Electronic Colloquium on Computational Complexity, Report No. 108 (2018)

Small Normalized Boolean Circuits for Semi-disjoint Bilinear Forms Require Logarithmic Conjunction-depth*

Andrzej Lingas

Department of Computer Science, Lund University
Box 118, 22100 Lund, Sweden
`Andrzej.Lingas@cs.lth.se`

Abstract

We consider *normalized* Boolean circuits that use binary operations of disjunction and conjunction, and unary negation, with the restriction that negation can be only applied to input variables. We derive a lower bound trade-off between the size of normalized Boolean circuits computing Boolean semi-disjoint bilinear forms and their conjunction-depth (i.e., the maximum number of and-gates on a directed path to an output gate). In particular, we show that any normalized Boolean circuit of at most $\epsilon \log n$ conjunction-depth computing the n-dimensional Boolean vector convolution has $\Omega(n^{2-4\epsilon})$ and-gates. For Boolean matrix product, we derive even a stronger lower-bound trade-off. Instead of conjunction-depth we use the negation-dependent conjunction-depth, where one counts only and-gates whose each direct predecessor has a (not necessarily direct) predecessor representing a negated input variable. We show that if a normalized Boolean circuit of at most $\epsilon \log n$ negation-dependent conjunction-depth computes the $n \times n$ Boolean matrix product then the circuit has $\Omega(n^{3-2\epsilon})$ and-gates. We complete our lower-bound trade-offs for the Boolean convolution and matrix product with upper-bound trade-offs of similar form yielded by the known fast algebraic algorithms.

Keywords and phrases Boolean circuits, semi-disjoint bilinear form, Boolean vector convolution, Boolean matrix product

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

1.1 Background

A set F of polynomials over a semi-ring is a *form* (in case of the Boolean semi-ring, just a set of monotone Boolean functions). F is a *semi-disjoint bilinear form* if it is defined on the set of variables $X \cup Y$ and the following properties hold.

1. For each polynomial Q in F and each variable $z \in X \cup Y$, there is at most one monomial (in the Boolean case, called a prime implicant [24]) of Q containing z.
2. Each monomial of a polynomial in F consists of exactly one variable in X and one variable in Y.
3. The sets of monomials of polynomials in F are pairwise disjoint.

The n-dimensional vector convolution and the $n \times n$ matrix product are important and popular examples of semi-disjoint bilinear forms (for the convolution, $|X| = |Y| = n$ and

* This is an extended version of that to appear in CCC 2018 proc. The extension includes an improved lower bound trade-off for Boolean matrix product.

© Andrzej Lingas;
licensed under Creative Commons License CC-BY
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ISSN 1433-8092

$|F| = 2n - 1$ while for the matrix product, $|X| = |Y| = |F| = n^2$). Both semi-disjoint bilinear forms in the arithmetic and Boolean case have a wide range of fundamental applications, for instance, in stringology (see, e.g., [6]) and graph algorithms (see, e.g., [27]).

Two $n \times n$ integer matrices can be arithmetically multiplied using $O(n^3)$ additions and multiplications following the definition of matrix product. This is optimal if neither other operations nor negative constants are allowed [13, 16, 20]. If additionally subtraction or negative constants are allowed then the so-called fast matrix multiplication algorithms can be implemented using $O(n^\omega)$ operations [7, 22, 26], where $\omega < 3$. They rely on algebraic equations following from the possibility of term cancellation (for a study on the power of arithmetic term cancellation see [23]). Le Gall and Vassilevska Williams have recently shown the exponent ω of fast matrix multiplication to be smaller than 2.373 in [7, 26]. The fast arithmetic algorithms run on $0-1$ matrices yield the same asymptotic upper time bounds for the $n \times n$ Boolean matrix multiplication. On the other hand, Raz proved that if only addition, multiplication and products with constants of absolute value not exceeding one are allowed then $n \times n$ matrix multiplication requires $\Omega(n^2 \log n)$ operations [17].

Similarly, the arithmetic convolution of two n-dimensional vectors can be computed using $O(n^2)$ additions and multiplications. Next, the convolution of two n-dimensional vectors over a commutative ring with the so-called principal n-th root of unity can be computed via Fast Fourier Transform using $O(n \log n)$ operations of the ring. The n-dimensional Boolean vector convolution admits an algorithm using $O(n \log^2 n \log \log n)$ Boolean operations by reduction to the fast integer multiplication algorithm from [21] in turn relying on Fast Fourier Transform [6].

It is well known that for uniform problems, their Boolean circuit complexity corresponds up to logarithmic factors to their Turing complexity [24]. Unfortunately, until today no super-linear lower bounds on the size of circuits using binary and unary Boolean operations forming a complete Boolean basis are known for natural problems [24]. On the other hand, such lower bounds are known in case of monotone Boolean circuits that use only the binary operations of disjunction and conjunction [1, 2, 3, 11, 13, 14, 15, 16, 18, 24, 25]. In particular, Alon and Boppana showed by refining Razborov's breakthrough method [18] that the (m, s)-clique, i.e., the problem of determining if a graph on m vertices includes a complete subgraph on s vertices, requires monotone Boolean circuits of $2^{\sqrt{m}}$ size [1].

