Improving Circuit Size Upper Bounds Using SAT-Solvers

Alexander S. Kulikov
St. Petersburg Department of Steklov Institute of Mathematics
https://logic.pdmi.ras.ru/~kulikov/

***Abstract*—Boolean circuits is arguably the most natural model for computing Boolean functions. Despite intensive research, for many functions, we still do not know what optimal circuits look like. In this paper, we discuss how SAT-solvers can be used for constructing optimal circuits for functions on moderate number of variables. We first discuss why this problem is important and then indicate the current frontiers: what can and cannot be found by state-of-the-art SAT-solvers, and for what functions we are interested in finding efficient circuits.**

I. General Setting

Denote by $B_{n,m}$ the set of all Boolean functions

$$f\colon \{0,1\}^n \to \{0,1\}^m$$

and let $B_n = B_{n,1}$. A *straight line program* is a natural way of computing (and describing) such a Boolean function $f(x_1,\dots,x_n)$. It consists of r lines where the i-th line has the form

$$x_{n+i} = b_i(x_j, x_k)$$

such that $b_i \in B_2$ is a binary Boolean operation and $1 \le j,k \le n+i-1$. In other words, each line is just an application of some binary function to input variables $(x_1,\dots,x_n)$ or results of the previous lines $(x_{n+1},\dots,x_{n-i+1})$. We say that such a program computes f, if each of the m outputs of f is computed by some of the lines of the program. To give a specific example, consider a fundamental function

$$\mathrm{SUM}_n \in B_{n,\lceil \log_2(n+1)\rceil}$$

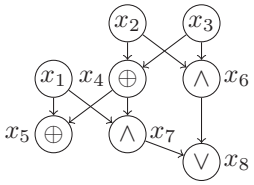
that outputs the binary representation of the sum (over integers) of the n input bits. This is a *symmetric* function since its output depends on the sum of input bits only. Then, $\mathrm{SUM}_3(x_1,x_2,x_3)$ can be computed by the following straight line program with five lines:

$$x_4 = x_2 \oplus x_3$$
$$x_5 = x_1 \oplus x_4$$
$$x_6 = x_2 \wedge x_3$$
$$x_7 = x_1 \wedge x_4$$
$$x_8 = x_6 \vee x_7$$

It is not difficult to check, that x_5 computes the sum of input bits modulo 2 (i.e., $x_1 \oplus x_2 \oplus x_3$) while x_8 computes the carry bit (which is equal to 1 iff $x_1+x_2+x_3 \ge 2$). This ensures that for all $(x_1,x_2,x_3) \in \{0,1\}^3$,

$$x_1 + x_2 + x_3 = x_5 + 2x_8\,.$$

A *Boolean circuit* is a convenient way of representing a straight line program. This is best illustrated using the sample program above. The circuit below is known as (one-bit) full adder.



Hence, a Boolean circuit is a directed acyclic graph where each node has in-degree either 0 or 2. Nodes of in-degree 0 are called *input gates* and are labeled with $x_1,\dots,x_n$. Nodes of in-degree 2 are called *internal gates* or just *gates* and are labeled with binary Boolean operations and $x_{n+1},\dots,x_{n+r}$. For a circuit C, by $\mathrm{gates}(C)$ we denote the *size* of C, i.e., the number of (internal) gates. For a function $f \in B_{n,m}$, by $\mathrm{gates}(f)$ we denote the minimum size of a circuit computing f.

Proving *lower bounds* on circuit size is a fundamental problem in theoretical computer science that turned out to be extremely difficult. By comparing the number of small size circuits with the total number 2^{2^n} of Boolean predicates on n variables, Shannon [1] showed that the circuit size of almost all Boolean functions from B_n is $\Omega(\frac{2^n}{n})$. However this is a *non-constructive* argument: it shows that functions of high circuit complexity exist without providing any specific candidate. Usually, we want a candidate function to belong to some reasonable complexity class, e.g., P or NP. For such functions, only modest linear lower bounds are known, the strongest one being $(3+\frac{1}{86})n - o(n)$ [2]. Much stronger lower bounds are known for restricted circuit classes like monotone circuits, constant depth circuits, and formulas.

