Searching for circuit complexity lower bounds using SAT solvers

August Taylor

April 5, 2022

Abstract

Where the abstract will go

Contents

1 Introduction

1.1 Motivation

Finding lower bounds on the size of Boolean circuits computing given Boolean functions is one of the most fundamental open problems in computer science. Despite the fact that the majority of functions have exponential circuits, the best lower bounds proven on unrestricted circuits are only linear [**?**] and we still do not have optimal circuits or close upper and lower bounds for many important functions. Results in this area can be used to separate computational complexity classes, possibly even P and NP (i.e. to show that a large class of fundamental and practically useful problems do not have polynomially fast algorithms). [**?**]

If functions with circuits of a given complexity could be identified by an efficient algorithm, this could be used to construct efficient learning algorithms for circuits of a smaller complexity. [**?**] Furthermore, such an algorithm could be used to break pseudorandom generators (i.e. distinguish random from pseudorandom inputs), which would have major implications in the field of cryptography. [?]

A heuristic approach is to find minimal circuits for small instances of the functions. This can be used to prove new bounds [?] and may lead to theoretical insights on the structure of their optimal circuits generally [?]. Finding efficient circuits is also necessary in practice when designing electronic systems.

One approach to automating the search for efficient circuits is to reduce the problem of designing a correct circuit (logical design synthesis) to the Boolean satisfiability problem (SAT). Existing algorithms (SAT solvers) can then be used to solve the problem, either finding a correct circuit of fixed size or generating a proof that none exists [?].

1.2 Previous work

Circuits are a popular model of non-uniform computation. In the past there have been theoretical proofs of lower bounds for restricted circuit classes, e.g. AC0[?][?], AC0[k] (i.e. AC0 additionally with MOD-k gates)[?][?], and monotone circuits[?]. However, no lower bounds better than 5n-c[?] have been found for general circuits. Furthermore, there are results stating that it is impossible to use circuit lower bounds to separate P and NP by relativisation[?], algebrization[?], or natural proofs[?]. It has been proved that circuit lower bounds are hard to prove using resolution and resolution with k-DNFS, and it is conjectured that they are also hard under stronger proof systems such as Frege systems[?]. Due to this difficulty in theoretical progress, we would like to investigate the problem empirically.

Kamath et al [?] propose a reduction from logical design synthesis to SAT which fixes the DNF structure of the circuit and number of disjunctions used. Experiments using this reduction to find minimal circuits for 2-4 bit adders and multipliers are reported in [?]. Kojevnikov et al [?] demonstrate a more general reduction allowing any circuit with a fixed number of gates, proving linear bounds for MOD_3^n circuits over different Boolean bases. This was done by checking for circuit 'building blocks' with 5 to 11 gates. In [?] the same reduction was used to improve large circuit building blocks by searching for smaller versions of their sub-circuits. Specifically, linear upper bounds were proved or verified for SUM, MOD_4^n, and MOD_3^n by checking for circuits with up to 13 gates.

1.3 Our contributions

We investigate feasible ways to find small circuits using SAT reductions. Previous research in this area has focused on a small number of target Boolean functions and a single reduction at a time. Instead, we use multiple combinations of reductions and SAT solvers to see which leads to the best performance. We test them on a range of symmetric functions whose lower bounds are already somewhat understood, in order to analyse the performance in relation to the bound.

We also search for a general relationship between solver performance and minimum circuit size by applying our methods to a sample of randomly generated functions. We then use solvers to prove a result, that a lower bound conjecture about MOD-3 functions due to Knuth[**?**] does not hold for the de Morgan basis.

From these experiments we note that available solvers running on a personal computer can decide the existence of some circuits of size 11 in just a few hours, while others of size 11 and up can take many hours on a high-powered computing cluster. That is to say, the amount of time taken to solve comparable problems can vary considerably.

*Extension axioms - should actually try them first

2 Background

2.1 General setting

Denote by $B_{n,m}$ the set of all Boolean functions $f : \{0,1\}^n \to \{0,1\}^m$ and let $B_n = B_{n,1}$. 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 called inputs. Nodes of in-degree 2 are assigned functions from A and are called gates. Some nodes may be also be marked as outputs.

