Processing Succinct Matrices and Vectors*

Markus Lohrey[1] and Manfred Schmidt-Schauß[2]

[1] Universität Siegen, Department für Elektrotechnik und Informatik, Germany
[2] Institut für Informatik, Goethe-Universität, D-60054 Frankfurt, Germany

Abstract. We study the complexity of algorithmic problems for matrices that are represented by multi-terminal decision diagrams (MTDD). These are a variant of ordered decision diagrams, where the terminal nodes are labeled with arbitrary elements of a semiring (instead of 0 and 1). A simple example shows that the product of two MTDD-represented matrices cannot be represented by an MTDD of polynomial size. To overcome this deficiency, we extended MTDDs to MTDD_+ by allowing componentwise symbolic addition of variables (of the same dimension) in rules. It is shown that accessing an entry, equality checking, matrix multiplication, and other basic matrix operations can be solved in polynomial time for MTDD_+-represented matrices. On the other hand, testing whether the determinant of a MTDD-represented matrix vanishes is PSPACE-complete, and the same problem is NP-complete for MTDD_+-represented diagonal matrices. Computing a specific entry in a product of MTDD-represented matrices is #P-complete. Complete proofs can be found in the full version [19] of this paper.

1 Introduction

Algorithms that work on a succinct representation of certain objects can nowadays be found in many areas of computer science. A paradigmatic example is the use of OBDDs (ordered binary decision diagrams) in hardware verification [5, 21]. OBDDs are a succinct representation of Boolean functions. Consider a boolean function $f(x_1, \ldots, x_n)$ in n input variables. One can represent f by its decision tree, which is a full binary tree of height n with $\{0, 1\}$-labelled leaves. The leaf that is reached from the root via the path $(a_1, \ldots, a_n) \in \{0, 1\}^n$ (where $a_i = 0$ means that we descend to the left child in the i-th step, and $a_i = 1$ means that we descend to the right child in the i-th step) is labelled with the bit $f(a_1, \ldots, a_n)$. This decision tree can be folded into a directed acyclic graph by eliminating repeated occurrences of isomorphic subtrees. The result is the OBDD for f with respect to the variable ordering $x_1, \ldots, x_n$.[1] Bryant was the first who realized that OBDDs are an adequate tool in order to handle the state explosion problem in hardware verification [5].

OBDDs can be also used for storing large graphs. A graph G with 2^n nodes and adjacency matrix M_G can be represented by the boolean function $f_G(x_1, y_1, \ldots, x_n, y_n)$, where $f_G(a_1, b_1, \ldots, a_n, b_n)$ is the entry of M_G at position (a, b); here $a_1 \cdots a_n$ (resp.,

* The first (second) author is supported by the DFG grant LO 748/8-2 (SCHM 986/9-2).

[1] Here, we are cheating a bit: In OBDDs a second elimination rule is applied that removes nodes for which the left and right child are identical. On the other hand, it is known that asymptotically the compression achieved by this elimination rule is negligible [31].

$b_1 \cdots b_n$) is the binary representation of the index a (resp. b). Note that we use the so called interleaved variable ordering here, where the bits of the two coordinates a and b are bitwise interleaved. This ordering turned out to be convenient in the context of OBDD-based graph representation, see e.g. [10].

Classical graph problems (like reachability, alternating reachability, existence of a Hamiltonian cycle) have been studied for OBDD-represented graphs in [9, 30]. It turned out that these problems are exponentially harder for OBDD-represented graphs than for explicitly given graphs. In [30] an upgrading theorem for OBDD-represented graphs was shown. It roughly states that completeness of a problem A for a complexity class C under quantifier free reductions implies completeness of the OBDD-variant of A for the exponentially harder version of C under polynomial time reductions.

In the same way as OBDDs represent boolean mappings, functions from $\{0, 1\}^n$ to any set S can be represented. One simply has to label the leaves of the decision tree with elements from S. This yields multi-terminal decision diagrams (MTDDs) [11]. Of particular interest is the case, where S is a semiring, e.g. $\mathbb{N}$ or $\mathbb{Z}$. In the same way as an adjacency matrix (i.e., a boolean matrix) of dimension 2^n can be represented by an OBDD, a matrix of dimension 2^n over any semiring can be represented by an MTDD. As for OBDDs, we assume that the bits of the two coordinates a and b are interleaved in the order $a_1, b_1, \ldots, a_n, b_n$. This implies that an MTDD can be viewed as a set of rules of the form

$$A \to \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \qquad \text{or} \qquad B \to a \text{ with } a \in S. \tag{1}$$

where A, $A_{1,1}$, $A_{1,2}$, $A_{2,1}$, and $A_{2,2}$ are variables that correspond to certain nodes of the MTDD (namely those nodes that have even distance from the root node). Every variable produces a matrix of dimension 2^h for some $h \geq 0$, which we call the height of the variable. The variables $A_{i,j}$ in (1) must have the same height h, and A has height $h+1$. The variable B has height 0. We assume that the additive monoid of the semiring S is finitely generated, hence every $a \in S$ has a finite representation.

MTDDs yield very compact representations of sparse matrices. It was shown that an $(n \times n)$-matrix with m nonzero entries can be represented by an MTDD of size $O(m \log n)$ [11, Thm. 3.2], which is better than standard succinct representations for sparse matrices. Moreover, MTDDs can also yield very compact representations of non-sparse matrices. For instance, the Walsh matrix of dimension 2^n can be represented by an MTDD of size $O(n)$, see [11]. In fact, the usual definition of the n-th Walsh matrix is exactly an MTDD. Matrix algorithms for MTDDs are studied in [11] as well, but no precise complexity analysis is carried out. In fact, the straightforward matrix multiplication algorithm for multi-terminal decision diagrams from [11] has an exponential worst case running time, and this is unavoidable: The smallest MTDD that produces the product of two MTDD-represented matrices may be of exponential size in the two MTDDs, see Thm. 2. The first main contribution of this paper is a generalization of MTDDs that overcomes this deficiency: An MTDD_+ consists of rules of the form (1) together with addition rules of the form $A \to B+C$, where "$+$" refers to matrix addition over the underlying semiring. Here, A, B, and C must have the same height, i.e., produce matrices of the same dimension. We show that an MTDD_+ for the product of two MTDD_+-represented matrices can be computed in polynomial time (Thm. 3). In Sec. 4.1 we also present efficient (polynomial time) algorithms for several other important matrix problems on MTDD_+-represented input matrices: computation of a specific matrix entry, computation of the trace, matrix transposition, tensor and Hadamard product. Sec. 5 deals with equality checking. It turns out that equality of MTDD_+-represented matrices can be checked in polynomial time, if the additive monoid is cancellative, in all other cases equality checking is coNP-complete.