There exist interesting connections between the general Boolean circuit complexity and the monotone one [4]. In particular, any Boolean circuit using disjunctions, conjunctions and negations can be easily transformed into a Boolean circuit using the same operations, where negations are applied solely to input variables. The transformations follows from de Morgan's laws and keep the circuit size within a factor 2. In other words, one can see such Boolean circuits as monotone Boolean circuits with respect to the input literals, i.e., input variables and their negations. We shall term Boolean circuits in the latter form *normalized*.

In case of the $n \times n$ Boolean matrix product, almost tight or even tight lower bounds of the form $\Omega(n^3)$ for the monotone circuit complexity were presented in a series of papers [13, 14, 16] more than three decades ago. The best known (in the literature) lower bound on monotone Boolean circuit complexity for the n-dimensional Boolean vector convolution is $\Omega(n^2/\log^6 n)$ due to Grinchuk and Sergeev [8]. It improves on the previously best $n^{3/2}$ lower bound due to Weiss [25] and an earlier best $n^{4/3}$ lower bound due to Blum [3]. The lower bounds of Weiss, Grinchuk and Sergeev are on the number of disjunctions while that of Blum is on the number of conjunctions.

Furthermore, Lingas studied the complexity of monotone Boolean circuits for Boolean semi-disjoint bilinear forms under various monotone circuit restrictions in [12]. In particular, he

author	year	lower bound
N. Pippinger and L.G. Valiant [15]	1976	$\Omega(n \log n)$
E.A. Lamagna [11]	1979	$\Omega(n \log n)$
N. Blum [3]	1980	$n^{4/3}$ conjunctions
R. Weiss [25]	1981	$n^{3/2}$ disjunctions
M.I. Grinchuk and I.S. Sergeev [8]	2011	$\Omega(n^2/\log^6 n)$ disjunctions

Table 1 Lower bounds on the monotone Boolean circuit complexity for the n-dimensional Boolean vector convolution in a historical perspective.

considered monotone Boolean circuits of bounded conjunction-depth, i.e., bounded maximum number of and-gates on any single directed path to an output gate in the monotone circuit. He showed that any monotone Boolean circuit of conjunction-depth at most d computing a Boolean semi-disjoint form with p prime implicants has to have at least $p/2^{2d}$ and-gates. As a corollary, he obtained the $\Omega(n^{2-2\epsilon})$ lower bound on the size of any monotone Boolean circuit of $\epsilon \log n$-bounded conjunction-depth computing the n-dimensional Boolean vector convolution.

1.2 Our contributions

Surprisingly enough, we can derive a lower-bound trade-off between the circuit size and its conjunction-depth for normalized Boolean circuits computing semi-disjoint bilinear forms similar to that for monotone Boolean circuits from [12].

More exactly, we show that any normalized Boolean circuit of conjunction-depth at most d computing a Boolean semi-disjoint form with p prime implicants has to have $\Omega(p/2^{4d})$ and-gates. As a corollary, we obtain the $\Omega(n^{2-4\epsilon})$ lower bound on the size of any normalized Boolean circuit of $\epsilon \log n$-bounded conjunction-depth computing the n-dimensional Boolean vector convolution.

For Boolean matrix product, we present even a stronger lower-bound trade-off additionally relying on the tight lower bound on the monotone complexity of this product [13, 14, 16]. Instead of conjunction-depth we use the negation-dependent conjunction-depth, where one counts only and-gates whose each direct predecessor has a (not necessarily direct) predecessor representing a negated input variable. We show that if a normalized Boolean circuit of at most $\epsilon \log n$ negation-dependent conjunction-depth computes the $n \times n$ Boolean matrix product then the circuit has $\Omega(n^{3-2\epsilon})$ and-gates.

We complete our lower-bound trade-offs with upper-bounds trade-offs of similar form yielded by the aforementioned fast algebraic algorithms. We observe that there is a positive constant $c \le 1$ such that for any $\epsilon \in (0, \frac{1}{c})$, the n-dimensional Boolean vector convolution can be computed by a normalized Boolean circuit of $\epsilon \log n$-bounded conjunction-depth and $O(n^{2-c\epsilon} + n \log^2 n \log \log n)$ size. Similarly, there is a positive constant $c \le 1$ such that for any $\epsilon \in (0, \frac{1}{c})$, the $n \times n$ Boolean matrix product can be computed by a normalized Boolean circuit of $\epsilon \log n$-bounded conjunction-depth and $O(n^{3-(3-\omega)c\epsilon})$ size.

1.3 Motivations

Our primary motivation is the very weak progress in deriving non-trivial lower bounds on the size of Boolean circuits using disjunctions, conjunctions and negations to compute explicit Boolean functions computable in polynomial time, since the 70s (from $3n$ [19] to almost $5n$ [9, 10]). For this reason, trade-offs between structural parameters and the size for the aforementioned circuits computing explicit functions should be of interest.

We believe that the conjunction-depth of a normalized Boolean circuit computing a Boolean form whose prime implicants (see Preliminaries) consist of relatively few literals is an interesting structural characteristic. (For not-necessarily normalized Boolean circuit using disjunctions, conjunctions and negations, the concept of conjunction-depth does not make sense since conjunctions can be eliminated by composing negations with disjunctions via de Morgan's laws. Also, there are trivial examples of Boolean functions that require a large conjunction-depth in normalized circuits. E.g., the function given by $\neg \bigvee_{i=1}^{n} x_i \equiv \bigwedge_{i=1}^{n} \bar{x}_i$ obviously requires $\log n$ conjunction-depth. The reason is that it has a prime implicant consisting of n literals.)