In this paper, we deal with circuit size *upper bounds*: we show how to use SAT-solvers to check whether a given function $f \in B_n$ has circuits of size r when n and r are small constants (say, $n \le 7$, $r \le 12$). At this point, it is natural to ask: why should we care about the case when n and r are so small? Indeed, in practice, we are interested in much larger values of n and r (also, in practice, we should take into

Research was supported by Russian Science Foundation (project 16-11-10123).

account other parameters of a circuit like its depth). In theory, we are interested in upper bounds w.r.t. the input size n for *families of functions* (when we have a separate function for each input length: $\{f_n\}_{n=0}^{\infty}$ where $f_n \in B_n$) rather than for a single function $f \in B_n$ with n being a small constant. Still, there are several reasons to be interested in the $n = O(1)$ case.

1) For some families of functions $\{f_n\}_{n=0}^{\infty}$, proving an upper bound on $\mathrm{gates}(f_n)$ for $n = O(1)$ *automatically* translates into an upper bound on $\mathrm{gates}(f_n)$ for all n. Below, we show that $\mathrm{gates}(\mathrm{SUM}_3) = 5$ implies that $\mathrm{gates}(\mathrm{SUM}_n) \leq 5n + o(n)$ and that, in turn, implies a $5n+o(n)$ upper bound for *all* symmetric functions. We show also how to improve the $5n$ upper bound. A similar effect occurs for some other symmetric functions.
2) For some families of functions $\{f_n\}_{n=0}^{\infty}$, knowing good upper bounds on $\mathrm{gates}(f_n)$ for small values of n *may help* us to improve known upper bounds for all n. Let us quote Williams [3] at this point: "Our knowledge of Boolean circuit complexity is quite poor. [...] One good reason why we don't know much about the true power of circuits is that we don't have many examples of minimum circuits. We don't know, for example, what an optimal circuit for 3×3 Boolean matrix multiplication looks like. It is possible that we could make progress in understanding circuits by cataloging the smallest circuits we know for basic functions, on small input sizes (such as $n = 1, \ldots, 10$). [...] Might we benefit from an Encyclopedia of Minimum Circuits? For example, what do the smallest Boolean circuits for 10×10 Boolean matrix multiplication look like? Are they regular in structure? It is likely that the answers would give valuable insight into the complexity of the problem. [...] Even if the cataloged circuits are not truly minimal but close to that, concrete examples for small inputs could be useful for theoreticians to mine for inspiration, or perhaps for computers to mine for patterns via machine learning techniques. The power of small examples cannot be underestimated."
   In [4, Section 7.1.2] Knuth lists optimum size circuits for all symmetric functions on four variables: "Some of them are astonishingly beautiful; some of them are beautifully simple; and others are simply astonishing. (Look, for example, at the 8-step computation of $S_{2,3}(x_1, x_2, x_3, x_4, x_5)$, or the elegant formula for $S_{2,3,4}$, or the nonmonotonic chains for $S_{4,5}$ and $S_{3,4,5}$.)"

II. Reduction to SAT

The way of writing down a statement "for a given truth table of a function $f \in B_{n,m}$ and a given integer r, there exists a circuit of size r computing f" as a CNF formula is fairly straightforward and can be found in [5] or in [6, Solution to exercise 477]. The corresponding `Python` script can be found at [7]. The same repository contains several benchmarks whose satisfiability status is still unknown. Finding a satisfying assignment for any of them would automatically improve known circuit upper bound for a certain symmetric function. Knuth's implementation can be found at [8] (`sat-chains.w` file in `SATexamples` archive).

Computational experiments reveal that it is already difficult for state-of-the-art SAT solvers to check whether there exists a circuit of size $r = 12$ for the given Boolean function on $n = 6$ variables. While for $r \leq 9$, the running time is reasonable (no more than several hours). Proving unsatisfiability usually takes much longer than finding a satisfying assignment.

III. Circuit Complexity of Symmetric Functions

We show that an upper bound on $\mathrm{gates}(\mathrm{SUM}_k)$ for a constant k, immediately implies an upper bound on $\mathrm{gates}(\mathrm{SUM}_k)$ for all n. That is, a circuit for SUM_k may be used as a building block when computing SUM_n.

Theorem 1. *If* $\mathrm{gates}(\mathrm{SUM}_k) \leq r$*, then for all* n*,*

$$\mathrm{gates}(\mathrm{SUM}_n) \leq \frac{rn}{k - \lceil \log_2(k+1) \rceil} + o(n)\,.$$

Proof. (This proof is a straightforward generalization of an argument communicated to us by Donald Knuth.) Consider each of the input bits x_i as a "pawn" on a chessboard, initially all in the rightmost column. (This is like Napier's binary abacus.) We can take k pawns out of every column and replace them by $\lceil \log_2(k+1) \rceil$ pawns: one in the same column, one in the column to the left, and so on. Repeat this operation until no column contains k or more pawns.

