

Reviewing bounds on the circuit size of the hardest functions

Gudmund Skovbjerg Frandsen, Peter Bro Miltersen

*BRICS*¹

*Department of Computer Science, University of Aarhus, IT-parken, Aabogade 34,
DK-8200 Aarhus N, Denmark*

Abstract

In this paper we review the known bounds for $L(n)$, the circuit size complexity of the hardest Boolean function on n input bits. The best known bounds appear to be

$$\frac{2^n}{n} \left(1 + \frac{\log n}{n} - O\left(\frac{1}{n}\right)\right) \leq L(n) \leq \frac{2^n}{n} \left(1 + 3\frac{\log n}{n} + O\left(\frac{1}{n}\right)\right)$$

However, the bounds do not seem to be explicitly stated in the literature. We give a simple direct elementary proof of the lower bound valid for the full binary basis, and we give an explicit proof of the upper bound valid for the basis $\{\neg, \wedge, \vee\}$.

Key words: Computational complexity

1 Introduction

Shannon introduced the Boolean circuit size as a complexity measure [1], and showed upper and lower bounds for the minimum number of gates, $L(n)$, needed in a Boolean circuit for computing a hardest function in B_n , the set of Boolean functions with n inputs and 1 output. Shannon proved that for every $\epsilon > 0$ and n sufficiently large [1]

$$(1 - \epsilon) \frac{2^n}{n} < L(n) < \frac{2^{n+2}}{n}$$

Email addresses: gudmund@daimi.au.dk (Gudmund Skovbjerg Frandsen),
bromille@daimi.au.dk (Peter Bro Miltersen).

¹ Basic Research in Computer Science, Centre of the Danish National Research Foundation.

The first improvement came when Lupanov showed a better upper bound [2]:

$$L(n) \leq \frac{2^n}{n} \left(1 + O\left(\frac{1}{\sqrt{n}}\right)\right)$$

Lupanov essentially closed the gap so that together with Shannons original lower bound it was now known that

$$L(n) = \frac{2^n}{n} \pm o\left(\frac{2^n}{n}\right)$$

Later Lutz showed that for every real $\alpha < 1$ and almost every n [3]

$$\frac{2^n}{n} \left(1 + \alpha \frac{\log n}{n}\right) < L(n)$$

Lutz really showed a much more general result, and we give in section 2 a simple direct proof that

$$\frac{2^n}{n} \left(1 + \frac{\log n}{n} - O\left(\frac{1}{n}\right)\right) < L(n)$$

Our proof is robust in that a change of the basis or simple improvements seem only to effect the $O(\frac{1}{n})$ term. One might take this as an indication that it is also possible to give tight bounds on the second order term in the expression for $L(n)$. It appears that several text book authors have observed that one can make a tighter analysis of Lupanovs construction [4–6], and they more or less explicitly deduce that $L(n) \leq \frac{2^n}{n} (1 + O(\frac{\log n}{n}))$. We present such a tight analysis in section 3 where we particularly seek to minimize the constant hidden under the big-Oh notation obtaining the bound

$$L(n) \leq \frac{2^n}{n} \left(1 + 3 \frac{\log n}{n} + O\left(\frac{1}{n}\right)\right)$$

2 Lower bound

We will demonstrate the lower bound by showing how to transform a Boolean circuit into a list of instructions for a simple stack machine and then use a counting argument to bound the length of the latter.

Each instruction for the stack machine is either a push or a binary Boolean operation. Only the push operation has an argument, which is the number of an input or an (earlier) Boolean operation. Inputs are numbered $1, \dots, n$, and Boolean operations are numbered $n + 1, \dots, n + s$ in a stack program with s Boolean operations. Execution of a push operation places an input or an earlier computed bit designated by the argument on top of the stack. Each Boolean operation removes the 2 topmost elements of the stack and writes a

single element onto the stack. After execution of all instructions, there should be exactly one element left on the stack, namely the result.