Observe that each prime implicant of the functions occurring in semi-disjoint bilinear forms consists solely of two literals. Hence, any semi-disjoint bilinear form admits a normalized (in fact, monotone) Boolean circuit having conjunction-depth 1 and the number of gates proportional to the total number of prime implicants (see also Fact 1).

Our lower-bound trade-offs showing that in order to decrease the size of normalized Boolean circuits computing a semi-disjoint bilinear form one has to increase their conjunction-depth should be of interest. Our upper-bound trade-offs imply that normalized Boolean circuits of even sub-logarithmic conjunction-depth for Boolean vector convolution or Boolean matrix product have substantially smaller size than their monotone counterparts of unbounded conjunction-depth.

1.4 Paper structure

In Preliminaries, we introduce basic definitions and notation. In Section 3, we present three lemmata on restricted normalized circuits computing a Boolean form. In Section 4, we show our lower-bound trade-offs for semi-disjoint bilinear forms and the stronger lower-bound trade-off for Boolean matrix product. In Section 5, we present our upper-bound trade-offs. We conclude with final remarks where among other things a related result is discussed.

2 Preliminaries

For two Boolean n-dimensional vectors $a = (a_0, ..., a_{n-1})$ and $b = (b_0, ..., b_{n-1})$, their convolution is a vector $c = (c_0, ..., c_{2n-2})$, where $c_i = \bigvee_{l=\max\{i-n+1,0\}}^{\min\{i,n-1\}} a_l \wedge b_{i-l}$ for $i = 0, ..., 2n-2$.

A *literal* is a variable or the negation of a variable.

A *(Boolean) circuit* is a finite directed acyclic graph with the following properties:

1. The indegree of each vertex (termed gate) is either 0, 1 or 2.
2. The source vertices (i.e., vertices with indegree 0 called input gates) are labeled by elements in some set of literals, i.e., variables and their negations, and the Boolean constants 0, 1.
3. The vertices of indegree 2 are labeled by elements of the set $\{and, or\}$ and termed and-gates and or-gates, respectively.
4. The vertices of indegree 1 are labeled by *negation* and termed negation-gates.

A Boolean circuit is *normalized* if it does not use negation-gates. A Boolean circuit is *monotone* if it is normalized and it does not use negated variables.

The *size* of a Boolean circuit C is the total number of non-input gates in C while the *depth* of C is the maximum length of a directed path in C. Furthermore, C is of *conjunction-depth* d if the number of and-gates on any directed path in C does not exceed d. We say that a gate in C is *negation dependent* if an input gate labeled by a negated variable is its (not necessarily direct) predecessor. C is of *negation-dependent conjunction-depth* d if on any directed path in C the number of and-gates whose each direct predecessor is negation dependent does not exceed d.

With each gate g of a normalized Boolean circuit, we associate a set $T(g)$ of terms in a natural way. Thus, with each input gate, we associate the singleton set consisting of the corresponding variable, negated variable or constant. Next, with an or-gate, we associate the union of the sets associated with its direct predecessors. Finally, with an and-gate g, we associate the set of concatenations t_1t_2 of all pairs of terms t_1, t_2, where $t_i \in T(g_i)$ and g_i stands for the i-th direct predecessor of g for $i = 1, 2$. The function computed at the gate g is the disjunction of the functions (called monoms) represented by the terms in $T(g)$. The monom represented by a term t is obtained by replacing concatenations in t with conjunctions, respectively. A term in $T(g)$ is a *zero-term* if it contains the Boolean constant 0 or a variable and its negation. Clearly, a zero-term represents the Boolean constant 0.

A form composed of k Boolean functions is computed by a Boolean circuit if there are k distinguished gates (called output gates) computing the k functions.

A term (an output term, respectively) of a circuit C is a term in $T(g)$ for some gate (output gate, respectively) g of C.

An *implicant* of a Boolean form F is a conjunction of some variables and/or some negated variables of F and/or Boolean constants (monom) such that there is a function belonging to F which is true whenever the conjunction is true. If the conjunction includes the Boolean 0 or a variable x and its negation $\bar{x}$ then it is a *trivial implicant* of (any) F.

A non-trivial implicant of F that is minimal with respect to included literals is a *prime implicant* of F.

The following upper bound is straight-forward.

▶ Fact 1. [12] Each Boolean semi-disjoint bilinear form composed of l functions on $x_0, ..., x_{n-1}$ and $y_0, ..., y_{n-1}$ with p prime implicants in total can be computed by a monotone Boolean circuit of conjunction-depth 1 with $p \le n^2$ and-gates and $p - l$ or-gates.

Proof. First, we use p and-gates to compute each prime implicant x_iy_j separately. Then, we form l disjoint or-unions of the prime implicants corresponding to the l functions of the bilinear form using $p - l$ or-gates. ◀

3 Lemmata on Normalized Circuits

Recall that the monom represented by a term t is obtained by replacing concatenations in t with conjunctions, respectively. We shall say that an implicant (in particular, a prime implicant) of a function f_g computed at the gate g is represented by a single term in $T(g)$ if there is a term $t \in T(g)$ such that the monom represented by t is equivalent to the implicant.

