Dortmund, Germany
daniel.sawitzki@cs.uni-dortmund.de

Abstract. Highly regular data can be represented succinctly by various kinds of implicit data structures. Many problems in P are known to be hard if their input is given as circuit or Ordered Binary Decision Diagram (OBDD). Nevertheless, in practical areas like CAD and Model Checking, symbolic algorithms using functional operations on OBDD-represented data are well-established. Their theoretical analysis has mostly been restricted to the number of functional operations yet. We show that P-complete problems have no symbolic algorithms using a polylogarithmic number of functional operations, unless P=NC. Moreover, we complement PSPACE-hardness results for problems on OBDD-represented inputs by fixed-parameter intractability results, where the OBDD width serves as the fixed parameter.

1 Introduction

Algorithms on (weighted) graphs G with node set V and edge set $E \subseteq V^2$ typically work on adjacency lists of size $\Theta(|V|+|E|)$ or on adjacency matrices of size $\Theta(|V|^2)$. But in many of today's application areas, graphs occur which cannot be represented explicitly on current computers, or on which even efficient algorithms are not applicable. *Ordered Binary Decision Diagrams* (*OBDDs*) [2], [20] are a data structure for Boolean functions which is proven as succinct representation for structured and regular data.

Having an OBDD representation of a graph, we are interested in solving problems on it without extracting too much explicit information from it. Algorithms whose access to the input graph is mainly restricted to functional operations are called *implicit* or *symbolic* algorithms. In this way, OBDD-based methods are well-established heuristics for special problems in CAD and Model Checking (see, e. g., [10], [20]). These algorithms are observed to be very efficient in practical applications handling large inputs. However, their theoretical analysis has mostly been restricted to the *number* of functional operations up to the present.

Recent research tries to develop theoretical foundations on OBDD-based algorithms. On the one hand, this includes the development of symbolic methods

* Extended version available at http://ls2-www.cs.uni-dortmund.de/~sawitzki/.

** Supported by DFG grant We 1066/10-2.

J. Wiedermann et al. (Eds.): SOFSEM 2006, LNCS 3831, pp. 471–482, 2006.

for fundamental graph problems like topological sorting [21] and the computation of connected components [7], [8], maximum flows [11], [16], and shortest paths [15], [18]. On the other hand, we need more sophisticated analysis techniques to explain the practical success of symbolic algorithms.

In order to represent a directed graph $G = (V, E)$ by an OBDD, we consider its *characteristic Boolean function* χ_G, which maps binary encodings of node pairs to 1 if and only if they correctly reflect G. This representation is known to be not larger than classical ones. Nevertheless, we hope that advantageous properties of G lead to small, that is sublinear OBDD size. Nunkesser and Woelfel [13] show that OBDD representations of various kinds of P_4-sparse and interval graphs can be essentially smaller than explicit representations.

Problems typically get harder when their input is represented implicitly. For circuit representations, this is shown in [1], [6], [14]. Because OBDDs may be exponentially larger than circuits, these results do not directly carry over to problems on OBDD-represented inputs. Feigenbaum et al. [5] prove that the *Graph Accessibility Problem* is PSPACE-complete on OBDD-represented graphs. First efficient upper bounds on time and space of symbolic graph algorithms on special inputs have been presented by Sawitzki [16], [18] and Woelfel [21]. These results rely on restrictions on the complete-OBDD width of occurring OBDDs. The representational power of complete OBDDs with bounded width is discussed in [17].

The design of symbolic graph algorithms often pursues the aim of obtaining polylogarithmic runtime w. r. t. $|V|$ on special input instances. This requires two conditions: A small number of executed OBDD operations *and* small size of *all* occurring OBDDs. We contribute hardness results related to both conditions.

The paper is organized as follows: Section 2 formalizes symbolic algorithms working on the characteristic Boolean function of an input string. This framework enables us to describe a simulation of symbolic algorithms by parallel algorithms in Section 3, which implies that P-complete problems have no symbolic algorithms using a polylogarithmic number of functional operations, unless P=NC. For none of the existing OBDD-based symbolic algorithm analyses so far, a restriction on the input OBDD width suffices to prove efficiency. This would correspond to a fixed-parameter tractable algorithm with the input's OBDD width as parameter. For various fundamental graph problems, such algorithms do not exist unless P=PSPACE, which is shown in the second part of the paper. After foundations on OBDDs in Section 4, we discuss the fixed-parameter tractability of critical operations on OBDDs in Section 5. So we are able to prove implicit versions of several graph problems to be fixed-parameter intractable in Section 6. Finally, Section 7 gives conclusions on the work.

2 A Framework for Symbolic Algorithms

In order to formalize what typically makes a symbolic algorithm, we introduce *Symbolic Random Access Machines*. A classical Random Access Machine (RAM) gets its input as a binary string $I \in \{0,1\}^*$ on a read-only input tape and presents its output O on a write-only output tape.