To the knowledge of the authors, complexity results similar to those from [9, 30] for OBDDs do not exist in the literature on MTDDs. Our second main contribution fills this gap. We prove that already for MTDDs over $\mathbb{Z}$ it is PSPACE-complete to check whether the determinant of the generated matrix is zero (Thm. 6). This result is shown by lifting a classical construction of Toda [27] (showing that computing the determinant of an explicitly given integer matrix is complete for the counting class GapL) to configuration graphs of polynomial space bounded Turing machines, which are of exponential size. It turns out that the adjacency matrix of the configuration graph of a polynomial space bounded Turing machine can be produced by a small MTDD. Thm. 6 sharpens a recent result from [14] stating that it is PSPACE-complete to check whether the determinant of a matrix that is represented by a boolean circuit (see Sec. 4.2) vanishes. We also prove several hardness results for counting classes. For instance, computing a specific entry of a matrix power A^n, where A is given by an MTDD over $\mathbb{N}$ is $\#\mathsf{P}$-complete (resp. $\#\mathsf{PSPACE}$-complete) if n is given unary (resp. binary). Here, $\#\mathsf{P}$ (resp. $\#\mathsf{PSPACE}$) is the class of functions counting the number of accepting computations of a nondeterministic polynomial time Turing machine [29] (resp., a nondeterministic polynomial space Turing machine [15]). An example of a natural $\#\mathsf{PSPACE}$-complete counting problem is counting the number of strings not accepted by a given NFA [15].

2 Related work

Sparse matrices and quad-trees. To the knowledge of the authors, most of the literature on matrix compression deals with sparse matrices, where most of the matrix entries are zero. There are several succinct representations of sparse matrices. One of which are *quad-trees*, used in computer graphics for the representation of large constant areas in 2-dimensional pictures, see for example [24, 8]. Actually, an MTDD can be seen as a quad-tree that is folded into a dag by merging identical subtrees.

Two-dimensional straight-line programs. MTDDs are also a special case of 2-dimensional straight-line programs (SLPs). A (1-dimensional) SLP is a context-free grammar in Chomsky normal form that generates exactly one OBDD. An SLP with n rules can generate a string of length 2^n; therefore an SLP can be seen as a succinct representation of the string it generates. Algorithmic problems that can be solved efficiently (in polynomial time) on SLP-represented strings are for instance equality checking (first shown by Plandowski [23]) and pattern matching, see [18] for a survey.

In [3] a 2-dimensional extension of SLPs (2SLPs in the following) was defined. Here, every variable of the grammar generates a (not necessarily square) matrix (or picture), where every position is labeled with an alphabet symbol. Moreover, there are two (partial) concatenation operations: horizontal composition (which is defined for two pictures if they have the same height) and vertical composition (which is defined for two pictures if they have the same width). This formalism does not share all the nice algorithmic properties of (1-dimensional) SLPs [3]: Testing whether two 2SLPs produce the same picture is only known to be in coRP (co-randomized polynomial time). Moreover, checking whether an explicitly given (resp., 2SLP-represented) picture appears within a 2SLP-represented picture is NP-complete (resp., Σ_2^P-complete). Related hardness results in this direction concern the convolution of two SLP-represented strings of the same length (which can be seen as a picture of height 2). The convolution of strings $u = a_1 \cdots a_n$ and $v = b_1 \cdots b_n$ is the string $(a_1, b_1) \cdots (a_n, b_n)$. By a result from [4] (which is stated in terms of the related operation of literal shuffle), the size of a shortest SLP for the convolution of two strings that are given by SLPs G and H may be exponential in the size of G and H. Moreover, it is PSPACE-complete to check for two SLP-represented strings u and v and an NFA T operating on strings of pairs of symbols, whether T accepts the convolution of u and v [17].

MTDDs restrict 2SLPs by forbidding unbalanced derivation trees. The derivation tree of an MTDD results from unfolding the rules in (1); it is a tree, where every non-leaf node has exactly four children and every root-leaf path has the same length.

Tensor circuits. In [2, 7], the authors investigated the problems of evaluating tensor formulas and tensor circuits. Let us restrict to the latter. A tensor circuit is a circuit where the gates evaluate to matrices over a semiring and the following operations are used: matrix addition, matrix multiplication, and tensor product. Recall that the tensor product of two matrices $A = (a_{i,j})_{1 \le i \le m, 1 \le i \le m}$ and B is the matrix

$$A \otimes B = \begin{pmatrix} a_{1,1}B & \cdots & a_{1,m}B \\ \vdots & & \vdots \\ a_{n,1}B & \cdots & a_{n,m}B \end{pmatrix}$$

It is a $(mk \times nl)$-matrix if B is a $(k \times l)$-matrix. In [2] it is shown among other results that computing the output value of a scalar tensor circuit (i.e., a tensor circuit that yields a (1×1)-matrix) over the natural numbers is complete for the counting class #EXP. An MTDD_+ over $\mathbb{Z}$ can be seen as a tensor circuit that (i) does not use matrix multiplication and (ii) where for every tensor product the left factor is a (2×2)-matrix. To see the correspondence, note that

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \otimes A_{1,1} + \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \otimes A_{1,2} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \otimes A_{2,1} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes A_{2,2}$$

$$\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \otimes B = \begin{pmatrix} a_{1,1}B & a_{1,2}B \\ a_{2,1}B & a_{2,2}B \end{pmatrix}$$

Each of the matrices $a_{i,j}B$ can be generated from B and $-B$ using $\log |a_{i,j}|$ many additions (here we use the fact that the underlying semiring is $\mathbb{Z}$).

3 Preliminaries

We consider matrices over a semiring $(S, +, \cdot)$ with $(S, +)$ a finitely generated commutative monoid with unit 0. The unit of the monoid $(S, \cdot)$ is 1. We assume that $0 \cdot a =$

$a \cdot 0 = 0$ for all $a \in S$. Hence, if $|S| > 1$, then $1 \neq 0$ ($0 = 1$ implies $a = 1 \cdot a = 0 \cdot a = 0$ for all $a \in S$). With $S^{n \times n}$ we denote the set of all $(n \times n)$-matrices over S.