In the following two lemmata, we shall show that if the output terms of a normalized circuit computing a form contain a bounded number of different negated variables, we can obtain a situation somewhat similar to that in monotone circuits, where each prime implicant of an output function has to be represented by a single output term. Namely, we can zero some part of variables such that in the resulting circuit, a substantial fraction of the prime implicants of the form is represented by single output terms.

▶ **Lemma 1.** *Let C be a normalized Boolean circuit computing a form F whose prime implicants do not contain negated variables. For each prime implicant of the function $f_o \in F$ computed at the output gate o of C, there is a term in $T(o)$ representing the (whole) prime implicant or a conjunction of the prime implicant with solely negated variables.*

Proof. Consider a prime implicant of f_o. Assign the Boolean 1 to the variables in the prime implicant and the Boolean 0 to all remaining variables in F. Under this assignment, the value of f_o should be 1. Hence, since each term in $T(o)$ has to represent an implicant of f_o, there must exist a term in $T(o)$ representing the whole prime implicant or a conjunction of the prime implicant with solely negated variables. ◀

▶ **Lemma 2.** *Let C be a normalized Boolean circuit computing a form F with p prime implicants. Suppose that each prime implicant of F is composed of q (not negated) variables and each output term of C contains at most k distinct negated variables. Let $0 < \beta < 1$. There is a subset of the set of variables of F such that after setting them to the Boolean 0 there are at least $p\beta^q(1-\beta)^k$ prime implicants of F represented by single output terms of the circuit C' resulting from C. Note that the circuit C' computes a form F' whose set of prime implicants is a subset of the set of prime implicants of F.*

Proof. Set each variable of F to the Boolean constant 0 with probability $1-\beta$ uniformly at random. Consider any prime implicant $x_{i_1}...x_{i_q}$ of F. The probability that none of $x_{i_1},...,x_{i_q}$ is set to 0 is β^q. By Lemma 1, there is a set of $0 \le l \le k$ negated variables whose conjunction with $x_{i_1}...x_{i_q}$ is represented by an output term of C. The probability that each of these negated l variables is set to 0 is at least $(1-\beta)^k$. Hence, the expected number of prime implicants of the form computed by the resulting circuit and represented by single output terms in this circuit is at least $p\beta^q(1-\beta)^k$. It follows that there is a subset of the set of variables satisfying the requirements of the lemma.

Finally, F can be represented by the disjunction of its prime implicants. After the setting of the variables in the aforementioned subset to the Boolean 0 the prime implicants containing them disappear and the form F' can be represented by the disjunction of the remaining prime implicants of F. The remaining prime implicants form the set of prime implicants of F'. ◀

The final lemma in this section is pretty obvious.

▶ **Lemma 3.** *Each term, in particular, each output term of a normalized Boolean circuit of d-bounded conjunction-depth includes at most 2^d literals. Similarly, each term of a normalized Boolean circuit of d-bounded negation-dependent conjunction-depth includes at most 2^d negated variables.*

Proof. An and-gate can at most double the number of literals in single terms while an or-gate does not increase it. Hence, by induction on the maximum number d of and-gates on a path from an input gate to a gate g in the circuit, any term in $T(g)$ includes at most 2^d literals. This proves the first part. Observe that an and-gate whose direct predecessor is not reachable by any directed path from input gates labeled with negated variables cannot increase the number of negated variables in single terms in the circuit. Hence, the proof of the second part is analogous. ◀

4 Lower-bound Trade-offs (main results)

4.1 Semi-disjoint bilinear forms

In monotone circuits, where negation is not used, each prime implicant of a function computed at a gate h has to be represented by a single term in $T(h)$ (there might be several such terms and many other terms having subterms representing the prime implicant). This is not the case in normalized circuits generally. There, we can associate to a prime implicant of the function the set of all terms in $T(g)$ representing a conjunction of the prime implicant with an additional conjunction of literals (e.g., x_iy_j could be represented by $\{x_iy_jx_k,\ x_iy_j\bar{x_k}\}$). Interestingly, the disjunction of the aforementioned additional conjunctions does not have to be always true (e.g., $x \vee y$ could be computed by $x\bar{y} \vee y$ so the prime implicant x would be represented just by $\{x\bar{y}\}$).

First, we shall show how a restriction on the maximum number of distinct literals which occur in an output term of a normalized Boolean circuit computing a Boolean semi-disjoint form can be used to derive a non-trivial lower bound on the number of and-gates in the circuit.

▶ **Lemma 4.** *Let C be a normalized Boolean circuit computing a semi-disjoint bilinear form F on the variables $x_0, ..., x_{n-1}$ and $y_0, ..., y_{n-1}$. Suppose that for each output gate o in C, each term in $T(o)$ contains at most k different literals. Let h be a gate connected by directed paths with some output gates in C such that the function computed at h has prime implicants $z_{q_1}, ..., z_{q_{l(h)}}$ which are single (not negated) variables represented by single terms in $T(h)$, and possibly some other prime implicants. The inequality $l(h) \le k$ holds or h can be replaced by the Boolean constant 1.*

Proof. Consider a directed path P connecting h with some output gate o in C. At the output gate o, for each z_{q_r}, $1 \le r \le l(h)$, any single term $t(z_{q_r}) \in T(h)$ representing z_{q_r} has to appear in terms $t_1t(z_{q_r})t_2$ in the associated set $T(o)$ (see Preliminaries) such that t_1t_2 is a concatenation (i.e., conjunction) of some terms added by subsequent and-gates on P and $t_1t(z_{q_r})t_2$ represents an implicant of the function f_o computed at o. In general, $t(z_{q_r})$ may include several occurrences of z_{q_r} and the Boolean 1, for simplicity we may assume w.l.o.g. that $t(z_{q_r}) = z_{q_r}$. (The reason of having t_1, t_2 instead of a single term t is that syntactically the concatenations can come from both sides.)