For $\mathbb{B} := \{0,1\}$, let us denote the ith character of a binary string $x \in \mathbb{B}^n$ by x_i and let $|x| := \sum_{i=0}^{n-1} x_i 2^i$ identify its value. The class of Boolean functions $f\colon \{0,1\}^n \to \{0,1\}$ will be denoted by B_n. We define the *characteristic Boolean function* $\chi_I \in B_n$ of some $I \in \mathbb{B}^N$ by $\chi_I(x) := I_{|x|}$ for $n := \lceil \log_2 N \rceil$, $x \in \mathbb{B}^n$, and $I_N, \ldots, I_{2^n-1} := 0$.

Definition 1. *A* Symbolic Random Access Machine (SRAM) $\mathcal{M}$ *corresponds to a classical RAM without input and output tapes. In addition to its working registers $R = R_0, R_1, \ldots$ (containing integers), it has symbolic registers $S = S_0, S_1, \ldots$ which contain Boolean functions initialized to the zero function. The input I is presented to $\mathcal{M}$ as characteristic Boolean function χ_I in S_0. Finally, $\mathcal{M}$ presents its output O as χ_O in S_0.*

Besides the usual RAM instructions, an SRAM $\mathcal{M}$ offers the following operations on registers (resp. functions) S_i and S_j:

- *Request the number n of Boolean variables all functions S_i are defined on (initially $\lceil \log_2 N \rceil$).*
- *Increase the variable count n by some amount $\Delta n \in \mathbb{N}$.*
- *Set $S_i := S_j$.*
- *Evaluate S_i due to some variable assignment $a \in \mathbb{B}^n$.*
- *Compute the negation $\overline{S_i}$.*
- *Compute $S_i \otimes S_j$ for some binary infix operator $\otimes \in B_2$.*
- *Replace a variable x_k for S_i by a constant $c \in \mathbb{B}$.*
- *Swap two variables x_k, x_ℓ for S_i, i. e., $S'(x_0, \ldots, x_k, \ldots, x_\ell, \ldots, x_n) := S_i(x_0, \ldots, x_{k-1}, x_\ell, x_{k+1}, \ldots, x_{\ell-1}, x_k, x_{\ell+1}, \ldots, x_{n-1})$.*
- *Decide whether $S_i = S_j$.*
- *Compute the number $|S_i^{-1}(1)|$ of satisfying variable assignments.*
- *Write all satisfying variable assignments $S_i^{-1}(1)$ into R.*
- *Compute the subset of $\{x_0, \ldots, x_{n-1}\}$ on which S_i essentially depends on.*
- *Set S_0 to some function $f \in B_n$ represented in R due to some standard encoding (e. g., as polynomial, circuit, or OBDD). The encoding must enable to be evaluated in linear sequential time w. r. t. its length.*

Each operation costs one unit of time.

(The last operation enables to create fundamental building block functions having some short description. Quantifications and variable replacements by functions can be implemented by a constant number of negations and binary operators.)

This model is independent of a concrete data structure for Boolean functions; it is chosen with the aim of showing *lower bounds* on the number of functional operations. It covers what is considered as a symbolic resp. implicit algorithm in most of the literature. Depending on the type of input data (e. g., graphs) the definition of χ_I may vary; due to its interchangeability in this context, this does not affect our results.

3 Parallel Simulation of Symbolic Algorithms

It is known from P-completeness theory that P-complete, FP-complete, resp. quasi-P-complete problems cannot be solved by PRAMS in parallel time $\mathcal{O}(\log^k N)$ using $\mathcal{O}(N^k)$ processors for problem size N and some constant k, unless P=NC (see, e.g., [9]). Sieling and Wegener [19] present NC-algorithms for all important OBDD operations. We use a simpler approach which suits better for our purpose to prove the first main result of this paper.

Theorem 1. *An SRAM $\mathcal{M}$ using time $t_{\mathcal{M}}(N)$ and at most $k \log N$ Boolean variables on implicitly represented inputs $I \in \mathbb{B}^N$ can be simulated by a CREW-PRAM $\mathcal{M}'$ in parallel time $\mathcal{O}((t_{\mathcal{M}}(N))^2 \cdot \log^2 N))$ using $\mathcal{O}(N^k)$ processors working on the explicit representation of I.*

Proof. Each assignment $a \in \mathbb{B}^n$ of the $n \leq k \log N$ Boolean variables the functions of $\mathcal{M}$ can be defined on at any point in time is handled by its own processor P_a which locally saves the value $S_i(a)$ for all symbolic registers S_i used so far. Hence, $2^n = \mathcal{O}(N^k)$ processors are used. At the beginning, P_a reads cell $|a|$ on the input tape and sets $S_0(a)$ accordingly. Common RAM instructions are executed only on P_0. Symbolic operations are simulated in parallel time $\mathcal{O}(t_{\mathcal{M}}(N) \cdot \log^2 N)$ each (proved in the paper's extended version). Finally, S_0 contains χ_O and each processor P_a writes $S_0(a)$ into position $|a|$ on the output tape. □