All time bounds in this paper implicitly refer to the RAM model of computation with a logarithmic cost measure for arithmetical operations on integers, where arithmetic operations on n-bit numbers need time $O(n)$. For a number $n \in \mathbb{Z}$ let us denote with $\mathsf{bin}(n)$ its binary encoding.

We assume that the reader has some basic background in complexity theory, in particular we assume that the reader is familiar with the classes NP, coNP, and PSPACE. A function $f : \{0,1\}^* \to \{0,1\}^*$ belongs to the class $\mathsf{FSPACE}(s(n))$ (resp. $\mathsf{FTIME}(s(n))$) if f can be computed on a deterministic Turing machine in space (resp., time) $s(n)$.[2] As usual, only the space on the working tapes is counted. Moreover, the output is written from left to right on the output tape, i.e., in each step the machine either outputs a new symbol on the output tape, in which case the output head moves one cell to the right, or the machine does not output a new symbol in which case the output head does not move. Let $\mathsf{FP} = \bigcup_{k \geq 1} \mathsf{FTIME}(n^k)$ and $\mathsf{FPSPACE} = \bigcup_{k \geq 1} \mathsf{FSPACE}(n^k)$. Note that for a function $f \in \mathsf{FPSPACE}$ we have $|f(w)| \leq 2^{|w|^{O(1)}}$ for every input.

The counting class $\#\mathsf{P}$ consists of all functions $f : \{0,1\}^* \to \mathbb{N}$ for which there exists a nondeterministic polynomial time Turing machine M with input alphabet Σ such that for all $x \in \Sigma^*$, $f(x)$ is the number of accepting computation paths of M for input x. If we replace nondeterministic polynomial time Turing machines by nondeterministic polynomial space Turing machines (resp. nondeterministic logspace Turing machines), we obtain the class $\#\mathsf{PSPACE}$ [15] (resp. $\#\mathsf{L}$ [1]). Note that for a mapping $f \in \#\mathsf{PSPACE}$, the number $f(x)$ may grow doubly exponential in $|x|$, whereas for $f \in \#\mathsf{P}$, the number $f(x)$ is bounded singly exponential in $|x|$. Ladner [15] has shown that a mapping $f : \Sigma^* \to \mathbb{N}$ belongs to $\#\mathsf{PSPACE}$ if and only if the mapping $x \mapsto \mathsf{bin}(f(x))$ belongs to $\mathsf{FPSPACE}$. One cannot expect a corresponding result for the class $\#\mathsf{P}$: If for every function $f \in \#\mathsf{P}$ the mapping $x \mapsto \mathsf{bin}(f(x))$ belongs to FP, then by Toda's theorem [28] the polynomial time hierarchy collapses down to P. For $f \in \#\mathsf{L}$, the mapping $x \mapsto \mathsf{bin}(f(x))$ belongs to NC^2 and hence to $\mathsf{FP} \cap \mathsf{FSPACE}(\log^2(n))$ [1, Thm. 4.1]. The class GapL (resp., GapP, $\mathsf{GapPSPACE}$) consists of all differences of two functions in $\#\mathsf{L}$ (resp., $\#\mathsf{P}$, $\#\mathsf{PSPACE}$). From Ladner's result [15] it follows easily that a function $f : \{0,1\}^* \to \mathbb{Z}$ belongs to $\mathsf{GapPSPACE}$ if and only if the mapping $x \mapsto \mathsf{bin}(f(x))$ belongs to $\mathsf{FPSPACE}$, see also [12, Thm. 6].

Logspace reductions between functions can be defined analogously to the language case: If $f, g : \{0,1\}^* \to X$ with $X \in \{\mathbb{N}, \mathbb{Z}\}$, then f is logspace reducible to g if there exists a function $h \in \mathsf{FSPACE}(\log n)$ such that $f(x) = g(h(x))$ for all x. Toda [27] has shown that computing the determinant of a given integer matrix is GapL-complete.

4 Succinct matrix representations

In this section, we introduce several succinct matrix representations. We formally define multi-terminal decision diagrams and their extension by the addition operation. Moreover, we briefly discuss the representation of matrices by boolean circuits.

[2] The assumption that the input and output alphabet of f is binary is made here to make the definitions more readable; the extension to arbitrary finite alphabets is straightforward.

4.1 Multi-terminal decision diagrams

Fix a semiring $(S, +, \cdot)$ with $(S, +)$ a finitely generated commutative monoid, and let $\Gamma \subseteq S$ be a finite generating set for $(S, +)$. Thus, every element of S can be written as a finite sum $\sum_{a \in \Gamma} n_a a$ with $n_a \in \mathbb{N}$. A *multi-terminal decision diagram G with addition* (MTDD_+) *of height h* is a triple (N, P, A_0), where N is a finite set of variables which is partitioned into non-empty sets N_i $(0 \leq i \leq h)$, $N_h = \{A_0\}$ (A_0 is called the *start variable*), and P is a set of rules of the following three forms:

- $A \to \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}$ with $A \in N_i$ and $A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2} \in N_{i-1}$ for some $1 \leq i \leq h$
- $A \to A_1 + A_2$ with $A, A_1, A_2 \in N_i$ for some $0 \leq i \leq h$
- $A \to a$ with $A \in N_0$ and $a \in \Gamma \cup \{0\}$

Moreover, for every variable $A \in N$ there is exactly one rule with left-hand side A, and the relation $\{(A, B) \in N \times N \mid B$ occurs in the right-hand side for $A\}$ is acyclic. If $A \in N_i$ then we say that A has height i. The MTDD_+ G is called an ***MTDD*** if for every addition rule $(A \to A_1 + A_2) \in P$ we have $A, A_1, A_2 \in N_0$. In other words, only scalars are allowed to be added. Since we assume that $(S, +)$ is generated by Γ, this allows to produce arbitrary elements of S as matrix entries. For every $A \in N_i$ we define a square matrix $\mathsf{val}(A)$ of dimension 2^i in the obvious way by unfolding the rules. Moreover, let $\mathsf{val}(G) = \mathsf{val}(A_0)$ for the start variable A_0 of G. This is a $(2^h \times 2^h)$-matrix. The size of a rule $A \to a$ with $a \in \Gamma \cup \{0\}$ is 1, all other rules have size $\log |N|$. The size $|G|$ of the MTDD_+ G is the sum of the sizes of its rules; this is up to constant factors the length of the binary coding of G. An MTDD_+ G of size $n \log n$ can represent a $(2^n \times 2^n)$-matrix. Note that only square matrices whose dimension is a power of 2 can be represented. Matrices not fitting this format can be filled up appropriately, depending on the purpose.

