Finding Efficient Circuits Using SAT-solvers [⋆]

Arist Kojevnikov[1], Alexander S. Kulikov[2], and Grigory Yaroslavtsev[3]

[1] OneSpin Solutions GmbH
[2] St. Petersburg Department of Steklov Institute of Mathematics
[3] Academic Physics and Technology University of the RAS
{arist,kulikov,grigory}@logic.pdmi.ras.ru

Abstract. In this paper we report preliminary results of experiments with finding efficient circuits (over binary bases) using SAT-solvers. We present upper bounds for functions with constant number of inputs as well as general upper bounds that were found automatically. We focus mainly on MOD-functions. Besides theoretical interest, these functions are also interesting from a practical point of view as they are the core of the residue number system. In particular, we present a circuit of size $3n + c$ over the full binary basis computing MOD_3^n.

1 Introduction

In the recent years SAT solving became one of the main tools for formal verification [21]. In [9] it was proposed to use SAT in another very important area of the digital hardware production, namely in logical design synthesis. At the present time most electronic design automation tools (EDA) use algebraic manipulations [16] or binary decision diagrams (BDD) [14]. There are some successful experiments with genetic algorithms [7] and annealing optimizations [23]. See [10] for a survey.

Kamath at el. [9] propose a translation of the logical design synthesis problem to SAT. In [5] experiments with modern SAT solvers using this translation were reported. One of the advantages of this method is that it can also be used to prove lower bounds on Boolean functions, i.e., to prove that circuits of a given size do not exist. We use a similar reduction to CNF, however we are working with a more general computational model, namely with circuits over any binary basis.

It is known that finding efficient circuits is a difficult and important task. For many important functions there is a large gap between known lower and upper bounds. This shows that our current understanding of circuits is quite poor. As Williams notes [27] it might be helpful to know optimal circuits for such functions at least for small values of input size. Knowing this could help us to understand the structure of optimal circuits for general functions.

[⋆] The first two authors are supported in part by RFBR (grant 08-01-00640-a) and RAS Program for Fundamental Research ("Modern Problems of Theoretical Mathematics").

In this paper we report results of experiments with finding efficient circuits using SAT-solvers. We focus mainly on circuit complexity of MOD-functions defined as follows:

$$\mathrm{MOD}^n_{K,k}(x_1,\ldots,x_n) = 1 \text{ iff } \sum_{i=1}^{n} x_i \equiv k \pmod{K}$$

(we omit k and/or n when they are not important). These are one of the simplest symmetric Boolean functions. Circuit complexity of these functions was studied by many researchers. Still, we know the exact circuit complexity for only a few values of K. Table 1 shows known lower and upper bounds for MOD^n_K in different computational models. There, by C and L we denote the circuit and formula complexity, respectively; B_2 is the full binary basis, $U_2 = B_2 \setminus \{\oplus, \equiv\}$. Interestingly, for formulas and circuits over the bases U_2 and B_2 it is known that the complexity of MOD^n_K, $K = 3$ or $K \geq 5$, is not less than the complexity of MOD^n_4. However, for none of these models it is known that MOD^n_3 or MOD^n_5 is strictly harder than MOD^n_4.

K		L_{U_2}	L_{B_2}	C_{U_2}	C_{B_2}
2	lower	$\Theta(n^2)$ [11]	n	$3n + c$ [24]	$n - 1$
	upper				
3	lower	$\Omega(n^2)$ [11]	$\Omega(n \log n)$ [6]	$4n + c$ [28]	$2.5n + c$ [25]
	upper	$O(n^{2.58})$ [2]	$O(n^2)$ [26]	$7n + o(n)$ [17]	$5n + o(n)$ [17]
4	lower	$\Theta(n^2)$ [11]	$\Omega(n \log n)$ [6]	$4n + c$ [28]	$2.5n + c$ [25]
	upper	$O(n^2 \log^2 n)$[6]		$5n$ [28]	
≥ 5	lower	$\Omega(n^2)$ [11]	$\Omega(n \log n)$ [6]	$4n + c$ [28]	$2.5n + c$ [25]
	upper	$O(n^{4.57})$ [19]	$O(n^{3.13})$ [19]	$7n + o(n)$ [17]	$5n + o(n)$ [17]

Table 1. Known lower and upper bounds on the complexity of MOD^n_K in different computational models.

Another motivation for studying MOD-functions is that a residue number system [12] is based on such functions. One of the main advantages of the residue number system is that additions, subtractions and multiplications are carry-free.