Without loss of generality, we can also assert that every gate has a larger index than both of its predecessors (i.e. the gates are sorted topologically w.r.t. the used numbering); that every gate's second predecessor has a larger index than its first predecessor; and that the last gate is an output. [**?**]

A circuit c of size N is said to compute the function $f \in B_{n,m}$ if, for all input vectors $v \in \{0,1\}^n$ in the range of f, $c(v) = f(v)$, where $c(v) = (o_1, ...o_m)$ are the values of the output gates of c, each input x_i takes value v_i, and each gate takes the value of its assigned function when applied to the value of its two predecessor nodes.

In this paper, we mainly consider circuits over the De Morgan basis

$$C_2 = \{\vee, \wedge, \neg\}$$

where $\neg$ denotes the Boolean function which outputs the negation of its first argument.

The size of a circuit is its number of gates. We define the circuit existence problem as the question of whether, given a truth table for a function $f : \{0,1\}^n \to \{0,1\}^m$, there exists a circuit of size N computing f.

2.2 SAT solvers

SAT solvers have previously seen success in proving mathematical theorems, most famously in 2016 when Heule et al solved the Boolean Pythagorean Triples problem[?] and in 2020 when Brakensiek, Heule et al solved Keller's conjecture[?]. In general, SAT solvers have the advantage of being effective even when heuristics are not known for solving the original problem, as in the case of circuit existence.

We used three SAT solvers, all based on conflict-driven clause learning. This is an algorithm closely related to DPLL, and which also makes use of the resolution proof system.[?][?]

MiniSAT[?] was the winner of the 2005 SAT Competition and was last updated to version 2.2 in 2010[?]. Despite being outdated, it is well-documented and the other solvers we used were based on it, making it a good baseline for performance.

PicoSAT is a solver inspired by miniSAT 1.14, with a focus on improving low-level performance by using optimised data structures[?]. It was submitted in the 2010 SAT-Race.

MapleSAT is the most recent solver, with variants having won the SAT Competition 2016 and placed second in 2017. It was based off MiniSAT 2.2 but has a different branching heuristic, incorporating ideas from machine learning[?].

*I'm not sure why we are using picoSAT - there is still time to re-run the results if we want to replace it.

2.3 Extension axioms

2.4 Reduction to SAT

We encoded the circuit existence problem as a Boolean formula using three reductions.

2.4.1 Kojevnikov reduction

The first is due to Kojevnikov et al [?] who present it together with bounds on the growth rate of the formula. The basis of the circuit can be specified as any subset of B_2.

It uses the following variables:

1. $t_{ib_1b_2}$ ($n \leq i < n+N, 0 \leq b_1 < 2, 0 \leq b_2 < 2$) is the value of the i-th gate if its first predecessor takes value b_1 and its second takes value b_2. The four variables $t_{i00}, t_{i01}, t_{i10}, t_{i11}$ thus define the function assigned to the i-th gate. This gives $O(N)$ variables.

2. c_{ikj} ($n \leq i < n+N-1, 0 \leq k < 2, 0 \leq j < n+N$) is true iff j is the k-th predecessor of i. This gives $O(N^2)$ variables.

3. o_{ij} ($n \leq i < n+N, 0 \leq j < m$) is true iff the j-th output is computed by the i-th gate. This gives $O(Nm)$ variables.

4. v_{it} ($0 \leq i \leq n+N, 0 \leq t < 2^n$) is the output value of the i-th gate if the input variables take values represented by the bits of t. These are used

to constrain correctness of the circuit on all possible inputs, including its outputs matching the truth table of the encoded function.

We encode the requirements for the circuit by the following clauses:

1. The binary function assigned to each gate belongs to our desired basis.

2. For all (i, k), exactly one variable c_{ikj} is true (there is exactly one k-th predecessor of the i-th gate). This gives $O(N^3)$ 2-clauses and $O(N)$ $O(N)$-clauses.

3. For all j, exactly one variable o_{ij} is true (the j-th output is computed by exactly one gate). This gives $O(N^2 m)$ 2-clauses and $O(m)$ $O(N)$-clauses.