An MTDD, where all rules have the form $A \to a \in \Gamma \cup \{0\}$ or $A \to B + C$ generates an element of the semiring S. Such an MTDD is an arithmetic circuit in which only input gates and addition gates are used, and is called a $+$*-circuit* in the following. In case the underlying semiring is $\mathbb{Z}$, a $+$-circuit with n variables can produce a number of size 2^n, and the binary encoding of this number can be computed in time $\mathcal{O}(n^2)$ from the $+$-circuit (since, we need n additions of numbers with at most n bits). In general, for a $+$-circuit over the semiring S, we can compute in quadratic time numbers n_a $(a \in \Gamma)$ such that $\sum_{a \in \Gamma} n_a \cdot a$ is the semiring element to which the $+$-circuit evaluates to.

Note that the notion of an MTDD_+ makes sense for commutative monoids, since we only used the addition of the underlying semiring. But soon, we want to multiply matrices, for which we need a semiring. Moreover, the notion of an MTDD_+ makes sense in any dimension, here we only defined the 2-dimensional case.

Example 1. It is straightforward to produce the unit matrix I_{2^n} of dimension 2^n by an MTDD of size $O(n \log n)$:

$$A_0 \to 1,\ \ 0_0 \to 0,\ \ A_j \to \begin{pmatrix} A_{j-1} & 0_{j-1} \\ 0_{j-1} & A_{j-1} \end{pmatrix},\ \ 0_j \to \begin{pmatrix} 0_{j-1} & 0_{j-1} \\ 0_{j-1} & 0_{j-1} \end{pmatrix}\ \ (1 \leq j \leq n).$$

(the start variable is A_n here). In a similar way, one can produce the lower triangular $(2^n \times 2^n)$-matrix, where entries on the diagonal and below are 1. To produce the $(2^n \times 2^n)$-matrix over $\mathbb{Z}$, where all entries in the k-th row are k, we need the following rules:

$$E_0 \to 1, \;\; E_j \to \begin{pmatrix} E_{j-1} + E_{j-1} & E_{j-1} + E_{j-1} \\ E_{j-1} + E_{j-1} & E_{j-1} + E_{j-1} \end{pmatrix} \;\; (1 \leq j \leq n)$$
$$C_0 \to 1, \;\; C_j \to \begin{pmatrix} C_{j-1} & C_{j-1} \\ C_{j-1} + E_{j-1} & C_{j-1} + E_{j-1} \end{pmatrix} \;\; (1 \leq j \leq n).$$

Here, we are bit more liberal with respect to the format of rules, but the above rules can be easily brought into the form from the general definition of an MTDD_+. Note that E_j generates the $(2^j \times 2^j)$-matrix with all entries equal to 2^j, and that C_n generates the desired matrix.

Note that the matrix from the last example cannot be produced by an MTDD of polynomial size, since it contains an exponential number of different matrix entries (for the same reason it cannot be produced by an 2SLP [3]). This holds for any non-trivial semiring.

Theorem 1. *For any semiring with at least two elements, MTDD$_+$ are exponentially more succinct than MTDDs.*

Proof. For simplicity we argue with MTDDs in dimension 1 (which generate vectors). We must have $1 \neq 0$ in S. Let $m, d > 0$ be such that $m = 2^d$. For $0 \leq i \leq m-1$ let A_i such that $\mathsf{val}(A_i)$ has length m, the i-th entry is 1 (the first entry is the 0-th entry) and all other entries are 0. Moreover, let B_i such that $\mathsf{val}(B_i)$ is the concatenation of 2^i copies of $\mathsf{val}(A_i)$. Let C_0 produce the 0-vector of length $m = 2^d$, and for $0 \leq i \leq m-1$ let $C_{i+1} \to (C_i, C_i + B_i)$. Then $\mathsf{val}(C_m)$ is of length 2^{d+m} and consists of the concatenation of all binary strings of length m. This MTDD_+ for this vector is of size $O(m^2 \log m)$, whereas an equivalent MTDD must have size at least 2^m, since for every binary string of length m there must exist a nonterminal. $\square$

The following result shows that the matrix product of two MTDD-represented matrices may be incompressible with MTDDs.

Theorem 2. *For any semiring with at least two elements there exist MTDDs G_n and H_n of the same height n and size $O(n^2 \log n)$ such that $\mathsf{val}(G_n) \cdot \mathsf{val}(H_n)$ can only be represented by an MTDD of size at least 2^n.*

On the other hand, the product of two MTDD_+-represented matrices can be represented by a polynomially sized MTDD_+:

Theorem 3. *For MTDD_+ G_1 and G_2 of the same height one can compute in time $O(|G_1| \cdot |G_2|)$ an MTDD_+ G of size $O(|G_1| \cdot |G_2|)$ with $\mathsf{val}(G) = \mathsf{val}(G_1) \cdot \mathsf{val}(G_2)$.*

For the proof, we compute from G_1 and G_2 a new MTDD_+ G that contains for all variables A of G_1 and B of G_2 of the same height a variable (A, B) such that $\mathsf{val}_G(A, B) = \mathsf{val}_{G_1}(A) \cdot \mathsf{val}_{G_2}(B)$.

The following proposition presents several further matrix operations that can be easily implemented in polynomial time for an MTDD_+-represented input matrix.