MOD-functions can be computed inductively. For example, the optimal circuit of size $2.5n + c$ for MOD^n_4 by Stockmeyer [25] is constructed from blocks consisting of 10 gates that sums 4 new variables with a residue number modulo 4, see Fig. 1. There, the bits z_0, z_1 encode the value of $\sum_{i=1}^{n} x_i \pmod{4}$ as follows:

$$\sum_{i=1}^{n} x_i \pmod 4 = \begin{cases} 0, & \text{if } (z_0, z_1) = (0,0), \\ 1, & \text{if } (z_0, z_1) = (1,1), \\ 2, & \text{if } (z_0, z_1) = (1,0), \\ 3, & \text{if } (z_0, z_1) = (0,1). \end{cases}$$

The two output bits z_0', z_1' encode the value of $\sum_{i=1}^{n+4} x_i \pmod 4$ in the same way. Thus, one can prove general upper bounds on the circuit complexity of MOD-functions by finding efficient blocks of constant size. We report the results of experiments with finding such blocks by translating them to SAT.

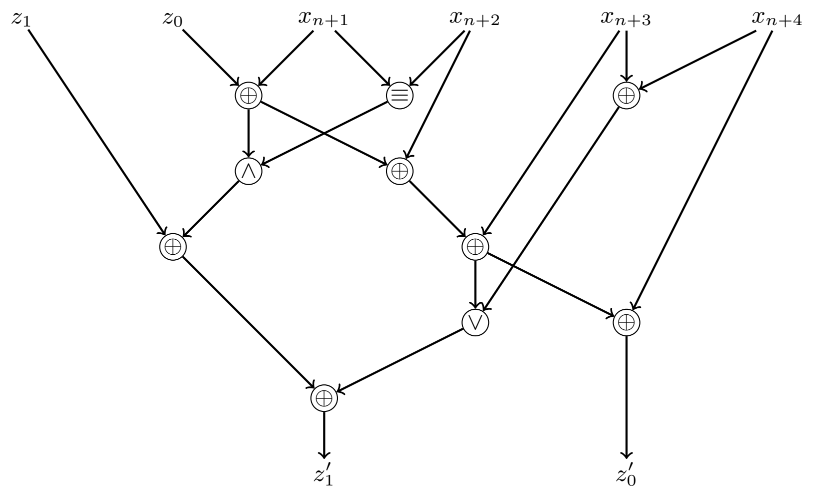


Fig. 1. Stockmeyer's block for MOD_4

The rest of the paper is organized as follows. In Sect. 2 we give all the necessary definitions. Section 3 describes the way we translate the fact of existence of a particular circuit into a CNF formula. In Sect. 4 we present some new circuit complexity upper bounds that were proved automatically. Sect. 5 presents results of experiments. Finally, in Sect. 6 we discuss some further directions.

2 General Setting

By B_n we denote the set of all Boolean functions $f \colon \{0,1\}^n \to \{0,1\}$. A function $f \in B_n$ is called symmetric if its value depends on the sum of the input bits only. That is, there must exist a vector $v \in \{0,1\}^{n+1}$ such that $f(x_1, \ldots, x_n) = v_s$ where $s = \sum_{i=1}^{n} x_i$. A typical symmetric function is a modular function $\text{MOD}_{K,k}^n$ defined as follows:

$$\text{MOD}_{K,k}^n(x_1, \ldots, x_n) = 1 \text{ iff } \sum_{i=1}^{n} x_i \equiv k \pmod K .$$

A circuit over the basis $A \subseteq B_2$ is a directed acyclic graph with nodes of in-degree 0 or 2. Nodes of in-degree 0 are marked by variables from $\{x_1, \ldots, x_n\}$ and are called inputs. Nodes of in-degree 2 are marked by functions from A and are called gates. There are also special output gates. The size of a circuit is its number of gates. In this paper we mainly consider circuits over the full binary basis B_2.

We call a function $f \in B^n$ degenerated if it does not depend essentially on some of its variables, i.e., there is a variable x_i such that the subfunctions $f|_{x_i=0}$ and $f|_{x_i=1}$ are equal. It is easy to see that a gate computing a degenerated function from B_2 can be easily eliminated from a circuit without increasing its size (when eliminating this gate one may need to change the functions computed at its successors). For example, a gate computing NOT is degenerated. The set B_2 contains exactly ten non-degenerated functions $f(x, y)$:

– eight functions of the form $((x \oplus a) \wedge (y \oplus b)) \oplus c$, where $a, b, c \in \{0, 1\}$;
– two functions of the form $x \oplus y \oplus a$, where $a \in \{0, 1\}$;

3 Using SAT-Solvers for Finding Small Circuits

In this section we give the details of the reduction we use to encode the fact of existence of a particular circuit in CNF. We first describe the general reduction which is quite similar to the one described in [3] (where circuits over the basis U_2 are considered) and then discuss also some additional encoding which is specific to the considered functions.