4. For all $0 \leq i < n$ and $0 \leq t < 2^n$, v_{it} is equal to the corresponding bit in t. This gives $O(n \cdot 2^n)$ 1-clauses.

5. For all $n \leq i < n + N$ and $0 \leq t < 2^n$, v_{it} is equal to the value computed by the i-th gate. This constrains all the gates with predecessors. Clauses of this type are written for all $n \leq i < n + N$, $n \leq j_0 < i$, $j_0 \leq j_1 < i$, $0 \leq i_0 < 2$, $0 \leq i_1 < 2$, $0 \leq r < 2^n$, and look as follows:

$$\neg c_{i0j_0} \vee \neg c_{i1j_1} \vee \neg(v_{j_0r} = i_0) \vee \neg(v_{j_1r} = i_1) \vee (v_{ir} = t_{ii_0i_1})$$

   This gives $O(N^3 \cdot 2^n)$ 6-clauses.

6. The outputs of the circuit match the truth table. Clauses of this type are written for all $0 \leq k < m, 0 \leq r < 2^n, n \leq i < n + N$, and look as follows:

$$\neg o_{ik} \vee (v_{ir} = value_{kr})$$

   where $value_{kr}$ is the required value of the k-th output when the circuit is given input r, according to the truth table. This gives $O(N2^n m)$ clauses.

2.4.2 Razborov reduction

The second is due to Razborov [**?**], assumes the function outputs only a single Boolean value, and requires a circuit over the basis C_2. Note that it has a lower growth rate (in terms of number of clauses) than the other two reductions, suggesting it encodes the requirements more efficiently. Let f be the target function with n inputs, and let t be the number of gates required in the circuit.

It uses the following variables:

1. y_{av} ($a \in 0, 1^n, v \in [t]$) is the value taken by node v on circuit input a. This gives $O(t2^n)$ variables.

2. $y_{a\nu v}$ ($a \in 0, 1^n, \nu \in 1, 2, v \in [t]$) is the value of the ν-th predecessor to v on input a. This gives $O(t2^n)$ variables.

3. $Fanin(v)$ is 0 if v is a $\neg$-gate and 1 otherwise. This gives $O(t)$ variables.

4. $Types(v)$ - when $Fanin(v) = 1$, this is 0 if v is a $\wedge$-gate and 1 if v is a $\vee$-gate. This gives $O(t)$ variables.

5. $InputType_\nu(v)$ is 0 if the ν-th predecessor to v is a constant or an input, 1 if it is another gate. This gives $O(t)$ variables.

6. $InputType'_\nu(v)$ - when $InputType_\nu(v) = 0$, this is 0 if the ν-th predecessor to v is a constant, 1 if it is an input. This gives $O(t)$ variables.

7. $InputType''_\nu(v)$ - when $InputType_\nu(v) = InputType'_\nu(v) = 0$, this is the value of the constant ν-th predecessor to v. This gives $O(t)$ variables.

8. $InputVar_\nu(v, i)$ $(i \in [n])$ - when $InputType_\nu(v) = 0, InputType'_\nu(v) = 1$, this is 1 iff the ν-th predecessor to v is the i-th input. This gives $O(tn)$ variables.

9. $INPUTVAR_\nu(v, i)$ is equal to $\bigvee_{i'<i} InputVar_\nu(v, i')$, which will be used to bound the fan-in below. This gives $O(tn)$ variables.

10. $InputNode_\nu(v, v')$ $(v' < v)$ - when $InputType_\nu(v) = 1$, this is 1 iff ν-th predecessor to v is the gate v'. This gives $O(t^2)$ variables.

11. $INPUTNODE_\nu(v, v')$ is analogous to $INPUTVAR_\nu(v, i)$. This gives $O(t^2)$ variables.