Suppose that there is such a t_1t_2, where $t_1zt_2 \in T(o)$ for some $z \in \{z_{q_r} | 1 \le r \le l(h)\}$, which does not represent an implicant of f_o. It follows from the definition of t_1t_2 that for any $z \in \{z_{q_r} | 1 \le r \le l(h)\}$, the term t_1zt_2 also appears in the set $T(o)$ of terms associated with the output gate o and consequently it has to represent an implicant of f_o as well. Therefore, for each such a z, either t_1t_2 contains $\bar{z}$ or t_1t_2 contains the unique "mate" variable z' for which zz' is a prime implicant of f_o. Note that if z is an x-variable then z' is a y-variable and *vice versa*. Set H to $\{z_{q_1}, ..., z_{q_{l(h)}}\}$. E.g., the case that t_1t_2 contains $\bar{z}$ could happen if there were some other variables $z" \in H$ for which $t_1z"t_2$ are not trivial implicants of f_o but t_1zt_2 becomes a trivial implicant because it contains both z and $\bar{z}$.

Consider the mapping of each $z \in H$ either to the z' in t_1t_2 (which must be the unique "mate" among the prime implicants of f_o) or to the $\bar{z} \in t_1t_2$. Clearly, all the $\bar{z}$ for $z \in H$ are distinct negated variables. Because no two elements of H have the same mate among the prime implicants of f_o, no two of the z' for $z \in H$ can be the same. Finally, the mates z' are single not negated variables. It follows that the mapping is one-to-one. We infer that $l(h) \le k$.

On the contrary, if each such term $t = t_1t_2$ for each path P connecting h with any output gate o, represents an implicant of f_o then on each P we could connect the successor of the start vertex h with the Boolean constant 1 instead of h and the output gate o still would output f_o. To see this observe that then each $u \in T(h)$ is a part of the terms of the form $t_1 u t_2$ in $T(o)$, where $t_1 t_2$ represents an implicant of the function f_o. Since this holds for each successor of h, this gate can be replaced by the constant 1. ◀

For an and-gate g in a normalized Boolean circuit C computing a semi-disjoint bilinear form F, S_g will denote the set of prime implicants s of F such that:

1. s is a prime implicant of the function computed at g that is represented by a single term in $T(g)$,
2. s is not a prime implicant of the function computed at either of the two direct predecessors h of g that is represented by a single term in $T(h)$, and
3. there is a directed path connecting g with the output gate computing the function whose prime implicant is s.

▶ **Lemma 5.** *Let C be a normalized Boolean circuit computing a semi-disjoint bilinear form F. Suppose that for each output gate o in C, each term in $T(o)$ contains at most k different literals. Next, suppose that C does not contain any and-gate that could be replaced by the Boolean 1 so the resulting circuit would still compute F. For any and-gate g in C, the inequality $|S_g| \le k^2$ holds.*

Proof. We may assume w.l.o.g. $|S_g| \ge 1$. It follows that at least for one of the direct predecessor gates h of g, the function computed at h has at least $\sqrt{|S_g|}$ single variable prime implicants represented by single terms in $T(h)$. By Lemma 4, we infer that either $\sqrt{|S_g|} \le k$ or the gate h can be replaced by the constant 1. The latter possibility contradicts the lemma assumptions. ◀

▶ **Theorem 6.** *Let C be a normalized Boolean circuit computing a semi-disjoint bilinear form F with p prime implicants. Suppose that each output term of C contains at most k distinct literals. The circuit C has at least $\frac{p}{k^4}(1-\frac{1}{k})^{k-2}$ and-gates.*

Proof. We shall apply Lemma 2 with $\beta = \frac{1}{k}$ and $q = 2$, and k set to $k-2$ in the lemma, to the circuit C. Let C' be the circuit resulting from C by zeroing the subset of variables specified in this lemma. Note that the output terms of C' still contain at most k different literals, and that C' computes a semi-disjoint bilinear form F' whose prime implicants are prime implicants of F. Among the prime implicants of F', at least $\frac{p}{k^2}(1-\frac{1}{k})^{k-2}$ are represented by single output terms by Lemma 2.

Iterate the following steps starting from the circuit C'. Whenever the current circuit contains an and-gate or an or-gate h that can be replaced by the Boolean constant 1 without affecting the functions computed at the output gates, replace h by 1. By induction on the number of iterations, the new circuit still computes the same bilinear form F'. Also, the number of prime implicants of F' represented by single output terms does not drop and each output term of the new circuit contains at most k literals.