Corollary 1. *Unless P=NC, (strongly) P-complete, FP-complete, and quasi-P-complete problems cannot be solved by SRAMs in (pseudo-)polylogarithmic time $\mathcal{O}(\log^k(N))$ ($\mathcal{O}(\log^k(N) \cdot \log^k(M))$) using at most $k \log N$ Boolean variables, where N is the input size, M is the maximum magnitude of all numbers in the input, and k is constant.*

We briefly add an inapproximability result. Let $\mathcal{A}$ be a strongly quasi-P-complete integer-valued combinatorial maximization problem whose optimal solution value is polynomially bounded both in the input size N and the input's largest number M. Analog to Theorem 10.3.4 in [9] it follows:

Proposition 1. *If $\mathcal{A}$ has a fully polynomial symbolic approximation scheme, it can be solved by an SRAM in pseudopolylogarithmic time $\mathcal{O}(\log^k(N) \cdot \log^k(M))$ using $\mathcal{O}(\log N)$ Boolean variables, k constant.*

Corollary 2. *$\mathcal{A}$ has no fully polynomial symbolic approximation scheme using $\mathcal{O}(\log N)$ Boolean variables, unless P=NC.*

We have proved that none of the many P-complete problems can be solved by symbolic algorithms using a polylogarithmic number of functional operations and $\mathcal{O}(\log N)$ variables, unless P=NC. All existing symbolic methods known to the author use less than $10 \log_2 N$ variables, which is a usual restriction to keep concrete data structures small. In particular, there is neither an NC algorithm nor a P-completeness proof for the unit capacity maximum flow problem yet [9], which gives a hint why not even the best known symbolic methods [16] for this problem can guarantee polylogarithmic behavior on all the instances.

In the remainder of the paper, we will consider the complexity of problems on OBDD-represented inputs, which makes it necessary to give some foundations on this well-established data structure. Hence, the terms "implicit", "symbolic", and "OBDD-based" will be used interchangeable.

4 Ordered Binary Decision Diagrams

A Boolean function $f \in B_n$ defined on variables $x_0, \ldots, x_{n-1}$ can be represented by an *Ordered Binary Decision Diagram (OBDD)* [2]. An OBDD $\mathcal{G}$ is a directed acyclic graph consisting of *internal nodes* and *sink nodes*. Each internal node is labeled with a Boolean variable x_i, while each sink node is labeled with a Boolean constant. Each internal node is left by two edges one labeled 0 and the other 1. A *function pointer* p marks a special node that represents f. Moreover, a permutation $\pi \in \Sigma_n$ called *variable order* must be respected by the internal nodes' labels on every path from p to a sink. For a given variable assignment $a \in \mathbb{B}^n$, we compute the function value $f(a)$ by traversing $\mathcal{G}$ from p to a sink labeled with $f(a)$ while leaving each node labeled with x_i via its a_i-edge.

An OBDD with variable order π is called π-OBDD. The minimal-size π-OBDD for a function $f \in B_n$ is known to be canonical and will be denoted by π-OBDD$[f]$. Its size size(π-OBDD$[f]$) is measured by the number of its nodes. We adopt the usual assumption that all OBDDs occurring in symbolic algorithms have minimal size, since all essential OBDD operations produce minimized diagrams. On the other hand, finding an optimal variable order leading to the minimum size OBDD for a given function is known to be NP-hard. Independent of π it is size(π-OBDD$[f]$) $\leq \big(2 + o(1)\big)2^n/n$ for any $f \in B_n$.

Efficient Algorithms on OBDDs. OBDDs offer algorithms (called *OBDD operations* in the following) for nearly all the symbolic operations of Definition 1, which are efficient w. r. t. the size of involved OBDDs. The satisfiability of f can be decided in time $\mathcal{O}(1)$. The negation $\overline{f}$, the replacement of a variable x_i by some constant c (i. e., $f_{|x_i=c}$), and computing $|f^{-1}(1)|$ are possible in time $\mathcal{O}\big(\text{size}(\pi\text{-OBDD}[f])\big)$. The set $f^{-1}(1)$ of f's minterms can be obtained in time $\mathcal{O}\big(n \cdot |f^{-1}(1)|\big)$. Whether two functions f and g are equivalent (i. e., $f = g$) can be decided in time $\mathcal{O}\big(\text{size}(\pi\text{-OBDD}[f]) + \text{size}(\pi\text{-OBDD}[g])\big)$. The most important OBDD operation is the *binary synthesis* $f \otimes g$ for $f, g \in B_n$, $\otimes \in B_2$ (e. g., $\wedge$, $\vee$), which corresponds to the binary operator of SRAMs; in general, it produces the result π-OBDD$[f \otimes g]$ in time and space $\mathcal{O}\big(\text{size}(\pi\text{-OBDD}[f]) \cdot \text{size}(\pi\text{-OBDD}[g])\big)$. The synthesis is also used to implement *quantifications* $(\mathcal{Q}x_i)f$ for $\mathcal{Q} \in \{\exists, \forall\}$. Hence, computing π-OBDD$\big[(\mathcal{Q}x_i)f\big]$ takes time $\mathcal{O}\big(\text{size}^2(\pi\text{-OBDD}[f])\big)$ in general.