We encode the requirements for the circuit by the following expressions:

1. Constant predecessors take the correct values, giving $O(t)$ clauses:
   $\neg InputType_\nu(v) \wedge \neg InputType'_\nu(v) \rightarrow (y_{a\nu v} = InputType''_\nu(v))$

2. Gates have at most one input per input predecessor, giving $O(tn^2)$ clauses:
   $\neg InputType_\nu(v) \wedge InputType'_\nu(v) \rightarrow \neg(InputVar_\nu(v, i) \wedge InputVar_\nu(v, i'))$ for $i \neq i'$

3. INPUTVAR variables take the correct value, giving $O(tn)$ clauses:
   $\neg InputType_\nu(v) \wedge InputType'_\nu(v) \rightarrow (INPUTVAR_\nu(v, i) = (InputType_\nu(v), i-1) \vee InputVar_\nu(v, i)))$, where $InputType_\nu(v, 0) = 0$

4. Gates have at least one input per input predecessor, giving $O(t)$ clauses:
   $\neg InputType_\nu(v) \wedge InputType'_\nu(v) \rightarrow INPUTVAR_\nu(v, n)$

5. Input predecessors take the correct values, giving $O(tn \cdot 2^n)$ clauses:
   $\neg InputType_\nu(v) \wedge InputType'_\nu(v) \wedge InputVar_\nu(v, i) \rightarrow (y_{a\nu v} = a_i)$

6. The analogous clauses to the above, but for InputNode and gate predecessors.

7. Gates take the correct value, giving $O(t2^n)$ clauses.

8. The final gate (i.e. the output) matches the truth table, giving $O(2^n)$ clauses.

2.4.3 Naive reduction

The third reduction is straightforward. It also assumes the function outputs only a single Boolean value and requires a circuit over the basis C_2.

It uses the following variables:

1. e_a^i ($n \le i < N, a \in \{0,1\}^n$) is the value of the i-th gate when the input to the circuit is a. This gives $O(N \cdot 2^n)$ variables.

2. $x_{i,j}$ ($0 \le i < N+n, 0 \le j < i$) is true iff j is a predecessor of i. This gives $O(N \cdot (N+n))$ variables.

3. g_i^0, g_i^1 ($n \le i < N$) together encode the function assigned to the i-th gate by their truth values, giving $2N$ variables:

g_i^0	g_i^1	Function
0	0	$\neg$
0	1	$\vee$
1	0	$\wedge$
1	1	$\neg$

We encode the requirements for the circuit by the following clauses:

1. For all (i,j,k) with $n \le i < n+N$, $0 \le j < i$ and $j < k < i$, and for all $a \in \{0,1\}^n$, if j and k are both predecessors of i, then e_a^i must take the value computed by the ith gate's assigned function on e_a^j and e_a^k. Clauses must be added for each function in the basis. For example, the expression for when the ith gate is an $\wedge$-gate:

$$(g_i^0 \wedge \neg g_i^1) \wedge (x_{i,j} \wedge x_{i,k}) \rightarrow (e_a^i \leftrightarrow e_a^j \wedge e_a^k)$$

   Note that the gate must compute the conjunction of any two predecessors' variables. This means it can have at most two predecessors, otherwise the conjunctions of different pairs of predecessors may disagree and the gate will have no correct value. This gives $O(2^n \cdot N^3)$ 7-clauses.

2. For all i with $n \le i < n+N$, gate i must have at least one predecessor. This gives $O(N)$ $O(N+n)$-clauses.

3. For all (i,j) with $n \le i < n+N$, $0 \le j < i$, if i is an $\vee$-gate or $\wedge$-gate with j its predecessor, it must have another distinct predecessor (i.e. it must have at least 2 overall). For example, the expression for when the ith gate is an $\wedge$-gate:

$$(g_i^0 \wedge \neg g_i^1) \rightarrow \left(x_{i,j} \rightarrow \bigvee_{j<k<i} x_{i,k} \right)$$

   This gives $O(N^2)$ $O(N)$-clauses.

4. For all i with $0 \leq i < n$, $a \in \{0,1\}^n$, the i-th input must equal a_i. This gives $O(n \cdot 2^n)$ 1-clauses.

5. For all $a \in \{0,1\}^n$, the final m gates (i.e. the outputs) should agree with the output values given for a in the target truth table. This gives $O(2^n)$ 1-clauses.

3 Results

Timed experiments were carried out on a 3.400GHz Intel i3-8130U processor running Linux. The times shown were obtained by running the software on the problem 3 times and taking the median.