3.1 Representing Circuits as CNFs

Given a truth table of a Boolean function $f\colon B^n \to B^m$ we would like to find a Boolean circuit computing f over the given basis $A \subseteq B_2$ with the smallest possible number of gates. We can encode the fact of existence of a circuit with N gates computing the function f in CNF using the following Boolean variables (input variables are numbered from 0 to $n-1$ and gates from n to $n+N-1$):

1. $t_{ib_1b_2}$ ($n \le i < n+N$, $0 \le b_1 < 2$, $0 \le b_2 < 2$) is the output of the i-th gate if its first input is b_1 and the second is b_2. Four variables t_{i00}, t_{i01}, t_{i10}, t_{i11} thus completely define the binary Boolean function computed by the i-th gate. It gives $O(N)$ variables in total.
2. c_{ikj} ($n \le i < n+N-1$, $0 \le k < 2$, $0 \le j < n+N$) is true if the k-th input of the i-th gate comes from the j-th gate and false otherwise. These variables completely define the underlying graph of a circuit. It gives $O(N^2)$ variables in total.
3. o_{ij} ($n \le i < n+N$, $0 \le j < m$) is true iff the j-th output of a circuit is computed by the i-th gate. These variables define which gates are used as outputs. It gives $O(Nm)$ variables in total.

4. v_{it} ($0 \le i \le n+N$, $0 \le t < 2^n$) is the output value of the i-th gate if the input variables have values represented by the bits of t. These variables are used to describe the fact that the values computed by a circuit are correct (according to the given truth table) on all 2^n assignments to input variables. It gives $O(2^n N)$ variables in total.

The following requirements about the circuit are written down as clauses.

1. Binary functions computed by gates belong to the basis A.
2. For all (i,k), exactly one variable c_{ikj} is true (the k-th input of the i-th gate is connected to only one gate). It gives $O(N^3)$ 2-clauses and $O(N)$ $O(N)$-clauses.
3. For all j, only one variable o_{ij} is true (the j-th output is computed by exactly one of the gates). It gives $O(N^2 m)$ 2-clauses and $O(m)$ $O(N)$-clauses.
4. For all $0 \le i < n$ and $0 \le t < 2^n$, v_{it} is equal to the corresponding bit in t. It gives $O(n \cdot 2^n)$ 1-clauses.
5. For all $n \le i < n+N$ and $0 \le t < 2^n$, v_{it} is equal to the value computed by the i-th gate in the part of a circuit described by other variables. It gives $O(N^3 \cdot 2^n)$ 6-clauses and this is where the most significant part of all clauses comes from. Clauses of this type are written for all $n \le i < n+N$, $n \le j_0 < i$, $j_0 < j_1 < i$, $0 \le i_0 < 2$, $0 \le i_1 < 2$, $0 \le r < 2^n$ and look as follows:
$$\neg c_{i0j_0} \vee \neg c_{i1j_1} \vee \neg(v_{j_0 r} = i_0) \vee \neg(v_{j_1 r} = i_1) \vee (v_{ir} = t_{ii_0i_1})\,.$$
Here first two literals let us find two gates, that are inputs of the i-th gate, the following two literals let us find the values of these gates on input assignment r and the last literal is for checking that the value in the i-th gate is correct.
6. The outputs of a circuit are computed correctly. It gives $O(N2^n m)$ 2-clauses. Clauses of this type are written for all $0 \le k < m$, $0 \le r < 2^n$, $n \le i < n+N$ and look as follows:
$$\neg o_{ik} \vee (v_{ir} = value_{kr})\,,$$
where $value_{kr}$ is the required value of the k-th output on input assignment r, according to the given truth table.

The clauses described above completely define all the requirements on a circuit. W.l.o.g. we can assume also that the following statements are true.

1. Both inputs of every gate are computed by gates with smaller numbers (i.e., the gates are sorted topologically w.r.t. the used numbering).
2. For every gate its first input gate has a smaller number than the second one.
3. The gates do not compute degenerated functions.
4. At least one of the outputs is computed by the last gate.