Nevertheless, a sequence of only n synthesis operations may cause an exponential blow-up on OBDD sizes, in general. The book of Wegener [20] gives a comprehensive survey on different types of Binary Decision Diagrams.

Representing Graphs by OBDDs. In Section 2, we defined characteristic functions χ_I for inputs I of general problems. The next sections' results will mostly be connected to decision problems on graphs $G = (V, E)$ with N nodes $v_0, \ldots, v_{N-1}$. Hence, we adapt the definition of χ_I to $\chi_G(x, y) = 1 :\Leftrightarrow (|x|, |y| < N) \wedge (v_{|x|}, v_{|y|}) \in E$, where $x, y \in \mathbb{B}^n$ and $n := \lceil \log_2 N \rceil$, which is common in the literature. Undirected edges are represented by symmetric directed ones. It can be easily seen that this is equivalent to the definition of χ_I in Section 2 if I is the row-wise adjacency matrix.

Symbolic graph algorithms typically use intermediate functions defined on a constant number $k > 2$ of variable vectors $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$ mostly interpreted as node numbers or components of them. Therefore, reordering a function's arguments becomes an important operation:

Definition 2. *Let $\rho \in \Sigma_k$ and $f \in B_{kn}$ be defined on variable vectors $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$. The* argument reordering *$\mathcal{R}_\rho(f) \in B_{kn}$ w. r. t. ρ is defined by $\mathcal{R}_\rho(f)\big(x^{(1)}, \ldots, x^{(k)}\big) = f\big(x^{(\rho(1))}, \ldots, x^{(\rho(k))}\big)$.*

In order to enable efficient argument reorderings (see Lemma 3), it is common to use *k-interleaved* variable orders, denoted by $\pi^\tau_{k,n}$, which read bits of same significance en bloc:

$$\pi^\tau_{k,n} := \left(x^{(1)}_{\tau(0)}, \ldots, x^{(k)}_{\tau(0)}, x^{(1)}_{\tau(1)}, \ldots, x^{(k)}_{\tau(1)}, \ldots\ldots, x^{(k)}_{\tau(n-1)}\right) \quad ,$$

where τ is the local order of every $x^{(1)}, \ldots, x^{(k)}$. The order $\pi^{\mathrm{id}}_{k,n}$ is called *natural* in the following.

5 Fixed-Parameter Tractable OBDD Operations

Feigenbaum et al. have proved some fundamental graph problems to be hard if the input is represented as OBDD. That is, there is no hope of beating classical algorithms on explicit inputs in general. However, symbolic methods for maximum flows [16], shortest paths [18], and topological sortings [21] could be proved to have polylogarithmic runtime when the input graphs are of special structure. The analysis technique relies on the *complete-OBDD width* of Boolean functions:

Definition 3. *An OBDD for $f \in B_n$ is called* complete *if every path from its function pointer to a sink has length n.*

That is, complete OBDDs are not allowed to skip variable tests. The minimal-size complete π-OBDD for $f \in B_n$ is also known to be canonical [20] and will be denoted by $\pi\text{-OBDD}_\mathrm{c}[f]$ in the following.

Definition 4. *The* complete-OBDD width *of a function $f \in B_n$ w. r. t. a variable order $\pi \in \Sigma_n$ is the maximum number of OBDD nodes labeled with the same variable in $\pi\text{-OBDD}_\mathrm{c}[f]$.*

Clearly, it is $\mathrm{size}(\pi\text{-OBDD}[f]) \le \mathrm{size}(\pi\text{-OBDD}_\mathrm{c}[f]) = \mathcal{O}(nw)$ for any $f \in B_n$ with complete-OBDD width w and variable order π. On the other hand, it is $\mathrm{size}(\pi\text{-OBDD}_\mathrm{c}[f]) \le n \cdot \mathrm{size}(\pi\text{-OBDD}[f])$ (see, e. g., [20]).

We now briefly introduce the concept of fixed-parameter tractability. For a comprehensive introduction, the reader is referred to the book of Downey and Fellows [4].

Definition 5.

(1) Let Γ be a finite alphabet. A parameterized problem *Π is a map $\Pi\colon \Gamma^* \times \mathbb{N} \to \Gamma^*$. The second component k of a problem instance $(I,k) \in \Gamma^* \times \mathbb{N}$ is called the* problem parameter*.*
(2) An algorithm for a parameterized problem Π is called fixed-parameter tractable (FPT)*, if it solves Π in time $\mathcal{O}(N^\alpha \cdot \beta(k))$ on any instance $(I,k) \in \Gamma^N \times \mathbb{N}$ for a constant α and an arbitrary function $\beta\colon \mathbb{N} \to \mathbb{N}$.*

That is, Π can be solved in polynomial time for fixed k. Recent symbolic algorithm analyses [16], [18], [21] use that critical OBDD operations which may cause OBDDs to grow are fixed-parameter tractable, where the complete-OBDD width serves as the fixed parameter.