Algorithm 1 Translation of circuit to stack program

Require: circuit C of size s described as set of gates $\{g_i \leftarrow g_{i_1} \text{ op } g_{i_2} \mid i = n + 1, \dots, n + s\}$ where g_1, \dots, g_n are inputs and g_{n+s} is output gate

Ensure: Stack program P computes same function as C

- 1: Let P initially be empty
- 2: virtualPush($n + s$)

Procedure virtualPush(i)

- 3: **if** g_i is an input **then**
 - 4: add program line “push i ” to P
 - 5: **if** g_i is already computed by the j th Boolean operation in P **then**
 - 6: add program line “push $n + j$ ” to P
 - 7: **if** gate $g_i \leftarrow g_{i_1} \text{ op } g_{i_2}$ is not computed so far **then**
 - 8: virtualPush(i_1)
 - 9: virtualPush(i_2)
 - 10: add program line “operation op ” to P
-

Algorithm 1 describes a recursive procedure virtualPush that given the output gate of the circuit will construct a stack program computing the same function as the circuit, and the number of gates in the circuit will be equal to the number of Boolean operations in the stack program.

If the stack program contains s Boolean operations, then it contains $s + 1$ push operations. To see this observe that when executing the stack program, each push operation increases the stack size by one and each Boolean operation decreases the stack size by one. Since the stack is initially empty and it contains only the single output value at the end, there must be exactly $s + 1$ push operations.

The argument of a push operation can be represented by $\lceil \log(n + s) \rceil$ bits. A single bit is needed to distinguish push operations from Boolean operations, and 4 bits suffice to distinguish the Boolean operations. In total the stack program can be described using at most $(s + 1)(c + \log(n + s))$ bits, for $c = 7$. Since there are 2^{2^n} distinct Boolean functions on n inputs, for some function the optimal circuit size s must satisfy that $(s + 1)(c + \log(n + s)) \geq 2^n$.

The last inequality implies that $s > 2^n/n \cdot (1 + \log n/n - c/n)$ for n sufficiently large. We will argue this lower bound by way of contradiction, so we assume that $s \leq 2^n/n \cdot (1 + \log n/n - c/n)$, which by a simple rewriting is equivalent to $n + s \leq 2^n/n \cdot (1 + \log n/n - c/n + n^2/2^n)$. Using that $\log(1 + x) \leq x \log e$, the assumption implies that $\log(n + s) \leq n - \log n + (\log n/n - c/n + n^2/2^n) \log e$. Combining with the inequality of the previous paragraph, we see that

$$\begin{aligned}
2^n &\leq (s+1)(c + \log(n+s)) \\
&\leq \frac{2^n}{n} \left(1 + \frac{\log n}{n} - \frac{c}{n} + \frac{n}{2^n}\right) (n - \log n + c + \left(\frac{\log n}{n} - \frac{c}{n} + \frac{n^2}{2^n}\right) \log e) \\
&\leq \frac{2^n}{n^2} \left(n + \log n - c + \frac{n^2}{2^n}\right) (n - \log n + c + \left(\frac{\log n}{n} - \frac{c}{n} + \frac{n^2}{2^n}\right) \log e) \\
&\leq \frac{2^n}{n^2} (n^2 - \log^2 n + O(\log n)) \\
&< 2^n \quad \text{for } n \text{ sufficiently large}
\end{aligned}$$

Thus we have a contradiction proving that $s > 2^n/n \cdot (1 + \log n/n - c/n)$ for n sufficiently large.

Note that the two most significant terms in the bound seem robust. Simple restrictions on the basis or any simple improvements in the stack representation seem only to influence the value of c . One might also try to improve the lower bound by using that many stack programs compute the same Boolean function. By changing the order of lines 8 and 9 in Algorithm 1 (and changing the Boolean operation in line 10 correspondingly), one may generate up to 2^s distinct stack programs that all compute the same Boolean function. However, a formalisation of this argument will only influence the value of c , and cannot change the two most significant terms in the bound.

We have shown

Theorem 1

$$L(n) \geq \frac{2^n}{n} \left(1 + \frac{\log n}{n} - O\left(\frac{1}{n}\right)\right)$$

3 Tight analysis of Lupanovs upper bound

Recall the (k, s) -Lupanov representation of a Boolean function f on n bits as it is described by Savage [6].