In most interesting cases the resulting formulas turn out to be quite difficult for modern SAT-solvers. E.g., a formula encoding the fact of existence of a circuit consisting of, say, 12 gates is already quite hard. This is because the number of different circuits as a function of the number of gates grows extremely fast. That is why in some cases we used also some additional restrictions that reduce the set of considered circuits. The main two of them are given below. Note however that unsatisfiability of a CNF formula with at least one of these restrictions does not imply that a circuit of a given size does not exist.

1. The out-degree of every gate is at most 2.
2. The i-th gate is fed by the $(i-1)$-th gate (i.e., there is a directed path through all the gates).

3.2 Residue Number Encodings

In the previous subsection we consider functions given by a truth table. Note however that one can work as well with partially defined functions and, more generally, with functions satisfying some particular properties. E.g., when searching for an inductive block for a MOD-function it is not clear what is the optimal encoding of a residue number. Thus, instead of providing a truth table of a block one can write down the fact that this block sums up new variables with a residue number which is encoded somehow.

Assume that we are looking for a block for MOD_K. Such a block sums up several variables with a residue number modulo K whose encoding is not known in advance. This residue number is encoded by $\lceil \log_2 K \rceil$ bits. Since the encoding is not known, we introduce new variables e_{ij}, where e_{ij} is true iff bit representation of $0 \le j < 2^{\lceil \log_2 K \rceil}$ encodes the residue number $0 \le i < K$. Thus, a particular residue number can be encoded by several j's. Except for some straightforward clauses stating that each i is encoded by some j and that each j is used for exactly one i we add also the following statement. For each possible assignment for inputs of a block, if the sum of input variables is s and the input residue number is i (i.e., the corresponding e_{ij} is true), then the output residue number cannot be j' for all j' such that $e_{i'j'}$ is true for some $i' \not\equiv i+s \pmod{K}$.

To give an example assume that we are searching for a block that takes as input a residue number t modulo 3 which is somehow encoded by two bits (z_0, z_1) and a new variable x_n and outputs two bits (z'_0, z'_1) that encode in the same way $t + x_n \pmod{3}$. Then, e_{23} is true iff $(z_0, z_1) = (1, 1)$ implies $t = 2$ and e_{11} is true if $(z_0, z_1) = (0, 1)$ implies $t = 1$. Now we add the following constraint:

$$(e_{23} \wedge z_0 \wedge z_1 \wedge x_n \wedge e_{11}) \Rightarrow (z'_0 \vee \neg z'_1)\,.$$

These residue number encodings turned out to be quite helpful as only with them we were able to find an efficient block implying a $5.5n + c$ upper bound for $C_{U_2}(\mathrm{MOD}_3^n)$. However in most cases finding a block with an unknown encoding is a much more difficult task.

4 New Upper Bounds for MOD_3

Our main theoretical results are circuits of size $3n + O(1)$ and $5.5n + O(1)$ for MOD_3^n over the bases B_2 and U_2, respectively. The building blocks of the circuits (as well as their truth tables) are given in Fig. 2 and Fig. 3. The blocks take as input the value of $\sum_{i=1}^{n} x_i \pmod 3$ encoded by a pair of bits (z_1, z_2) and three respectively two new variables. The output is the pair of bits (z_1', z_2') encoding the value of $\sum_{i=1}^{n+3} x_i \pmod 3$ respectively $\sum_{i=1}^{n+2} x_i \pmod 3$. The residue number encodings for the blocks are the following:

$$\sum_{i=1}^{n} x_i \pmod 3 = \begin{cases} 0, & \text{if } (z_0, z_1) = (0,0), \\ 1, & \text{if } (z_0, z_1) = (0,1), \\ 2, & \text{if } z_0 = 1, \end{cases}$$

and

$$\sum_{i=1}^{k} x_i \pmod 3 = \begin{cases} 0, & \text{if } z_0 = 0, \\ 1, & \text{if } (z_0, z_1) = (1,0), \\ 2, & \text{if } (z_0, z_1) = (1,1). \end{cases}$$

The upper bounds $C_{B_2}(\mathrm{MOD}_3^n) \le 3n + O(1)$ and $C_{U_2}(\mathrm{MOD}_3^n) \le 5.5n + O(1)$ follow immediately from the existence of such blocks.

The blocks were found after a long sequence of experiments with different restrictions on considered circuits as without restrictions the resulting formulas cannot be handled by solvers. We still do not know whether the first block for adding three variables is optimal and thus a $8n/3$ upper bound for $C_{B_2}(\mathrm{MOD}_3^n)$ is not excluded.