Since the circuit C' is finite and each iteration eliminates at least one gate, after a finite number of iterations, we obtain a circuit C'' sharing the aforementioned properties, not containing any and-gate or or-gate that could be replaced by 1, and still computing F'. It follows from Lemma 4 that C'' does not have any gate h such that the function computed at h contains more than k single-variable prime implicants represented by single terms in $T(h)$.

Let S be the set of at least $\frac{p}{k^2}(1-\frac{1}{k})^{k-2}$ prime implicants of F' represented by single output terms of C''. Recall the definition of the set S_g of prime implicants of a form for an and-gate g given before Lemma 5. For each $s \in S$, there must be at least one and-gate g

of C'' such that $s \in S_g$. (To find such a gate g start from the output gate computing the function of F' for which s is a prime implicant represented by a single term and iterate the following steps: check if the current gate g satisfies $s \in S_g$, if not go to the direct predecessor of g that computes a function having s as a prime implicant represented by a single term.) By the latter lemma, we have $|S_g| \leq k^2$. Hence, C'', and consequently C' and C, have at least $|S|/k^2 \geq \frac{p}{k^2}(1-\frac{1}{k})^{k-2}/k^2 \geq \frac{p}{k^4}(1-\frac{1}{k})^{k-2}$ and-gates since $|S| \geq \frac{p}{k^2}(1-\frac{1}{k})^{k-2}$. ◀

By combining Theorem 6 with Lemma 3, we obtain our first main result.

▶ **Theorem 7.** *Let C be a normalized Boolean circuit of conjunction-depth at most d computing a semi-disjoint bilinear form F with p prime implicants. The circuit C has at least $\frac{p}{2^{4d}}(1-\frac{1}{2^d})^{2^d-2}$ and-gates.*

Observe that the n-dimensional Boolean vector convolution has $\Theta(n^2)$ prime implicants while the $n \times n$ Boolean matrix product has $\Theta(n^3)$ prime implicants.

▶ **Corollary 8.** *For $\epsilon > 0$, any normalized Boolean circuit of $\epsilon \log n$-bounded conjunction-depth that computes the n-dimensional Boolean vector convolution has $\Omega(n^{2-4\epsilon})$ and-gates.*

▶ **Corollary 9.** *For $\epsilon > 0$, any normalized Boolean circuit of $\epsilon \log n$-bounded conjunction-depth that computes the $n \times n$ Boolean matrix product has $\Omega(n^{3-4\epsilon})$ and-gates.*

4.2 A stronger lower-bound trade-off for Boolean matrix product

Recall Lemma 2. We can provide a stronger lower bound trade-off for Boolean matrix product by using the following simple complementary lemma.

▶ **Lemma 10.** *Let the normalized Boolean circuits C, C' and the forms F, F' computed by them be defined as in Lemma 2. Let F'' be a form having the following properties: for each $f'' \in F''$ different from a constant there is a distinct $f \in F$ such that the prime implicants of f'' are implicants of f and all prime implicants of f represented by single output terms in C' are also prime implicants of f'', Suppose that any monotone Boolean circuit computing such form F'' has at least u and-gates and at least w or-gates. Then the circuits C, C' have also at least u and-gates and at least w or-gates.*

Proof. We can transform C' to a monotone Boolean circuit C'' computing a form F'' having the properties stated in the lemma as follows. We substitute the Boolean constant 0 for all negated variables. Thus, we replace each directed edge from an input gate labeled with a negated variable to a and-gate or an or-gate by a directed edge from the gate labeled with 0 to the and-gate or or-gate. The substitution turns all output terms of C' including a negated variable to an output term equivalent to the Boolean 0, but it leaves the output terms containing only not negated variables unchanged. Hence, it follows from the properties of the circuit C' and the form F' that the form F'' computed by the resulting monotone circuit C'' has the properties stated in the lemma. By our assumptions, the circuit C'', and consequently also C' and C have at least u and-gates and at least w or-gates. ◀

If F stands for Boolean matrix product then tight lower bounds on the monotone circuit complexity of F'' follow from the known tight lower bounds on the monotone circuit complexity of Boolean matrix product.

▶ Fact 2. [13, 14, 16] Any monotone Boolean circuit computing the $n \times n$ Boolean matrix product has at least n^3 and-gates.

▶ **Corollary 11.** *Suppose that for each function h in a form H there is a distinct function f in the $n \times n$ Boolean matrix product form such that each prime implicant of h is an implicant of f. Let p be the total number of prime implicants of H that are also prime implicants of the $n \times n$ Boolean matrix product. Any monotone Boolean circuit computing H has at least p and-gates.*

Proof. Suppose that there is a monotone Boolean circuit that computes H using a number of and-gates smaller than p We can augment this circuit with $n^3 - p$ and-gates forming the prime implicants of the Boolean matrix product that are not prime implicants of H and no more than $n^2(n-1)$ or-gates in order to obtain a monotone circuit computing the Boolean matrix product. Since the resulting circuit has less than n^3 and-gates, we obtain a contradiction with Fact 2. ◀

In the resulting lower bound trade-off, the number of not negated variables in output terms does not have to be limited.