This representation has the form

$$f(\mathbf{x}) = \bigvee_{i=1}^p \bigvee_{\mathbf{v}} f_{i,\mathbf{v}}^{(r)}(\mathbf{a}) \wedge f_{i,\mathbf{v}}^{(c)}(\mathbf{b})$$

where input $\mathbf{x} = (x_1, \dots, x_n)$ is divided in two parts $\mathbf{a} = (x_1, \dots, x_k)$ and $\mathbf{b} = (x_{k+1}, \dots, x_n)$. The 2^k possible \mathbf{a} -tuples of bits are divided into $p = \lceil 2^k/s \rceil$ lists A_1, \dots, A_p containing s tuples each (though A_p may contain fewer tuples). \mathbf{v} ranges over bit tuples of length s . $f_{i,\mathbf{v}}^{(c)}(\mathbf{b}) = 1$ precisely when \mathbf{v} describes the table of function values $f(\mathbf{a}, \mathbf{b})$ arising when \mathbf{a} runs through the tuples in

A_i . $f_{i,\mathbf{v}}^{(r)}(\mathbf{a}) = 1$ precisely when there is some j such that \mathbf{a} is the j th tuple in A_i and the j th bit of \mathbf{v} is 1.

A circuit for f may be constructed in three steps. In the first step, we use $2(2^k + 2^{n-k})$ gates to compute all minterms over \mathbf{a} and \mathbf{b} . In the second step we use $p2^{n-k}$ or-gates to compute $f_{i,\mathbf{v}}^{(c)}$ for all i, \mathbf{v} , and we use $p2^s$ or-gates to compute $f_{i,\mathbf{v}}^{(r)}$ for all i, \mathbf{v} . In the third step f is computed using $2p2^s$ gates.

Arguments for these bounds are given by both Wegener [5] and Savage [6] except for our bound $p2^s$ on the computation of $f_{i,\mathbf{v}}^{(r)}$, where they mention only the weaker bound 2^{k+s} . To see that our bound is valid, observe that we may compute $f_{i,\mathbf{v}}^{(r)}$ separately for each i . This accounts for the factor p . For a fixed i we first compute those $f_{i,\mathbf{v}}^{(r)}$ where \mathbf{v} contains a single 1-bit, then those where \mathbf{v} contains two 1-bits, etc. In this way we need only use one or-gate per \mathbf{v} , implying the stated bound.

By the above arguments we need at most $2(2^k + 2^{n-k}) + p2^{n-k} + 3p2^s$ gates to compute f . We may eliminate p from the expression, when using that $p \leq 1 + 2^k/s$. This results in the upper bound $2^n/s + 2 \cdot 2^k + 3(2^{n-k} + 2^s + 2^{k+s}/s)$ on the number of gates. Taking $k = \lceil 2 \log n \rceil$ and $s = \lceil n - 3 \log n \rceil$, all terms but the first are bounded by $O(2^n/n^2)$ and we may bound the first term separately

$$\begin{aligned} \frac{2^n}{s} &\leq \frac{2^n}{n - 3 \log n} \\ &= \frac{2^n}{n} \left(1 + \frac{3 \log n}{n - 3 \log n}\right) \\ &= \frac{2^n}{n} \left(1 + 3 \frac{\log n}{n} + O\left(\frac{\log^2 n}{n^2}\right)\right) \end{aligned}$$

Combining the bounds, we have shown

Theorem 2

$$L(n) \leq \frac{2^n}{n} \left(1 + 3 \frac{\log n}{n} + O\left(\frac{1}{n}\right)\right)$$

Acknowledgements

We would like to thank the anonymous referee for comments that greatly improved the presentation of the results.

References

- [1] C. E. Shannon, The synthesis of two-terminal switching circuits, *Bell System Tech. J.* 28 (1949) 59–98.
- [2] O. B. Lupanov, The synthesis of contact circuits, *Dokl. Akad. Nauk SSSR (N.S.)* 119 (1958) 23–26.
- [3] J. H. Lutz, Almost everywhere high nonuniform complexity, *J. Comput. System Sci.* 44 (2) (1992) 220–258.
- [4] R. G. Nigmatullin, *Slozhnost bulevykh funktsii*, Kazan. Gos. Univ., Kazan', 1983.
- [5] I. Wegener, *The complexity of Boolean functions*, Wiley-Teubner Series in Computer Science, John Wiley & Sons Ltd., Chichester, 1987.
- [6] J. E. Savage, *Models of Computation*, Addison Wesley, 1998.