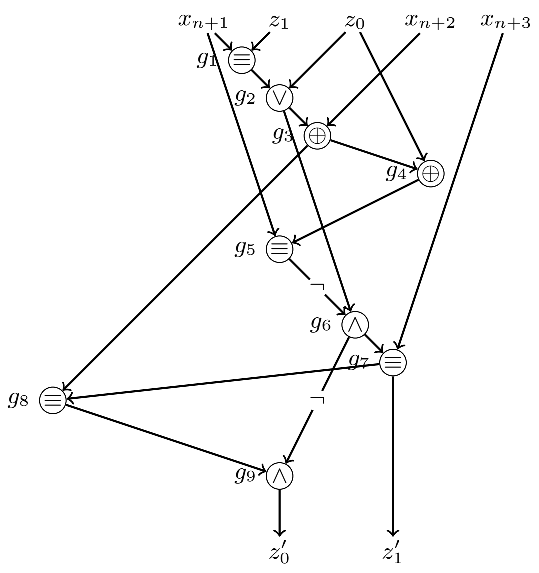
In Table 2 we also present the sizes of optimal circuits over B_2 for $\mathrm{MOD}_{3,k}^n$ for different values of n and k. $C_{B_2}(\mathrm{MOD}_{3,2}^5) \le 10$ means that we found a circuit of size 10, but were unable to prove unsatisfiability of the fact that there exists a circuit (without any restrictions on its structure) of size 9. Optimal circuits are given in Fig. 4.

	$n=3$	$n=4$	$n=5$
$k=0$	3	7	≤ 10
$k=1$	4	7	≤ 9
$k=2$	4	6	≤ 10

Table 2. Sizes of optimal circuits over B_2 for $\mathrm{MOD}_{3,k}^n$

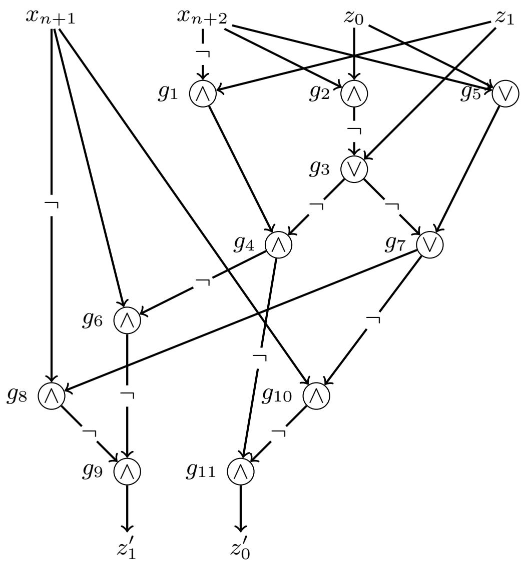
5 Empirical Studies

Table 3 provides the results of experiments with several formulas. All our experiments were made on a 2.40GHz AMD Opteron Processor 250 running under Linux. The DIMACS hardware benchmark program *dfmax r500.5* takes 7.11 seconds on the machine. We take compiled versions of best solvers from



x_{n+1}	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_{n+2}	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_{n+3}	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
z_0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
z_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
g_1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
g_2	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	1
g_3	1	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1	0	1	1	0	0	1	1	0	0
g_4	1	0	0	1	1	0	0	1	0	0	1	1	0	0	1	1	0	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1
g_5	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	1	0	0	1	1	0	0	1
g_6	1	0	0	0	1	0	0	0	0	1	1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	1	1	0	0	1	1	0
g_7	0	1	1	1	1	0	0	0	1	0	0	1	0	1	1	0	1	1	1	0	0	0	0	1	1	0	0	1	0	1	1	0
g_8	0	0	0	1	1	1	1	0	1	0	1	0	0	1	0	1	0	1	1	1	1	0	0	0	1	0	1	0	0	1	0	1
g_9	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	1
z'_0	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	1
z'_1	0	1	1	1	1	0	0	0	1	0	0	1	0	1	1	0	1	1	1	0	0	0	0	1	1	0	0	1	0	1	1	0

Fig. 2. An inductive block for MOD$_3$ over the basis B_2 and its truth table.



x_{n+1}	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_{n+2}	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
z_0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
z_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
g_1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0
g_2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1
g_3	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1
g_4	1	1	0	0	1	1	0	0	0	0	0	0	0	0	1	1
g_5	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1	1
g_6	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0
g_7	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1
g_8	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
g_9	0	1	1	1	0	1	1	1	1	1	1	0	1	0	0	1
g_{10}	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1	0
g_{11}	0	0	0	1	0	0	0	1	0	1	1	1	1	1	0	0
z'_0	0	1	1	1	0	1	1	1	1	1	1	0	1	0	0	1
z'_1	0	0	0	1	0	0	0	1	0	1	1	1	1	1	0	0

Fig. 3. An inductive block for MOD_3 over the basis U_2 and its truth table.