This way, we reduce from n pawns to $O(\log n)$ pawns, at a cost of r for each $(k - \lceil \log_2(k+1) \rceil)$ pawns saved. To finish up, if there are still m pawns left, at most $5m$ more operations (full or half adders) will finish. □

An immediate consequence of this theorem is that

$$\mathrm{gates}(\mathrm{SUM}_n) \leq 5n + o(n)$$

(in fact, it is not difficult to show that the $o(n)$ term in this case is not needed). This folklore upper bound has been known for a long time, but it turns out that it is not optimal. An improved upper bound follows already from an efficient circuit for SUM_7. Namely, in [6, Solution to exercise 479] Knuth shows that $\mathrm{gates}(\mathrm{SUM}_7) \leq 19$. This gives an upper bound $\mathrm{gates}(\mathrm{SUM}_n) \leq 4.75n + o(n)$.

The strongest known upper bound on $\mathrm{gates}(\mathrm{SUM}_n)$ is $4.5n+o(n)$ due to Demenkov et al. [9]. Instead of constructing an efficient circuit for SUM_k, they use the following trick. It is known that $\mathrm{gates}(\mathrm{SUM}_3) = 5$ (full adder), but there exists a circuit of size four that outputs the sum of three input bits, but in a different encoding: $f(x_1, x_2, x_3) = (z_0, z_1)$ such that

$$x_1 + x_2 + x_3 = z_0 + 2 \cdot (z_0 \oplus z_1)\,.$$

This "modified full adder" was used by Stockmeyer [10]. It turns out that two such blocks can be composed into a larger block of size eight that can be, in turn, used as a building block for computing SUM_n. The resulting circuit looks as follows: it first uses $\frac{n}{2}$ to gates to re-encode pairs of input bits (x_i, x_{i+1}) as $(x_i, x_i \oplus x_{i+1})$ and then uses $\frac{n}{2}$ blocks of size eight.

This example illustrates that rather that just searching for efficient circuits for SUM_k, one may need additional ideas for proving strong upper bounds for SUM_n. On the other hand, provided that we have an unlimited computational power, we can get arbitrary close to an optimal upper bound for $\mathrm{gates}(\mathrm{SUM}_n)$, as illustrated by the following theorem that can be viewed as a converse of Theorem 1.

Theorem 2. *Let* $\mathrm{gates}(\mathrm{SUM}_n) \le \alpha n + o(n)$. *For any* $\varepsilon > 0$, *there exists* $k = k(\varepsilon)$ *such that using an optimal circuit for* SUM_k *as a building block gives a circuit of size* $(\alpha+\varepsilon)n+o(n)$ *for* SUM_n.

Proof. Since $\mathrm{gates}(\mathrm{SUM}_n) \le \alpha n + o(n)$, for a large enough value of k,

$$\mathrm{gates}(\mathrm{SUM}_k) \le \left(\alpha + \frac{\varepsilon}{2}\right) \cdot k \,.$$

Since $\log x = o(x)$, for a large enough value of k,

$$\frac{\alpha + \frac{\varepsilon}{2}}{\alpha + \varepsilon} \cdot k \le k - \lceil \log_2(k+1) \rceil \,.$$

Plugging $r = \mathrm{gates}(\mathrm{SUM}_k)$ into Theorem 1 and using these two inequalities gives a circuit for SUM_n of size at most

$$\frac{\mathrm{gates}(\mathrm{SUM}_k)}{k - \lceil \log_2(k+1) \rceil} \cdot n + o(n) \le (\alpha+\varepsilon)n + o(n) \,.$$

□

It should be noted also that the SUM_n function is no easier to compute than all symmetric predicates from B_n. More formally, if $\mathrm{gates}(\mathrm{SUM}_n) \le \alpha n + o(n)$, then $\mathrm{gates}(f) \le \alpha n + o(n)$ for any $f \in B_n$. In order to construct such a circuit for f, let's first compute SUM_n. Since f is symmetric, the resulting $\lceil \log_2(n+1) \rceil$ bits are sufficient for computing f. By the results of Muller [11] and Lupanov [12], any predicate on n variables can be computed by a circuit of size $O(\frac{2^n}{n})$. Hence, the last step costs

$$O\left(\frac{2^{\log_2 n}}{\log n}\right) = o(n) \,,$$

giving a circuit of total size $\alpha n + o(n)$. Hence, improving an upper bound for SUM_n automatically improves upper bounds for *all* symmetric predicates.