Let $f^{(1)}, f^{(2)} \in B_n$ be defined on variables $x_0, \ldots, x_{n-1}$; assume $f^{(1)}$ resp. $f^{(2)}$ has complete-OBDD width w_1 resp. w_2 w. r. t. some variable order $\pi \in \Sigma_n$.

Lemma 1 (Binary synthesis). *The binary synthesis result π-OBDD$[f^{(1)} \otimes f^{(2)}]$, $\otimes \in B_2$, is computed in time $\mathcal{O}\big(nw_1w_2 \log(nw_1w_2)\big)$ and space $\mathcal{O}\big(nw_1w_2\big)$ and has a complete-OBDD width of at most w_1w_2.*

Often, symbolic algorithms contain quantification sequences over $\Omega(n)$ variables of some variable vector (e. g., a graph node encoding). While each single one is efficient, a sequence of length $\Omega(n)$ may cause an exponential blow-up in general. Hence, we consider the properties of quantifications over a subset of variables.

Lemma 2 (Quantification). *Let $X \subseteq \big\{x_0, \ldots, x_{n-1}\big\}$. The quantification result π-OBDD$\big[(\mathcal{Q}X)f^{(1)}\big]$, $\mathcal{Q} \in \{\exists, \forall\}$, is computed in time $\mathcal{O}\big(|X|n2^{2w_1} \log(n2^{2w_1})\big)$ and space $\mathcal{O}\big(|X|n2^{2w_1}\big)$ and has a complete-OBDD width of at most 2^{w_1}.*

Let $f^{(3)} \in B_{kn}$ be defined on variable vectors $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$; assume $f^{(3)}$ has complete-OBDD width w_3 w. r. t. a variable order $\pi^\tau_{k,n}$, $\tau \in \Sigma_n$. Let $\rho \in \Sigma_k$.

Lemma 3 (Argument reordering). *The argument reordering result $\mathcal{R}_\rho(f^{(3)})$ of $f^{(3)}$ w. r. t. ρ is computed in time $\mathcal{O}\big(nw_3k3^k\big)$ and space $\mathcal{O}\big(nw_33^k\big)$ and has a complete-OBDD width of at most w_33^k.*

(Proofs of Lemmas 1–3 can be found in the paper's extended version.)

As a final building block we introduce *multivariate threshold functions*, which are used to implement weighted comparisons.

Definition 6 ([21]). *Let $f \in B_{kn}$ be defined on variable vectors $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$. Function f is called k-*variate threshold function *iff there are $W \in \mathbb{N}$, $T \in \mathbb{Z}$, and $\alpha_1, \ldots, \alpha_k \in \{-W, \ldots, W\}$ such that*

$$f\left(x^{(1)},\ldots,x^{(k)}\right)=\left(\sum_{i=1}^{k}\alpha_i\cdot\left|x^{(i)}\right|\geq T\right)\ .$$

The corresponding class of functions is denoted by $\mathbb{T}^W_{k,n}$.

Clearly, each of the relations $>$, $\leq$, $<$, and $=$ can be composed of $\mathcal{O}(1)$ multivariate threshold functions.

Lemma 4 ([21]). *Functions* $f\in\mathbb{T}^W_{k,n}$ *have complete OBDDs of width* $\mathcal{O}(k^2W)$ *using the natural variable order* $\pi^{\text{id}}_{k,n}$.

Having considered all critical OBDD operations which may enlarge their operands, Lemmas 1–4 imply a general result on the fixed-parameter tractability of bounded sequences of operations.

Theorem 2. *Let* $f\in B_{kn}$ *be defined on variable vectors* $x^{(1)},\ldots,x^{(k)}\in\mathbb{B}^n$ *for a constant* k. *Assume* f *has complete-OBDD width* w *w. r. t. the variable order* $\pi^{\text{id}}_{k,n}$. *Let* $\mathcal{S}$ *be a sequence of* $\mathcal{O}(1)$

- *operations as introduced in Section 4 and*
- *quantifications over variable subsets* $X\in\mathbb{B}^n$

applied on f, *functions from* $\mathbb{T}^{\mathcal{O}(1)}_{k,n}$, *and intermediate results generated by the current prefix of* $\mathcal{S}$.

Each function generated by $\mathcal{S}$ *has a complete-OBDD width of at most* $\beta(w)$ *w. r. t.* $\pi^{\text{id}}_{k,n}$ *for some appropriate function* $\beta\colon\mathbb{N}\to\mathbb{N}$. *So* $\mathcal{S}$ *can be implemented as an FPT algorithm on* $\pi^{\text{id}}_{k,n}$-OBDD$[f]$ *with parameter* w, *runtime* $\mathcal{O}(n\gamma(w)\log(n))$, *and space* $\mathcal{O}(n\gamma(w))$ *for some appropriate function* $\gamma\colon\mathbb{N}\to\mathbb{N}$.