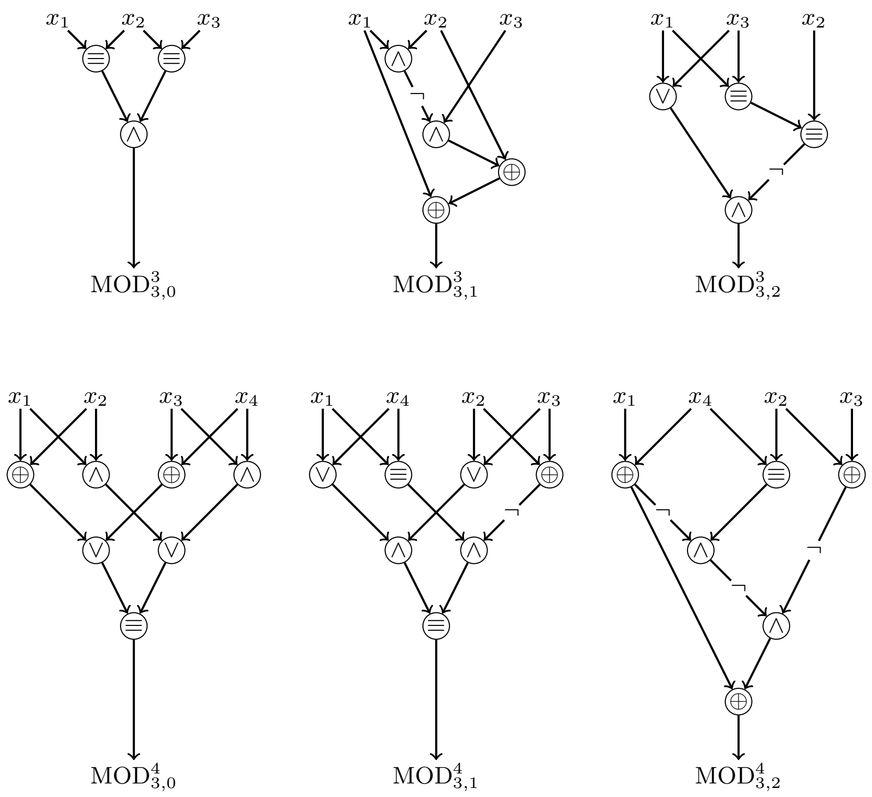


Fig. 4. Optimal circuits for $\mathrm{MOD}_{3,k}^{n}$ for $n = 3, 4$

	picosat [1]	minisat207 [4]	oksolver [13]	rsat [20]	onespinsat [18]	manysat [8]
`mod3_4vars_6gates.cnf`	*	11m32s	*	*	3m16s	35m23s
`mod4block_3vars_7gates.cnf`	*	*	*	*	*	*
`mod3block_3vars_9gates_restr.cnf`	1m27s	0m5s	0m59s	0m2s	0m18s	0m29s
`mod5block_1var_7gates.cnf`	0m1s	0m2s	*	0m1s	0m1s	0m3s
`mod4block_2vars_8gates_u2.cnf`	*	*	*	*	22m43s	*
`mod5block_2vars_12gates.cnf`	*	*	*	*	*	*
`mod4block_2vars_9gates_u2.cnf`	*	*	*	*	*	*
`owp_4vars_5gates.cnf`	0m0.2s	0m0.3s	2m14s	0m0.3s	0m0.1s	0m0.7s

Table 3. Statistics in real time

http://www.satcompetition.org/ and ManySAT and onespinsat from their authors.

A two-hour real time limit applied to all runs of solvers (for the best of 4 solvers of the parallel solver ManySAT). It is interesting to see from experiments that industrial solvers behave better on our benchmarks and more powerful solvers demonstrate better performance. Below we describe the benchmarks used in the table as well as benchmarks for which we still do not know the answer. By N, K and L we denote the number of variables, the number of clauses and the length (i.e., the total number of literals), respectively (however our formulas contain some unit clauses and can be simplified).