▶ **Theorem 12.** *Let C be a normalized Boolean circuit computing the $n \times n$ Boolean matrix product. Suppose that each output term of C contains at most k distinct negated variables. The circuit has at least $\frac{n^3}{k^2}(1-\frac{1}{k})^k$ and-gates.*

Proof. Set F to the $n \times n$ Boolean matrix product, q to 2, and β to $\frac{1}{k}$ in Lemma 2. Now it is sufficient to combine Lemma 10 with Corollary 11. Note that the form F'' in the latter lemma shares at least $\frac{n^3}{k^2}(1-\frac{1}{k})^k$ prime implicants with F and that it has the properties of the form H in the corollary. ◀

▶ **Corollary 13.** *Let C be a normalized Boolean circuit computing the $n \times n$ Boolean matrix product. Suppose that C is of negation-dependent conjunction-depth d. The circuit C has at least $\frac{n^3}{2^{2d}}(1-\frac{1}{2^d})^{2^d}$ and-gates. In particular, if $d = \epsilon \log n$ then C has $\Omega(n^{3-2\epsilon})$ and-gates.*

Proof. By Lemma 3, each output term of the circuit C contains at most 2^d distinct negated variables. By Theorem 12, C has at least $\frac{n^3}{2^{2d}}(1-\frac{1}{2^d})^{2^d}$ and-gates. Consequently, if $d = \epsilon \log n$ then it has $\Omega(n^{3-2\epsilon})$ and-gates. ◀

5 Upper-bound Trade-offs

The fast algebraic algorithms for arithmetic matrix multiplication [7, 22, 26] yield normalized Boolean circuits for the $n \times n$ Boolean matrix product of $O(n^\omega)$ size and $O(\log n)$ depth (see [5]). Similarly, the fast algorithm for integer multiplication [21] yields normalized Boolean circuits for the n-dimensional Boolean vector convolution of $O(n \log^2 n \log\log n)$ size and $O(\log n)$ depth [6, 5]. We can use these facts to derive the following upper-bound trade-offs analogous to our lower-bound trade-offs for these two problems.

▶ Proposition 1. There is a positive constant $c \leq 1$ such that for any $\epsilon \in (0, \frac{1}{c})$, the n-dimensional Boolean vector convolution can be computed by a normalized Boolean circuit of $\epsilon \log n$-bounded conjunction-depth and $O(n^{2-c\epsilon} n \log^2 n \log\log n)$ size.

Proof. By the aforementioned facts, for some positive constant $c \leq 1$, an $n^{c\epsilon}$-dimensional Boolean vector convolution can be computed by a normalized Boolean circuit of $\epsilon \log n$-bounded conjunction-depth and $O(n^{c\epsilon} \log^2 n \log\log n)$ size. On the other hand, since $c\epsilon < 1$, the n-dimensional Boolean vector convolution can be easily reduced to $n^{2-2c\epsilon}$ $n^{c\epsilon}$-dimensional Boolean vector convolutions using just disjunctions. The resulting normalized Boolean circuit has still $\epsilon \log n$-bounded conjunction-depth and $O(n^{2-c\epsilon} \log^2 n \log\log n)$ size. ◀

▶ Proposition 2. There is a positive constant $c \le 1$ such that for any $\epsilon \in (0, \frac{1}{c})$, the $n \times n$ Boolean matrix product can be computed by a normalized Boolean circuit of $\epsilon \log n$-bounded conjunction-depth and $O(n^{3-(3-\omega)c\epsilon})$ size.

Proof. By the aforementioned facts, there is a positive constant $c \le 1$ such that an $n^{c\epsilon} \times n^{c\epsilon}$ Boolean matrix product can be computed by a normalized Boolean circuit of $\epsilon \log n$-bounded conjunction-depth and $O(n^{\omega c\epsilon})$ size. On the other hand, since $c\epsilon < 1$, the $n \times n$ Boolean matrix product can be easily reduced to $n^{3-3c\epsilon}$ $n^{c\epsilon} \times n^{c\epsilon}$ Boolean matrix products using just disjunctions. The resulting normalized Boolean circuit has still $\epsilon \log n$-bounded conjunction-depth and $O(n^{3-(3-\omega)c\epsilon})$ size. ◀

6 Final Remarks

The disjointness of the sets of prime implicants of the Boolean functions forming a bilinear form is not essential in the proofs of Theorems 6, 7. Hence, these theorems hold even for Boolean bilinear forms satisfying only the two remaining conditions (see Introduction) provided that p denotes the number of distinct prime implicants of the form.

Our main results are the lower-bound trade-offs between the number of and-gates and conjunction-depth in normalized Boolean circuits computing semi-disjoint bilinear forms (Section 4). They rely on the analysis of output terms containing bounded numbers of literals because of the assumed bound on the conjunction-depth (Lemma 3, note that this lemma wouldn't hold if the fan-in of and-gates wasn't bounded). In case of the stronger lower bound trade-off for Boolean matrix product only the number of negated variables in output terms is bounded by the assumed bound on the negation-dependent conjunction-depth.

Likely, also for Boolean vector convolution and other non-necessarily bilinear forms, we could obtain stronger lower-bound trade-offs analogous to that for Boolean matrix product based on Lemma 10.