Using this result it is possible to prove that some OBDD-based graph algorithms have polylogarithmic runtime w. r. t. N on special instances [16], [18], [21]. Nevertheless, for none of these analyses it is sufficient to restrict only the input's complete-OBDD width; for example, the symbolic shortest paths algorithm in [18] requires also the output to have constant complete-OBDD width. This motivates the question if there are any FPT algorithms for fundamental graph problems whose parameter is associated solely to the input OBDD.

Starting from a PSPACE-hardness result in [5–Theorem 16], we show in the next section that such algorithms do not exist for some basic graph problems, unless P=PSPACE. This will incorporate FPT reductions build upon Theorem 2 which assure that the fixed parameter grows independently of N.

6 Fixed-Parameter Intractability Results

The *Graph Accessibility Problem (GAP)* is defined as follows: Given a directed graph $G=(V,E)$, decide whether there is a directed path from some source $s\in V$ to some terminal $t\in V$. Due to Theorem 16 in [5], the GAP is PSPACE-complete if G is represented by an OBDD for χ_G.

The reduction generates an OBDD representing the configuration graph $G_{\mathcal{M}}$ of a polynomially space bounded Turing machine $\mathcal{M}$ with some input $I \in \mathbb{B}^*$. The OBDD for $\chi_{G_{\mathcal{M}}}$ checks for each local pair $(X, Y, Z), (X', Y', Z')$ of three consecutive tape positions of the configuration encodings if they are consistent with a computation step. From the construction in [5] it directly follows that the complete-OBDD width of $\chi_{G_{\mathcal{M}}}$ w. r. t. the natural 2-interleaved variable order $\pi^{\mathrm{id}}_{2,p(|I|)}$ is constant (i. e., independent of $|I|$), where $p(|I|)$ is a polynomial number of Boolean variables used to encode one configuration. Hence, an FPT algorithm for GAP on OBDDs would be able to decide in polynomial time w. r. t. $|I|$ if there is a path between the start and accepting configuration – we have our first fixed-parameter intractability result:

Corollary 3 (from Theorem 16 in [5]). *The GAP on OBDD-represented graphs has no FPT algorithm with the fixed parameter being the input's complete-OBDD width, unless P=PSPACE.*

(In the following, we always assume that the fixed-parameter is the input's complete-OBDD width.)

In [18], the *All-Pairs Shortest-Paths Problem (APSPP)* on OBDD-represented graphs is investigated assuming a canonical generalization to graphs with edge weights $c\colon E \to \mathbb{N}$ by $\chi_G(x, y, a) = 1 :\Leftrightarrow c(v_{|x|}, v_{|y|}) = |a|$. An FPT algorithm is presented whose fixed parameter depends also on the output's complete-OBDD width. This additional condition is necessary (unless P=PSPACE) because the GAP can be trivially reduced to a shortest path problem. Similarly easy, the GAP can be reduced to the *Maximum Flow Problem*.

Proposition 2. *Neither the APSPP nor the Maximum Flow Problem on OBDD-represented graphs has an FPT algorithm, unless P=PSPACE.*

Analog to Theorem 3.2(1) in [3], the result $G_{\mathcal{M}}$ generated in the PSPACE-hardness proof for GAP can be modified to three fundamental problems on undirected graphs: *Acyclicity*, *Connectivity*, and the GAP in undirected planar graphs, *UPGAP*. In doing so, the OBDD width is not essentially enlarged (proved in the paper's extended version).

Theorem 3. *Acyclicity, Connectivity, and the UPGAP have no FPT algorithms on OBDD-represented graphs with 2-interleaved natural variable order, unless P=PSPACE.*

Last but not least, we transfer a selection of reductions from [1], [3], [12] to symbolic OBDD-based reductions which satisfy the preconditions of Theorem 2 and, hence, are transitive FPT reductions (see, e. g., [4–Definition 9.3]). We write $\mathcal{A} \leq_{\text{S-FPT}} \mathcal{B}$ if such a reduction exists for decision problems $\mathcal{A}$ and $\mathcal{B}$.

Theorem 4.

(1) Connectivity $\leq_{\text{S-FPT}}$ *Eulerian-Cycle,*
(2) UPGAP $\leq_{\text{S-FPT}}$ *Bipartiteness,*
(3) UPGAP $\leq_{\text{S-FPT}}$ *Planarity.*

Proof. We describe reductions from $\chi_G \in B_{2n}$ to $\chi_{G'}$ for $G = (V, E)$, $V = \{v_0, \ldots, v_{N-1}\}$, $N = 2^n$, and $G' = (V', E')$.

Part (1): We set $V' := V \cup \{u_{ij} \mid 0 \leq i < j < N\} \cup \{a_i, b_i \mid 0 \leq i < N\}$. E' contains E, $\{v_i, a_i\}$, $\{a_i, b_i\}$, and $\{b_i, v_i\}$ for all i, and $\{v_i, u_{ij}\}$, $\{u_{ij}, v_j\}$ iff $\{v_i, v_j\} \in E$. Note that all nodes in V' have even degree and G' is connected iff G is connected. Hence, G' has an Eulerian cycle iff G is connected.