- SAT
  - `mod3block_3vars_9gates_restr.cnf` ($N = 784$, $K = 219760$, $L = 1266513$) expresses the fact that there exists a circuit that sums up a residue number modulo 3 in a given encoding (0 — $(0,0)$, 1 — $(1,0)$, 2 — $(1,0),(1,1)$) with three new variables with two additional restrictions: there is a directed path through all the gates and the out-degree of any gate is at most 2. The corresponding circuit is given in Fig. 2.
  - `mod5block_1vars_7gates.cnf` ($N = 353$, $K = 50103$, $L = 292994$) encodes the fact that there exists an inductive block for MOD_5^n adding two new variables by 7 gates (residue number encoding is fixed).
- UNSAT
  - `mod3_4vars_6gates.cnf` ($N = 289$, $K = 36396$, $L = 213400$) expresses the fact that $C_{B_2}(\mathrm{MOD}^4_{3,0}) \leq 6$.
  - `owp_vars4_gates5.cnf` ($N = 267$, $K = 25667$, $L = 149802$) encodes the fact that $C_{B_2}(f) \leq 5$ for a permutation $f\colon B_4 \to B_4$ given in [15]. Un-

satisfiability of this benchmark justifies the fact that f has asymmetric circuit complexity ($C_{B_2}(f) = 6$, $C_{B_2}(f^{-1}) = 5$).

– UNKNOWN

`mod4block_3vars_7gates.cnf` ($N = 574$, $K = 125562$, $L = 742458$) encodes the fact that there exists an inductive block for MOD_4^n adding three new variables by 7 gates (residue number encoding is fixed). Must be unsatisfiable, as otherwise MOD_4^n could be computed by circuits of size about $7n/3$.

`mod4block_2vars_{8,9}gates_u2.cnf` ($N = 387/426$, $K = 66496/86121$, $L = 389012/503636$) encodes the fact that there exists an inductive block for MOD_4^n adding two new variables by $\{8, 9\}$ gates in the basis U_2 (residue number encoding is fixed).

`mod3block_2vars_{9,10,11}gates_u2_autoenc.cnf` ($N = 426/475/526$, $K = 99345/125336/155313$, $L = 588054/741756/918948$) encodes the fact that there exists an inductive block for MOD_3^n adding two new variables by 9 gates in the basis U_2. Here the residue number encoding is not fixed. Instead of this, benchmarks encode the fact that the required circuit sums up several bits with a residue number modulo 3 whose encoding is not known in advance (details are given in Sect. 3).

`mod4block_2vars_{8,9,10,11}gates_u2_autoenc.cnf` ($383 \leq N \leq 530$, $81176 \leq K \leq 164070$, $480856 \leq L \leq 965444$) encodes the fact that there exists an inductive block for MOD_4^n adding two new variables by $g = 8, 9, 10, 11$ gates in the basis U_2 (encoding is not fixed). Satisfiability of a benchmark from this family would imply that $C_{U_2}(\mathrm{MOD}_4^n) \leq gn/2 + c$ (note that at the moment it is only known that $4n - c \leq C_{U_2}(\mathrm{MOD}_4^n) \leq 5n + c$).

6 Further Directions

A natural further direction is to obtain more upper bounds for MOD_K-functions for different values of K. Note however that even if an optimal circuit for a MOD_K^n function can be constructed from inductive blocks, then these blocks must be large for large values of K, just because one needs many bits in order to encode a residue number modulo K. E.g., a block for summing up several new variables with a residue number modulo $K > 2^t$ must have at least t inputs and hence at least t gates. For $t \geq 15$, even finding such circuits is a really difficult task for modern SAT-solvers and proving that such a circuit does not exist is much more difficult.

It would be interesting also to find efficient circuits for other important functions. E.g., in [27] is is noted that it is easy to construct optimal circuits for 2×2-matrix multiplication, while already for 3×3 this is a difficult task. It would also be interesting to know optimal circuits for small input sizes for the well-known CLIQUE-function which has super-polynomial complexity in the model of monotone circuits [22]. Knowing optimal circuits for a function on small input sizes could help to construct efficient circuits for all input sizes. SAT-solvers could also apparently help to improve current upper bounds for addition and multiplication. Note that the smallest known circuits and formulas for these functions are also built from blocks [19].

Using the described reduction one can produce different unsatisfiable formulas (e.g., encoding the fact that there exists a circuit of size smaller than the corresponding known lower bound). Such benchmarks turned out to be difficult for modern solvers. One could think about complexity of such benchmarks in different proof systems.

Another direction of further research is automatizing lower bounds proofs. Essentially the only known method for proving lower bounds for unrestricted circuits is gate elimination. For example, in order to prove a $2.5n-c$ lower bound for MOD_4^n Stockmeyer [25] proved that for any circuit computing $\mathrm{MOD}_{4,k}^n$ it is possible to eliminate five gates by assigning values to two input variables. The lower bound then follows by induction. However to prove that it is possible to eliminate five gates one needs to consider many different cases. It would be interesting to automate this case analysis.