A similar effect (composing a circuit for every n from constant size building blocks) occurs for the so called *counting* functions defined as follows:

$$\mathrm{MOD}_n^{m,r}(x_1, \ldots, x_n) = [x_1 + \cdots + x_n \equiv r \bmod m]$$

($[\cdot]$ denotes the Iverson bracket: $[P]$ is 1, if P is true, and is 0 otherwise). Stockmeyer [10] constructed a circuit of size ten that takes four fresh bits (x_1, x_2, x_3, x_4) and two bits that encode a remainder k modulo 4 and outputs two bits encoding the remainder of $x_1+x_2+x_3+x_4+k$ modulo 4. This implies an upper bound $\mathrm{gates}(\mathrm{MOD}_n^4) \le 2.5n$ (when the remainder r is omitted, we mean an upper bound that holds for all values of r). He also proved a lower bound for many symmetric functions. For all counting functions with $m \ge 3$, it implies a lower bound $2.5n - O(m)$.

For almost all counting functions the best known upper bound is just $4.5n+o(n)$ (via SUM_n). The known exceptions are the following. For trivial reasons, $\mathrm{gates}(\mathrm{MOD}_n^2) = n-1$. For larger powers of 2, Demenkov et al. [9] generalized the upper bound by Stockmeyer:

$$\mathrm{gates}(\mathrm{MOD}_n^{2^k}) \le (4.5 - 2^{3-k}) \cdot n + O(k) \,.$$

For $m = 3$, Kojevnikov et al. [5] proved that $\mathrm{gates}(\mathrm{MOD}_3^n) \le 3n$. In [6, Solution to exercise 480], Knuth conjectures that

$$\mathrm{gates}(\mathrm{MOD}_n^{3,r}) = 3n - 5 - [(n+r) \bmod 3 = 0] \,.$$

The conjecture is supported by small values of n: it has been verified by SAT-solvers that the conjecture holds when $n \le 5$ and when $n = 6, r = 0$.

It should be noted, however, that the case of small values of n should not necessarily reflect the general picture. To give an example, define the *threshold* symmetric predicates:

$$\mathrm{THR}_n^t(x_1, \ldots, x_n) = [x_1 + \cdots + x_n \ge t]$$

and let us focus on the $t = 2$. For $n \le 5$,

$$\mathrm{gates}(\mathrm{THR}_n^2) = 3n - 5 \,.$$

Moreover, it is not difficult to design a circuit of size $3n-5$ for any n and there are various ways of doing this (either by induction, or by divide-and-conquer, or by performing one and a half iterations of the bubble sort). Thus, it is tempting to think that $3n-5$ is a tight bound for all n.

However, this intuition fails miserably: it was shown by Dunne [13] that THR_n^2 can be computed by a *monotone* circuit (i.e., a circuit consisting of $\wedge$ and $\vee$ gates only) of size $2n + o(n)$. The idea of the construction is the following. Organize the n input bits into a table of size $\sqrt{n} \times \sqrt{n}$. Then, there are at least two 1's among the input bits if and only if there are at least two rows or two columns each containing at least one 1. This allows to compute THR_n^2 in a recursive fashion: compute disjunctions $r_1, \ldots, r_{\sqrt{n}}$ of all rows and disjunctions $c_1, \ldots, c_{\sqrt{n}}$ of all columns (about $2n$ gates), then return

$$\mathrm{THR}_{\sqrt{n}}^2(r_1, \ldots, r_{\sqrt{n}}) \vee \mathrm{THR}_{\sqrt{n}}^2(c_1, \ldots, c_{\sqrt{n}})$$

($o(n)$ additional gates).

IV. Open Problems and Further Directions

A. Functions

Reduce the following annoying gaps for symmetric functions:

$$\begin{aligned} 2.5n \le \mathrm{gates}(\mathrm{SUM}_n) &\le 4.5n + o(n) \\ 2.5n \le \mathrm{gates}(\mathrm{MOD}_n^3) &\le 3n \\ 2n \le \mathrm{gates}(\mathrm{THR}_n^3) &\le 3n \\ 2n \le \mathrm{gates}(\mathrm{AND}_n, \mathrm{OR}_n, \mathrm{XOR}_n) &\le 2.5n \end{aligned}$$

In the last line, we mean a function from $B_{n,3}$ that outputs three basic symmetric predicates: the conjunction, the disjunction, and the parity of n input bits.