We define $\chi_{G'}$ on $4(n+1)$ variables with order $\pi^{\text{id}}_{4,n+1}$. A node number $x \in \mathbb{B}^{2(n+1)}$ consists of two concatenated variable vectors of length $n+1$ each. Bits $x_{n-1} \ldots x_0$ encode the index i, bits $x_{2n} \ldots x_{n+1}$ encode the index j for nodes $u_{i,j}$, and the remaining bits x_n and x_{2n+1} encode the node type (i. e., v, u, a, or b). We denote these three components of a node number x by $i(x)$, $j(x)$, resp. $T(x)$ and define

$$\begin{aligned}\chi_{G'}(x, y) := & \big[(T(x) = T(y) = v) \wedge \chi_G(i(x), i(y))\big] \\ & \vee \big[(T(x) = v) \wedge (T(y) = a) \wedge (i(x) = i(y))\big] \\ & \vee \big[(T(x) = a) \wedge (T(y) = b) \wedge (i(x) = i(y))\big] \\ & \vee \big[(T(x) = b) \wedge (T(y) = v) \wedge (i(x) = i(y))\big] \\ & \vee \big[(T(x) = v) \wedge (T(y) = u) \wedge (i(x) = i(y)) \wedge \chi_G(i(y), j(y))\big] \\ & \vee \big[(T(x) = u) \wedge (T(y) = v) \wedge (j(x) = j(y)) \wedge \chi_G(i(x), j(x))\big] \ ,\end{aligned}$$

where tests $T(x) = v, u, a, b$ check x_n and x_{2n+1} and ensure $|j(x)| = 0$ for $T(x) \neq u$.

Part (2): We set $V' := (V \cup E) \times \{1, 2\} \cup \{w\}$. E' contains edges $\big\{(v, r), (e, \ell)\big\}$ for $e \in E$, $v \in V \cap e$, and $r = \ell$. Moreover, E' contains $\big\{(s, 1), (s, 2)\big\}$, $\big\{(t, 1), w\big\}$, and $\big\{(t, 2), w\big\}$ for source and terminal $s, t \in V$. G' contains an odd cycle (i. e., is not bipartite) iff G contains a path between s and t.

We define $\chi_{G'}$ on $4(n+2)$ variables with order $\pi^{\text{id}}_{4,n+2}$. A node number $x \in \mathbb{B}^{2(n+2)}$ consists of two concatenated variable vectors of length $n+2$ each. The additional bits x_n, x_{n+1}, x_{2n+2}, and x_{2n+3} are used to encode the node type (i. e., v, e, or w) and the copy index (i. e., 1 or 2). We denote $x_{n-1} \ldots x_0$ by $i(x)$, $x_{2n+1} \ldots x_{n+2}$ by $j(x)$, the type by $T(x) \in \{v, e, w\}$, and the copy index by $c(x) \in \{1, 2\}$.

$$\begin{aligned}\chi_{G'}(x, y) := & \\ & \big[(T(x) = v) \wedge (T(y) = e) \wedge (i(x) = i(y)) \wedge (c(x) = c(y)) \wedge \chi_G(i(y), j(y))\big] \\ & \vee \big[(T(x) = e) \wedge (T(y) = v) \wedge (j(x) = j(y)) \wedge (c(x) = c(y)) \wedge \chi_G(i(x), j(x))\big] \\ & \vee \big[(T(x) = T(y) = v) \wedge (v_{|i(x)|} = v_{|i(y)|} = s) \wedge (c(x) \neq c(y))\big] \\ & \vee \big[(T(x) = v) \wedge (T(y) = w) \wedge (v_{|i(x)|} = t)\big] \ ,\end{aligned}$$

where tests against $T(x)$ and $c(x)$ check the additional bits x_n, x_{n+1}, x_{2n+2}, and x_{2n+3} and ensure $|j(x)| = 0$ for $T(x) = v$ as well as $|i(x)|, |j(x)| = 0$ for $T(x) = w$.

Part (3): We set $V' := V \cup \{w_1, w_2, w_3\}$ and define $w_4 := s$ and $w_5 := t$. E' is obtained by adding the edges of the complete graph on $w_1, \ldots, w_5$ to E except of the edge $\{w_4, w_5\}$. Because G is planar, G' is planar iff there is no path between $s = w_4$ and $t = w_5$. Now the definition of $\chi_{G'}$ in terms of binary operators and comparisons is straightforward and left to the reader.

Final thoughts: In order to obtain an undirected graph G', we set $\chi_{G'}(x, y) := \chi_{G'}(x, y) \vee \chi_{G'}(y, x)$. Additional singletons appearing due to the node encoding do not affect any of the three considered graph properties. We have seen that $\chi_{G'}$ can be expressed in terms of a constant number of disjunctions, conjunctions, negations, and argument reorderings applied to the original χ_G, multivariate threshold functions from $\mathbb{T}^{\mathcal{O}(1)}_{\mathcal{O}(1),\mathcal{O}(n)}$, and intermediate results. Due to Theorem 2, all three reductions can be implemented as an OBDD-based FPT algorithm on the $\pi^{\text{id}}_{2,n}$-OBDD for χ_G. □

Because Theorem 3 satisfies the preconditions on the variable order of Theorem 2, we conclude:

Corollary 4. *None of the problems Bipartiteness, Eulerian-Cycle, and Planarity on OBDD-represented graphs has an FPT algorithm, unless P=PSPACE.*

In contrast to this paper's exemplary applications of the symbolic FTP reduction technique, more sophisticated reductions (e. g., to the *Bipartite Perfect Matching Problem* [3]) require quantifications and more complex multivariate threshold functions.