Proposition 1. *Let G, H be a MTDD_+ with $|G| = n$, $|H| = m$, and $1 \leq i, j \leq 2^{\mathsf{height}(G)}$*

(1) An MTDD_+ for the transposition of $\mathsf{val}(G)$ can be computed in time $O(n)$.
(2) $+$-circuits for the sum of all entries of $\mathsf{val}(G)$ and the trace of $\mathsf{val}(G)$ can be computed in time $O(n)$.
(3) A $+$-circuit for the matrix entry $\mathsf{val}(G)_{i,j}$ can be computed in time $O(n)$.
(4) MTDD_+ of size $O(n \cdot m)$ for the tensor product $\mathsf{val}(G) \otimes \mathsf{val}(H)$ (which includes the scalar product) and the element-wise (Hadamard) product $\mathsf{val}(G) \circ \mathsf{val}(H)$ (assuming $\mathsf{height}(G) = \mathsf{height}(H)$) can be computed in time $O(n \cdot m)$.

4.2 Boolean circuits

Another well-studied succinct representation are boolean circuits [13]. A boolean circuit with n inputs represents a binary string of length 2^n, namely the string of output values for the 2^n many input assignments (concatenated in lexicographic order). In a similar way, we can use circuits to encode large matrices. We propose two alternatives:

A boolean circuit $C(\overline{x}, \overline{y}, \overline{z})$ with $|\overline{x}| = m$ and $|\overline{y}| = |\overline{z}| = n$ encodes a $(2^n \times 2^n)$-matrix $M_{C,2}$ with integer entries bounded by 2^{2^m} that is defined as follows: For all $\overline{a} \in \{0,1\}^m$ and $\overline{b}, \overline{c} \in \{0,1\}^n$, the $\overline{a}$-th bit (in lexicographic order) of the matrix entry at position $(\overline{b}, \overline{c})$ in M_C is 1 if and only if $C(\overline{a}, \overline{b}, \overline{c}) = 1$.

Note that in contrast to MTDD_+, the size of an entry in $M_{C,2}$ can be doubly exponential in the size of the representation C (this is the reason for the index 2 in $M_{C,2}$). The following alternative is closer to MTDD_+: A boolean circuit $C(\overline{x}, \overline{y})$ with $|\overline{x}| = |\overline{y}| = n$ and m output gates encodes a $(2^n \times 2^n)$-matrix $M_{C,1}$ with integer entries bounded by 2^m that is defined as follows: For all $\overline{a}, \overline{b} \in \{0,1\}^n$, $C(\overline{a}, \overline{b})$ is the binary encoding of the entry at position $(\overline{a}, \overline{b})$ in M_C.

Circuit representations for matrices are at least as succinct as MTDD_+. More precisely, from a given MTDD_+ G one can compute in logspace a Boolean circuit C such that $M_{C,1} = \mathsf{val}(G)$. This is a direct corollary of Proposition 1(3) (stating that a given entry of an MTDD_+-represented matrix can be computed in polynomial time) and the fact that polynomial time computations can be simulated by boolean circuits. Recently, it was shown that checking whether for a given circuit C the determinant of the matrix $M_{C,1}$ vanishes is PSPACE-complete [14]. An algebraic version of this result for the algebraic complexity class $\mathsf{VPSPACE}$ is shown in [20]. Thm. 6 from Sec. 6 will strengthen the result from [14] to MTDD-represented matrices.

5 Testing equality

In this section, we consider the problem of testing equality of MTDD_+-represented matrices. For this, we do not need the full semiring structure, but we only need the finitely generated additive monoid $(S, +)$. We will show that equality can be checked in polynomial time if $(S, +)$ is cancellative and coNP-complete otherwise.

First we consider the case of a finitely generated abelian group. The proof of the following lemma involves only basic linear algebra.

Lemma 1. *Let $a_{i,1}x_1 + \cdots + a_{i,n}x_n = 0$ for $1 \leq i \leq m \leq n+1$ be equations over a torsion-free abelian group A, where $a_{i,1}, \ldots, a_{i,n} \in \mathbb{Z}$, and the variables $x_1, \ldots, x_n$ range over A. One can determine in time polynomial in n and $\max\{\log |a_{i,j}| \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ an equivalent set of at most n linear equations.*

Recall that the *exponent* of an abelian group A is the smallest integer k (if it exists) such that $kg = 0$ for all $g \in A$. The following result is shown in [25]:

Lemma 2. *Let $k \geq 2$ and let A be an abelian group of exponent k. Let $a_{i,1}x_1 + \cdots + a_{i,n}x_n = 0$ for $1 \leq i \leq m \leq n+1$ be equations, where $a_{i,1}, \ldots, a_{i,n} \in \mathbb{Z}$, and the variables $x_1, \ldots, x_n$ range over A. Then one can determine in time polynomial in n, $\log(k)$, and $\max\{\log |a_{i,j}| \mid 1 \leq i \leq m, 1 \leq j \leq n\}$ an equivalent set of at most n linear equations.*

Proof. We can consider the coefficients $a_{i,j}$ as elements from $\mathbb{Z}_k$. By [25] we can compute the Howell normal form of the matrix $(a_{i,j})_{1\leq i\leq n+1, 1\leq j\leq n} \in \mathbb{Z}_k^{(n+1)\times n}$ in polynomial time. The Howell normal form is an $(n \times n)$-matrix with the same row span (a subset of the module $\mathbb{Z}_k^n$) as the original matrix, and hence defines an equivalent set of linear equations. $\square$

Theorem 4. *Let G be an MTDD_+ over a finitely generated abelian group S. Given two different variables A_1, A_2 of the same height, it is possible to check $\mathsf{val}(A_1) = \mathsf{val}(A_2)$ in time polynomial in $|G|$.*

Proof. Since every finitely generated group is a finite direct product of copies of $\mathbb{Z}$ and $\mathbb{Z}_k$ ($k \geq 2$), it suffices to prove the theorem only for these groups.

Consider the case $S = \mathbb{Z}$. The algorithm stores a system of m equations (m will be bounded later) of the form $a_{i,1}B_1 + \cdots + a_{i,k}B_k = 0$, where all $B_1, \ldots, B_k$ are pairwise different variables of the same height h. We treat the variables $B_1, \ldots, B_k$ as variables that range over the torsion-free abelian group $\mathbb{Z}^{2^h \times 2^h}$. We start with the single equation $A_1 - A_2 = 0$. We use the rules of G to transform the system of equations into another system of equations whose variables have strictly smaller height. Assume the current height is $h > 1$. We iterate the following steps until only variables of height $h-1$ occur in the equations:

Step 1. Standardize equations: Transform all equations into the form $a_1B_1 + \cdots + a_mB_m = 0$, where the B_i are different variables and the a_i are integers.