What are other natural or practically important Boolean functions that can be computed using constant size circuits as building blocks? For which Boolean functions we could benefit from knowing efficient circuits for small values of n? Boolean matrix multiplication is one such example. Many graph problems also fall naturally into this category as a graph is naturally represented as a sequence of bits defining its adjacency matrix. For some other problems (like, e.g., the satisfiability problem) the corresponding Boolean encoding is not so natural.

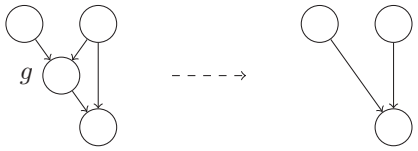
For graphs on n nodes and $n \times n$ matrices, already the input size is of the order n^2, making it impractical to use the SAT-based approach already for $n = 3, 4$. Still, it is not excluded that other approaches could help to construct efficient circuits for these functions for small values of n. We discuss such potential approaches in the next subsection.

B. Approaches

Nowadays, there are many efficient techniques for solving difficult optimization problems in practice: branch-and-bound, gradient descent, LP and ILP solvers. Can any of them be used for finding (at least approximately) optimal circuits in the ranges of n and r where the modern SAT-solvers are helpless?

Can one use local search for finding efficient circuits? We see two potential possibilities.

1) One starts with a large circuit (say, a straightforward CNF or DNF of size $O(n2^n)$) that computes the given function and tries to improve it locally. By a local improvement in this case we mean reducing the size of a circuit. What would be good heuristics for this? E.g., if there are two gates computing the same function, we replace them by a single gate. We can also get rid of the following gate g (provided that it has out-degree 1):



2) One works with a circuit of the required size r and tries to maximize the number of of inputs $(x_1, \ldots, x_n) \in \{0, 1\}^n$ where the circuit agrees with the given function. This is done by local modifications of the circuit: changing the underlying graph (rewiring) or changing the binary operations computed at the gates.

ACKNOWLEDGMENTS

I would like to thank Alexander Golovnev for fruitful discussions and proofreading the initial version of the paper.

REFERENCES

[1] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell Systems Technical Journal*, vol. 28, pp. 59–98, 1949.

[2] M. G. Find, A. Golovnev, E. A. Hirsch, and A. S. Kulikov, "A better-than-3n lower bound for the circuit complexity of an explicit function," in *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, I. Dinur, Ed. IEEE Computer Society, 2016, pp. 89–98. [Online]. Available: https://doi.org/10.1109/FOCS.2016.19

[3] R. Williams, "Applying practice to theory," *SIGACT News*, vol. 39, no. 4, pp. 37–52, Nov. 2008. [Online]. Available: http://doi.acm.org/10.1145/1466390.1466401

[4] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 0: Introduction to Combinatorial Algorithms and Boolean Functions (Art of Computer Programming)*, 1st ed. Addison-Wesley Professional, 2008.

[5] A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev, "Finding efficient circuits using sat-solvers," in *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, ser. Lecture Notes in Computer Science, O. Kullmann, Ed., vol. 5584. Springer, 2009, pp. 32–44. [Online]. Available: https://doi.org/10.1007/978-3-642-02777-2_5

[6] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*, 1st ed. Addison-Wesley Professional, 2015.

[7] https://github.com/alexanderskulikov/circuit-synthesis.

[8] http://www-cs-faculty.stanford.edu/~knuth/programs.html.

[9] E. Demenkov, A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev, "New upper bounds on the boolean circuit complexity of symmetric functions," *Inf. Process. Lett.*, vol. 110, no. 7, pp. 264–267, 2010. [Online]. Available: https://doi.org/10.1016/j.ipl.2010.01.007

[10] L. J. Stockmeyer, "On the combinational complexity of certain symmetric boolean functions," *Mathematical Systems Theory*, vol. 10, pp. 323–336, 1977.

[11] D. E. Muller, "Complexity in electronic switching circuits," *IRE Transactions on Electronic Computers*, vol. EC-5, pp. 15–19, 1956.

[12] O. Lupanov, "A method of circuit synthesis," *Izvestiya VUZov, Radiofizika*, vol. 1, pp. 120–140, 1959.

[13] P. E. Dunne, "Techniques for the analysis of monotone boolean networks," Ph.D. dissertation, University of Warwick, 1984.