In order to apply Lemma 10 to a form F with known non-trivial lower bound on its monotone circuit complexity, we need to derive a generalization of this lower bound. Such a generalization should include a sparsification of the set of the prime implicants of F resulting in a form F'' having the properties stated in Lemma 10. (In case of Boolean matrix product this has been easy since the corresponding lower bound is tight, see Corollary 11.) E.g., suppose F is the (m,s)-clique given by the Boolean polynomial $\bigvee_{S\in[m]^s}\bigwedge_{\{i,j\}\subset S} x_{\{i,j\}}$, where $[m]^s$ stands for the family of s element subsets of $\{1,2,...,m\}$. Alon and Boppana [1], by refining Razborov's method [18], showed in [1] that any monotone Boolean circuit computing the (m,s)-clique has $\Omega((m/\log m)^s)$ and-gates for $s = O(1)$ (Theorem 3.16 in [1]). It seems possible to generalize their complicated proof in order to include the aforementioned sparsification. By combining this with Lemmata 2, 10, 3, we could obtain a similar lower-bound trade-off for the (m,s)-clique. Likely, it would be as follows: For fixed $s \ge 3$ and $\epsilon < \frac{2}{s-1}$, if a normalized Boolean circuit of $\epsilon \log m$-bounded negation-dependent conjunction-depth computes the (m,s)-clique then the circuit has $\Omega(m^{s-\epsilon\binom{s}{2}}/(\log m)^s)$ and-gates. Since the (m,s)-clique easily reduces to the $m^{\lceil s/3\rceil} \times m^{\lceil s/3\rceil}$ Boolean matrix product (see the subsection 3.5 in [1]), we can obtain a corresponding upper-bound trade-off for the (m,s) clique by substituting $m^{\lceil s/3\rceil}$ for n in the upper bound in Proposition 2.

Acknowledgments

The author is grateful to Mike Paterson and the anonymous conference reviewers for valuable comments/suggestions and to Mia Persson for valuable discussions on different earlier versions of this paper. The author is also thankful to Norbert Blum for his insightful comment on a manuscript preceding [12] and this paper. Finally, thanks go to Igor Sergeev for informing on [8].

References

1 N. Alon and R. B. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.

2 A. E. Andreev. On one method of obtaining constructive lower bounds for the monotone circuit size. *Algebra and Logics*, 26(1):3–26, 1987.

3 N. Blum. An $\omega(n^{4/3})$ lower bound on the monotone network complexity of the n-th degree convolution. *Theoretical Computer Science*, 36:59–69, 1985.

4 N. Blum. On negations in boolean networks. In *Efficient Algorithms*, volume 5760 of *Lecture Notes in Computer Science*, pages 18–29. Springer-Verlag, 2009.

5 J. H. Reif (editor). *Synthesis of Parallel Algorithms.* Morgan Kaufmann Publishers, San Mateo, 1993.

6 M. J. Fisher and M. S. Paterson. String-matching and other products. In *Proceedings of the 7th SIAM-AMS Complexity of Computation*, pages 113–125, 1974.

7 F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, Lecture Notes in Computer Science, pages 296–303. Springer-Verlag, 2014.

8 M. I. Grinchuk and I. S. Sergeev. Thin circulant matrices and lower bounds on the complexity of some boolean operations. *Diskretn. Anal. Issled. Oper.*, 18:35–53, 2011.

9 K. Iwama and H. Morizumi. An explicit lower bound of $5n - o(n)$ for boolean circuits. In *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 353–364. Springer-Verlag, 2002.

10 O. Lachish and R. Raz. Explicit lower bound of $4.5n - o(n)$ for boolen circuits. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 399–408. ACM, 2001.

11 E. A. Lamagna. The complexity of monotone networks for certain bilinear forms, routing problems, sorting, and merging. *IEEE Transactions on Computers*, c-28(10), 1979.

12 A. Lingas. Towards an almost quadratic lower bound on the monotone circuit complexity of the boolean convolution. In *Theory and Applications of Models of Computation*, Lecture Notes in Computer Science, pages 401–411. Springer-Verlag, 2017.

13 K. Mehlhorn and Z. Galil. Monotone switching circuits and boolean matrix product. *Computing*, 16:99–111, 1976.

14 M. Paterson. Complexity of monotone networks for boolean matrix product. *Theoretical Computer Science*, 1(1):13–20, 1975.

15 N. Pippenger and L.G. Valiant. Shifting graphs and their applications. *Journal of the ACM*, 23(3):423–432, 1976.

16 R. Pratt. The power of negative thinking in multiplying boolean matrices. *SIAM J. Comput.*, 4(3):326–330, 1975.

17 R. Raz. On the complexity of matrix product. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 144–151. ACM, 2002.

18 A. A. Razborov. Lower bounds on the monotone complexity of some boolean functions. *Doklady Akademii Nauk*, 281(4):798–801, 1985.

19 C. P. Schnorr. Zwei lineare untere schranken für die komplexität boolescher funktionen. *Computing*, 13(2):155–171, 1974.

20 C. P. Schnorr. A lower bound on the number of additions in monotone computations. *Theoretical Computer Science*, 2(3):305–315, 1976.

21 A. Schönhage and V. Strassen. Schnelle multiplikation grober zahlen. *Computing*, 7:281–292, 1971.

22 V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

23 L.G. Valiant. Negation can be exponentially powerfull. *Theoretical Computer Science*, 12:303–314, 1980.

24 I. Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner Series in Computer Science, New York, Stuggart, 1987.

25 J. Weiss. An $n^{3/2}$ lower bound on the monotone network complexity of the boolean convolution. *Information and Control*, 59:184–188, 1983.

26 V. Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, pages 807–898. ACM, 2012.

27 U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il