Step 2. Reduce the number of equations, using Lemma 1 applied to the torsion-free abelian group $\mathbb{Z}^{2^h \times 2^h}$.

Step 3. If a variable A of height h occurs in the equations, and the rule for A has the form $A \to A_1 + A_2$, then replace every occurrence of A in the equations by $A_1 + A_2$.

Step 4. If none of steps 1–3 applies to the equations, then only rules of the form

$$A \to \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \tag{2}$$

are applicable to a variable A (of height h) occurring in the equations. Applying all possible rules of this form for the current height results in a set of equations where all variables are (2×2)-matrices over variables of height $h-1$ (like the right-hand side of (2)). Hence, every equation can be decomposed into 4 equations, where all variables are variables of height $h-1$.

If the height of all variables is finally 0, then only rules of the form $A \to a$ are applicable. In this case, replace all variables by the corresponding integers, and check whether all resulting equations are valid or not. If all equations hold, then the input equation holds, i.e., $\mathsf{val}(A_1) = \mathsf{val}(A_2)$. Otherwise, if at least one equation is not valid, then $\mathsf{val}(A_1) \neq \mathsf{val}(A_2)$.

The number of variables in the equations is bounded by the number of variables of G. An upper bound on the absolute value of the coefficients in the equations is $2^{|G|}$, since only iterated addition can be performed to increase the coefficients. Lemma 1 shows that the number of equations after step 2 above is at most $|G|$, (the bound for the number of different variables).

For the case $S = \mathbb{Z}_k$ the same procedure works, we only have to use Lemma 2 instead of Lemma 1. $\square$

Corollary 1. *Let M be a finitely generated cancellative commutative monoid. Given an MTDD_+ G over M and two variables A_1 and A_2 of G, one can check $\mathsf{val}(A_1) = \mathsf{val}(A_2)$ in time polynomial in $|G|$.*

Proof. A cancellative commutative monoid M embeds into its Grothendieck group A, which is the quotient of $M \times M$ by the congruence defined by $(a, b) \equiv (c, d)$ if and only if $a + d = c + b$ in M. This is an abelian group, which is moreover finitely generated if M is finitely generated. Hence, the result follows from Thm. 1. $\square$

Let us now consider non-cancellative commutative monoids:

Theorem 5. *Let M be a non-cancellative finitely generated commutative monoid. It is coNP-complete to check $\mathsf{val}(A_1) = \mathsf{val}(A_2)$ for a given MTDD_+ G over M and two variables A_1 and A_2 of G.*

Proof. We start with the upper bound. Let $\{a_1, \ldots, a_k\}$ be a finite generating set of M. Let G be an MTDD_+ over M and let A_1 and A_2 two variables of G. Assume that A_1 and A_2 have the same height h. It suffices to check in polynomial time for two given indices $1 \leq i, j \leq 2^h$ whether $\mathsf{val}(A_1)_{i,j} \neq \mathsf{val}(A_2)_{i,j}$. From $1 \leq i, j \leq 2^h$ we can compute $+$-circuits for the matrix entries $\mathsf{val}(A_1)_{i,j}$ and $\mathsf{val}(A_2)_{i,j}$. From these circuits we can compute numbers $n_1, \ldots, n_k, m_1, \ldots, m_k \in \mathbb{N}$ in binary representation such that $\mathsf{val}(A_1)_{i,j} = n_1 a_1 + \cdots + n_k a_k$ and $\mathsf{val}(A_2)_{i,j} = m_1 a_1 + \cdots + m_k a_k$. Now we can use the following result from [26]: There is a semilinear subset $S \subseteq \mathbb{N}^{2k}$ (depending only on our fixed monoid M) such that for all $x_1, \ldots, x_k, y_1, \ldots, y_k \in \mathbb{N}$ we have: $x_1 a_1 + \cdots + x_k a_k = y_1 a_1 + \cdots + y_k a_k$ if and only if $(x_1, \ldots, x_k, y_1, \ldots, y_k) \in S$. Hence, we have to check, whether $v =: (n_1, \ldots, n_k, m_1, \ldots, m_k) \in S$. The semilinear set S is a finite union of linear sets. Hence, we can assume that S is linear itself. Let

$$S = \{v_0 + \lambda_1 v_1 + \cdots + \lambda_l v_l \mid \lambda_1, \ldots, \lambda_l \in \mathbb{N}\},$$

where $v_0, \ldots, v_l \in \mathbb{N}^{2k}$. Hence, we have to check, whether there exist $\lambda_1, \ldots, \lambda_l \in \mathbb{N}$ such that $v = v_0 + \lambda_1 v_1 + \cdots \lambda_l v_l$. This is an instance of integer programming in the fixed dimension $2k$, which can be solved in polynomial time [16].

For the lower bound we take elements $x, y, z \in M$ such that $x \neq y$ but $x+z = y+z$. These elements exist since M is not cancellative. We use an encoding of 3SAT from [3]. Take a 3CNF formula $C = \bigwedge_{i=1}^{m} C_i$ over n propositional variables $x_1, \ldots, x_n$, and let $C_i = (\alpha_{j_1} \vee \alpha_{j_2} \vee \alpha_{j_3})$, where $1 \leq j_1 < j_2 < j_3 \leq n$ and every α_{j_k} is either x_{j_k} or $\neg x_{j_k}$. For every $1 \leq i \leq m$ we define an MTDD G_i as follows: The variables are $A_0, \ldots, A_n$, and $B_0, \ldots, B_{n-1}$, where B_i produces the vector of length 2^i with all entries equal to 0 (which corresponds to the truth value true, whereas $z \in M$ corresponds to the truth value false). For the variables $A_0, \ldots, A_n$ we add the following rules: For every $1 \leq j \leq n$ with $j \notin \{j_1, j_2, j_3\}$ we take the rule $A_j \to (A_{j-1}, A_{j-1})$. For every $j \in \{j_1, j_2, j_3\}$ such that $\alpha_j = x_j$ (resp. $\alpha_j = \neg x_j$) we take the rule

$$A_j \to (A_{j-1}, B_{j-1}) \text{ (resp. } A_j \to (B_{j-1}, A_{j-1})).$$

Finally add the rule $A_0 \to z$ and let A_n be the start variable of G_i. Moreover, let G (resp. H) be the 1-dimensional MTDD that produces the vector consisting of 2^n many x-entries (resp. y-entries). Then, $\mathsf{val}(G) + \mathsf{val}(G_1) + \cdots + \mathsf{val}(G_m) = \mathsf{val}(H) + \mathsf{val}(G_1) + \cdots + \mathsf{val}(G_m)$ if and only if C is unsatisfiable. $\square$

It is worth noting that in the above proof for coNP-hardness, we use addition only at the top level in a non-nested way.