Acknowledgments

The authors are very grateful to Anton Belov, Edward A. Hirsch, Evgeny Karibaev, Yakov Novikov and anonymous reviewers for helpful comments.

References

1. Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 4:75–97, 2008.
2. Andrew Chin. On the depth complexity of the counting functions. *Information Processing Letters*, 35:325–328, 1990.
3. Niklas Eén. Practical SAT — a tutorial on applied satisfiability solving. Slides of invited talk at FMCAD, 2007.
4. Niklas Een and Niklas Sörensson. An extensible sat-solver. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, 2003.
5. Giovani Gomez Estrada. A note on designing logical circuits using SAT. In *Proceedings of the 5th International Conference on on Evolvable Systems (ICES'03)*, volume 2606 of *Lecture Notes in Computer Science*, pages 410–421, 2003.
6. Michael J. Fischer, Albert R. Meyer, and Michael S. Paterson. $\Omega(n \log n)$ lower bounds on length of Boolean formulas. *SIAM Journal on Computing*, 11:416–427, 1982.
7. David B. Fogel. Evolutionary computation: The fossil record. New York, 1998. IEEE Press.
8. Youssef Hamadia, Said Jabbour, and Lakhdar Sais. Manysat: solver description. Technical Report MSR-TR-2008-83, Microsoft Research, 2008.
9. A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. An interior point approach to boolean vector function synthesis. In *Proceedings of the 36th International Midwest Symposium on Circuits and Systems (MSCAS'93)*, pages 185–189, 1993.

10. Sunil Khatri and Narendra Shenoy. Logic synthesis. In Louis Scheffer, Luciano Lavagno, and Grant Martin, editors, *Electronic Design Automation For Integrated Circuits Handbook*. CRC Press, Taylor & Francis Group, 2006.
11. V. M. Khrapchenko. Complexity of the realization of a linear function in the case of Π-circuits. *Math. Notes Acad. Sciences*, 9:21–23, 1971.
12. Israel Koren. *Computer Arithmetic Algorithms.* Prentice Hall, Englewood Cliffs, New Jersey 07632, 1993.
13. Oliver Kullmann. Specification of oksolver. version 1.2, standard options. Slides for the short presentation of OKsolver at the SAT 2002 symposium, 2002.
14. C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal*, 38:985–999, 1959.
15. James L. Massey. The difficulty with difficulty. A Guide to the Transparencies from the EUROCRYPT'96 IACR Distinguished Lecture, 1996.
16. E.J. McCluskey. *Logic Design Principles: with emphasis on testable semicustom circuits.* Prentice-Hall, Englewood Cliffs, NJ, 1986.
17. Roshal G. Nigmatullin. *Slognost' bulevikh funktsii.* Moskva, Nauka, 1991. In Russian.
18. Yakov Novikov. Private communication. OneSpin Solutions GmbH, 2009.
19. Michael S. Paterson and Uri Zwick. Shallow circuits and concise formulae for multiple addition and multiplication. *Computational Complexity*, 3:262–291, 1993.
20. Knot Pipatsrisawat and Adnan Darwiche. Rsat 2.0: Sat solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007.
21. Mukul R. Prasad, Armin Biere, and Gupta Aarti. A survey of recent advances in SAT-based formal verification. *International Journal on Software Tools for Technology Transfer*, 7(2):156–173, 2005.
22. Alexander A. Razborov. Lower bounds for the monotone complexity of some Boolean functions. *Soviet Math. Doklady*, 31:354–357, 1985.
23. C. D. Gelatt S. Kirkpatrick and M. P. Vecchi. Optimization by simulated annealing. *Science, New Series*, 220(4598):671–680, May 13 1983.
24. C. Schnorr. Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen. *Computing*, 13:155–171, 1974.
25. Larry J. Stockmeyer. On the combinational complexity of certain symmetric Boolean functions. *Mathematical Systems Theory*, 10:323–336, 1977.
26. D. C. van Leijenhorst. A note on the formula size of the "mod k" functions. *Information Processing Letters*, 24:223–224, 1987.
27. Ryan Williams. Applying practice to theory. *ACM SIGACT News*, 39(4):37–52, December 2008.
28. Uri Zwick. A $4n$ lower bound on the combinational complexity of certain symmetric boolean functions over the basis of unate dyadic Boolean functions. *SIAM Journal on Computing*, 20:499–505, 1991.