7 Conclusions

The complexity of problems on implicitly represented inputs has been considered from two different points of view: First, the number of Boolean operations as a lower bound on the over-all runtime of typical symbolic algorithms. Unless P=NC, no P-complete problem can be solved by $\mathcal{O}(\log^k N)$ operations on functions defined on $\mathcal{O}(\log N)$ variables.

Then, we turned to lower bounds on the concrete over-all runtime of OBDD-based graph algorithms. While the hardness of some basic problems in this scenario was already known, we showed that even the restriction to inputs with constant complete-OBDD width does not yield polylogarithmic algorithms w. r. t. $|V|$, unless P=PSPACE. While applied to a selection of fundamental problems yet, the technique of symbolic FPT reductions can be used for various further problems on OBDD-represented inputs by substituting existing constant depth reductions and projections used for circuit representations (which are more powerful in general, see [20–Section 4.12]).

We conclude that symbolic resp. OBDD-based algorithms, though very successful in practical applications, have quite limited capabilities on many polynomially solvable problems, even for strongly restricted instances.

Acknowledgments. Thanks to Detlef Sieling and Ingo Wegener for proofreading and discussions.

References

1. Balcázar, J.L., and Lozano, A.: The Complexity of Graph Problems for Succinctly Represented Graphs. In WG 1989, LNCS **411**, Springer, (1989) 277–285
2. Bryant, R.E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers **35** (1986) 677–691
3. Chandra, A.K., Stockmeyer, L., and Vishkin, U.: Constant Depth Reducibility. SIAM Journal on Computing **13** 2 (1984) 423–439
4. Downey, R.G., and Fellows, M.R.: Parameterized Complexity. Springer, Berlin Heidelberg New-York (1999)
5. Feigenbaum, J., Kannan, S., Vardi, M.Y., and Viswanathan, M.: Complexity of Problems on Graphs Represented as OBDDs. In STACS 1998, LNCS **1373**, Springer (1998) 216–226
6. Galperin, H., and Wigderson, A.: Succinct Representations of Graphs. Information and Control **56** (1983) 183–198
7. Gentilini, R., Piazza, C., and Policriti, A.: Computing Strongly Connected Components in a Linear Number of Symbolic Steps. In SODA 2003, ACM Press (2003) 573–582
8. Gentilini, R., and Policriti, A.: Biconnectivity on Symbolically Represented Graphs: A Linear Solution. In ISAAC 2003, LNCS **2906**, Springer (2003) 554–564
9. Greenlaw, R., Hoover, H.J., and Ruzzo, W.L.: Limits to Parallel Computation. Oxford University Press, New York (1995)
10. Hachtel, G.D., and Somenzi, F.: Logic Synthesis and Verification Algorithms. Kluwer Academic Publishers, Boston (1996)
11. Hachtel, G.D., and Somenzi, F.: A Symbolic Algorithm for Maximum Flow in 0–1 Networks. Formal Methods in System Design **10** (1997) 207–219
12. Jones, N.D., Lien, Y.E., and Laaser, W.T.: New Problems Complete for Nondeterministic Log Space. Mathematical Systems Theory **10** (1976) 1–17
13. Nunkesser, R., and Woelfel, P.: Representation of Graphs by OBDDs. To appear in ISAAC 2005
14. Papadimitriou, C.H., and Yannakakis, M.: A Note on Succinct Representations of Graphs. Information and Control **71** (1986) 181–185
15. Sawitzki, D.: Experimental Studies of Symbolic Shortest-Path Algorithms. In WEA 2004, LNCS **3059**, Springer (2004) 482–497
16. Sawitzki, D.: Implicit Flow Maximization by Iterative Squaring. In SOFSEM 2004, LNCS **2932**, Springer (2004) 301–313
17. Sawitzki, D.: On Graphs with Characteristic Bounded-Width Functions. Technical Report, University of Dortmund (2004)
18. Sawitzki, D.: A Symbolic Approach to the All-Pairs Shortest-Paths Problem. In WG 2004, LNCS **3353**, Springer (2004) 154–167
19. Sieling, D., and Wegener, I.: NC-Algorithms for Operations on Binary Decision Diagrams. Parallel Processing Letters **3** (1993) 3–12
20. Wegener, I.: Branching Programs and Binary Decision Diagrams. SIAM, Philadelphia (2000)
21. Woelfel, P.: Symbolic Topological Sorting with OBDDs. In MFCS 2003, LNCS **2747**, Springer (2003) 671–680