6 Computing determinants and matrix powers

In this section we present several completeness results for MTDDs over the rings $\mathbb{Z}$ and $\mathbb{Z}_n$ ($n \geq 2$). It turns out that over these rings, computing determinants, iterated matrix products, or matrix powers are infeasible for MTDD-represented input matrices, assuming standard assumptions from complexity theory. All completeness results in this section are formulated for MTDDs, but they remain valid if we add addition. In fact, all upper complexity bounds in this section even hold for matrices that are represented by circuits as explained in Sec. 4.2.

The value $\det(\mathsf{val}(G))$ for an MTDD G may be of doubly exponential size (and hence needs exponentially many bits): The diagonal $(2^n \times 2^n)$-matrix with 2's on the diagonal has determinant 2^{2^n}. We first show that checking whether the determinant of an MTDD-represented matrix over any of the rings $\mathbb{Z}$ or $\mathbb{Z}_n$ ($n \geq 2$) vanishes is PSPACE-complete, and that computing the determinant over $\mathbb{Z}$ is $\mathsf{GapPSPACE}$-complete:

Theorem 6. *The following holds for every ring $S \in \{\mathbb{Z}\} \cup \{\mathbb{Z}_n \mid n \geq 2\}$:*

(1) The set $\{G \mid G$ is an MTDD over $S, \det(\mathsf{val}(G)) = 0\}$ is PSPACE-complete.
(2) The function $G \mapsto \det(\mathsf{val}(G))$ with G an MTDD over $\mathbb{Z}$ is $\mathsf{GapPSPACE}$-complete.

To prove this result we use a reduction of Toda showing that computing the determinant of an explicitly given integer matrix is GapL-complete [27]. We apply this reduction to configuration graphs of polynomial space bounded Turing machines, which are of exponential size. It turns out that the adjacency matrix of the configuration graph of a polynomial space bounded machine can be produced by a small MTDD (with terminal entries 0 and 1). This was also shown in [9, proof of Thm. 7] in the context of OBDDs.

Note that the determinant of a diagonal matrix is zero if and only if there is a zero-entry on the diagonal. This can be easily checked in polynomial time for a diagonal matrix produced by an MTDD. For MTDD_+ (actually, for a sum of several MTDD-represented matrices) we can show NP-completeness of this problem:

Theorem 7. *It is* NP*-complete to check* $\det(\mathsf{val}(G_1) + \cdots + \mathsf{val}(G_k)) = 0$ *for given MTDDs* $G_1, \ldots, G_k$ *that produce diagonal matrices of the same dimension.*

Our NP-hardness proof uses again the 3SAT encoding from [3] that we applied in the proof of Thm. 5.

Let us now discuss the complexity of iterated multiplication and powering. Computing a specific entry, say at position $(1, 1)$, of the product of n explicitly given matrices over $\mathbb{Z}$ (resp., $\mathbb{N}$) is known to be complete for GapL (resp., $\#\mathsf{L}$) [27]. Corresponding results hold for the computation of the $(1, 1)$-entry of a matrix power A^n, where n is given in unary notation. As usual, these problems become exponentially harder for matrices that are encoded by boolean circuits (see Sec. 4.2). Let us briefly discuss two scenarios (recall the matrices $M_{C,1}$ and $M_{C,2}$ defined from a circuit in Sec. 4.2).

Definition 1. *For a tuple* $\overline{C} = (C_1, \ldots, C_n)$ *of boolean circuits we can define the matrix product* $M_{\overline{C}} = \prod_{i=1}^{n} M_{C_i,1}$.

Lemma 3. *The function* $\overline{C} \mapsto (M_{\overline{C}})_{1,1}$*, where every matrix* $M_{C_i,1}$ *is over* $\mathbb{N}$ *(resp.,* $\mathbb{Z}$*), belongs to* $\#\mathsf{P}$ *(resp.,* GapP*).*

Definition 2. *A boolean circuit* $C(\overline{w}, \overline{x}, \overline{y}, \overline{z})$ *with* $k = |\overline{w}|$, $m = |\overline{x}|$*, and* $n = |\overline{y}| = |\overline{z}|$ *encodes a sequence of* 2^k *many* $(2^n \times 2^n)$*-matrices: For every bit vector* $\overline{a} \in \{0, 1\}^k$*, define the circuit* $C_{\overline{a}} = C(\overline{a}, \overline{x}, \overline{y}, \overline{z})$ *and the matrix* $M_{\overline{a}} = M_{C_{\overline{a}},2}$*. Finally, let* $M_C = \prod_{\overline{a} \in \{0,1\}^k} M_{\overline{a}}$ *be the product of all these matrices.*

Lemma 4. *The function* $C(\overline{w}, \overline{x}, \overline{y}, \overline{z}) \mapsto M_C$ *belongs to* $\mathsf{FPSPACE}$.

Lemmas 3 and 4 yield the upper complexity bounds in the following theorem. For the lower bounds we use again succinct versions of Toda's techniques from [27], similar to the proof of Thm. 6.

Theorem 8. *The following holds:*

(1) The function $(G, n) \mapsto (\mathsf{val}(G)^n)_{1,1}$ *with* G *an MTDD over* $\mathbb{N}$ *(resp.* $\mathbb{Z}$*) and* n *a unary encoded number is complete for* $\#\mathsf{P}$ *(resp.,* GapP*).*
(2) The function $(G, n) \mapsto (\mathsf{val}(G)^n)_{1,1}$ *with* G *an MTDD over* $\mathbb{N}$ *(resp.* $\mathbb{Z}$*) and* n *a binary encoded number is* $\#\mathsf{PSPACE}$*-complete (resp.,* $\mathsf{GapPSPACE}$*-complete).*

By Thm. 8, there is no polynomial time algorithm that computes for a given MTDD G and a unary number n a boolean circuit (or even an MTDD_+) for the power $\mathsf{val}(G)^n$, unless $\#\mathsf{P} = \mathsf{FP}$.

By [27] and Thm. 8, the complexity of computing a specific entry of a matrix power A^n covers three different counting classes, depending on the representation of the matrix A and the exponent n (let us assume that A is a matrix over $\mathbb{N}$):

– $\#\mathsf{L}$-complete, if A is given explicitly and n is given unary.
– $\#\mathsf{P}$-complete, if A is given by an MTDD and n is given unary.
– $\#\mathsf{PSPACE}$-complete, if A is given by an MTDD and n is given binary.

Let us also mention that in [6, 12, 22] the complexity of evaluating iterated matrix products and matrix powers in a fixed dimension is studied. It turns out that multiplying a sequence of $(d \times d)$-matrices over $\mathbb{Z}$ in the fixed dimension $d \geq 3$ is complete for the class GapNC^1 (the counting version of the circuit complexity class NC^1) [6]. It is open whether the same problem for matrices over $\mathbb{N}$ is complete for $\#\mathsf{NC}^1$. Moreover, the case $d = 2$ is open too. Matrix powers for matrices in a fixed dimension can be computed in TC^0 (if the exponent is represented in unary notation) using the Cayley-Hamilton theorem [22]. Finally, multiplying a sequence of $(d \times d)$-matrices that is given succinctly by a boolean circuit captures the class $\mathsf{FPSPACE}$ for any $d \geq 3$ [12].

For the problem, whether a power of an MTDD-encoded matrix is zero (a variant of the classical mortality problem) we can finally show the following:

Theorem 9. *It is* coNP*-complete (resp.,*PSPACE*-complete) to check whether* $\mathsf{val}(G)^m$ *is the zero matrix for a given MTDD G and a unary (resp., binary) encoded number* m*.*

7 Conclusion and future work

We studied algorithmic problems on matrices that are given by multi-terminal decision diagrams enriched by the operation of matrix addition. Several important matrix problems can be solved in polynomial time for this representation, e.g., equality checking, computing matrix entries, matrix multiplication, computing the trace, etc. On the other hand, computing determinants, matrix powers, and iterated matrix products are computationally hard. For further research, it should be investigated whether the polynomial time problems, like equality test, belong to NC.

References

1. C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theor. Comput. Sci.*, 107:3–30, 1993.
2. M. Beaudry and M. Holzer. The complexity of tensor circuit evaluation. *Computational Complexity*, 16(1):60–111, 2007.
3. P. Berman, M. Karpinski, L. L. Larmore, W. Plandowski, and W. Rytter. On the complexity of pattern matching for highly compressed two-dimensional texts. *J. Comput. Syst. Sci.*, 65:332–350, 2002.
4. A. Bertoni, C. Choffrut, and R. Radicioni. Literal shuffle of compressed words. In *Proc. IFIP TCS 2008*, volume 273 of *IFIP*, 87–100. Springer, 2008.
5. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.
6. H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57(2):200–212, 1998.
7. C. Damm, M. Holzer, and P. McKenzie. The complexity of tensor calculus. *Computational Complexity*, 11(1-2):54–89, 2002.

8. D. Eppstein, M. T. Goodrich, and J. Z. Sun. Skip quadtrees: Dynamic data structures for multidimensional point sets. *Int. J. Comput. Geometry Appl.*, 18:131–160, 2008.
9. J. Feigenbaum, S. Kannan, M. Y. Vardi, and M. Viswanathan. The complexity of problems on graphs represented as obdds. *Chicago J. Theor. Comput. Sci.*, 1999.
10. H. Fujii, G. Ootomo, and C. Hori. Interleaving based variable ordering methods for ordered binary decision diagrams. In *Proc. ICCAD 1993*, 38–41. IEEE Computer Society, 1993.
11. M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, 1997.
12. M. Galota and H. Vollmer. Functions computable in polynomial space. *Inf. Comput.*, 198(1):56–70, 2005.
13. H. Galperin and A. Wigderson. Succinct representations of graphs. *Inform. and Control*, 56:183–198, 1983.
14. B. Grenet, P. Koiran, and N. Portier. On the complexity of the multivariate resultant. *J. Complexity*, 29(2): 142–157, 2013.
15. R. E. Ladner. Polynomial space counting problems. *SIAM J. Comput.*, 18:1087–1097, 1989.
16. H. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.
17. M. Lohrey. Leaf languages and string compression. *Inf. Comput.*, 209:951–965, 2011.
18. M. Lohrey. Algorithmics on SLP-compressed strings: a survey. *Groups Complex. Cryptol.*, 4:241-299, 2012.
19. M. Lohrey and M. Schmidt-Schauß. Processing Succinct Matrices and Vectors. *arXiv*, 2014. http://arxiv.org/abs/1402.3452
20. G. Malod. Succinct algebraic branching programs characterizing non-uniform complexity classes. In *Proc. FCT 2011*, *LNCS 6914*, 205–216. Springer, 2011.
21. C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design: OBDD - Foundations and Applications*. Springer, 1998.
22. C. Mereghetti and B. Palano. Threshold circuits for iterated matrix product and powering. *ITA*, 34(1):39–46, 2000.
23. W. Plandowski. Testing equivalence of morphisms in context-free languages. In *Proc. ESA 94*, *LNCS 855*, 460–470. Springer, 1994.
24. H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
25. A. Storjohann and T. Mulders. Fast algorithms for for linear algebra modulo N. In *Proc. ESA 98*, *LNCS 1461*, 139–150. Springer, 1998.
26. M. A. Taĭclin. Algorithmic problems for commutative semigroups. *Dokl. Akda. Nauk SSSR*, 9(1):201–204, 1968.
27. S. Toda. Counting problems computationally equivalent to computing the determinant. Technical Report CSIM 91-07, Tokyo University of Electro-Communications, 1991.
28. S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20:865–877, 1991.
29. L. G. Valiant. Completeness classes in algebra. In *Proc. STOC 1979*, 249–261. ACM, 1979.
30. H. Veith. How to encode a logical structure by an OBDD. In *Proc. 13th Annual IEEE Conference on Computational Complexity*, 122–131. IEEE Computer Society, 1998.
31. I. Wegener. The size of reduced OBDD's and optimal read-once branching programs for almost all boolean functions. *IEEE Trans. Computers*, 43(11):1262–1269, 1994.