We chose to investigate the existence of circuits up to 9 gates, as these were reliably solved within less than 9 hours on the hardware described.

3.1 Comparison of Boolean functions

We investigated the relationship between the size of a circuit and the time taken for a SAT solver to confirm or deny its existence. We encoded a range of simple symmetric functions using the Kulikov encoding *define them more clearly:

1. AND on 7 and 8-bit inputs

2. Parity on 3- and 4-bit inputs

3. MOD_3^0 and MOD_4^0 on 3- and 4-bit inputs

4. Majority on 3 and 4-bit inputs

5. Partial parity - Explain why partial functions are useful for encoding lower bounds on big input-lengths. But also need to find a way to get good data for them (ie. more than 1 second per problem...)

The size of the minimal circuit is shown in red. It was found that the time needed tended to spike on or around the optimal circuit size. This may be because it is difficult to confirm or rule out circuits that come close to satisfying the requirements. A similar phenomenon was reported in [**?**].

A complete display of graphs can be found in the appendix.

3.2 Comparison of SAT solvers and reductions

We investigated whether there was a relationship between the reduction used to encode a Boolean function, the solver used, and the time taken by the solver.

It can be seen that MapleSAT on the Kulikov encoding performs the best. Apart from this, the effect of the reduction and the solver seem to be independent, with MapleSAT consistently performing better than MiniSAT and

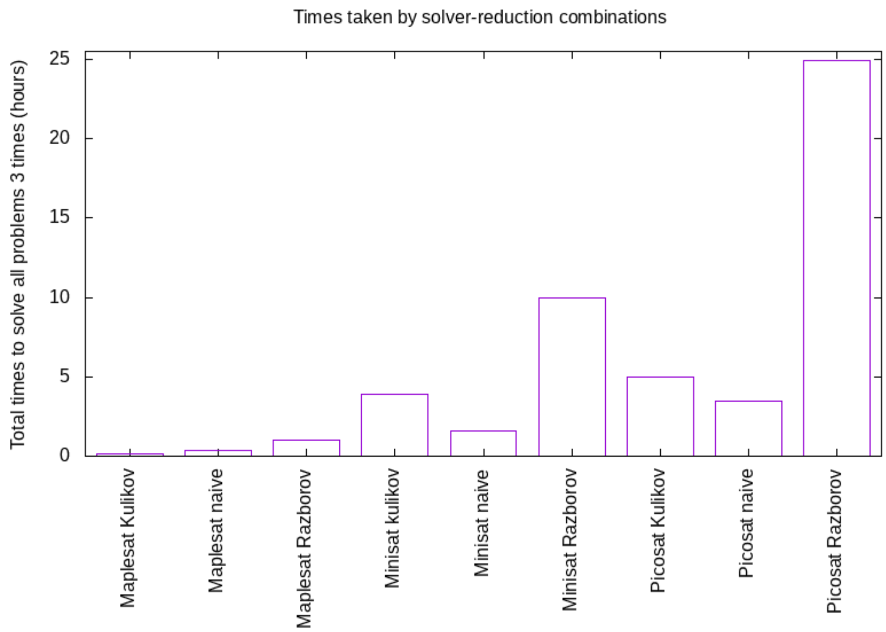

Figure 1: Comparison of times taken by SAT solvers on different reductions

PicoSAT, and the naive and Kulikov reductions performing better than the Razborov reduction. It is possible that the efficient encoding of the Razborov reduction is detrimental to the performance of the solver. However, these results are only applicable to small values of n. **however I should take out the constants and re-run the experiments to be sure it's not that..

3.3 Random functions

16 truth tables were generated for 3 and 4-bit inputs using a pseudorandom number generator (specifically, xorshift128+ due to the Javascript implementation) simulating a uniform distribution. The tables were encoded with the Kulikov reduction and solved using MapleSAT. The times taken to check for circuits of different sizes are shown in change to 64 and show the tables

images

The distribution of minimal circuit sizes needed to implement these functions are shown in 3.3 and 3.3.

A further 32 truth tables were generated for 4-bit inputs. While we expect a large proportion of circuits to be hard for large input sizes [**?**], the usual counting argument breaks down for such small values of n (ask for help - why?). However,

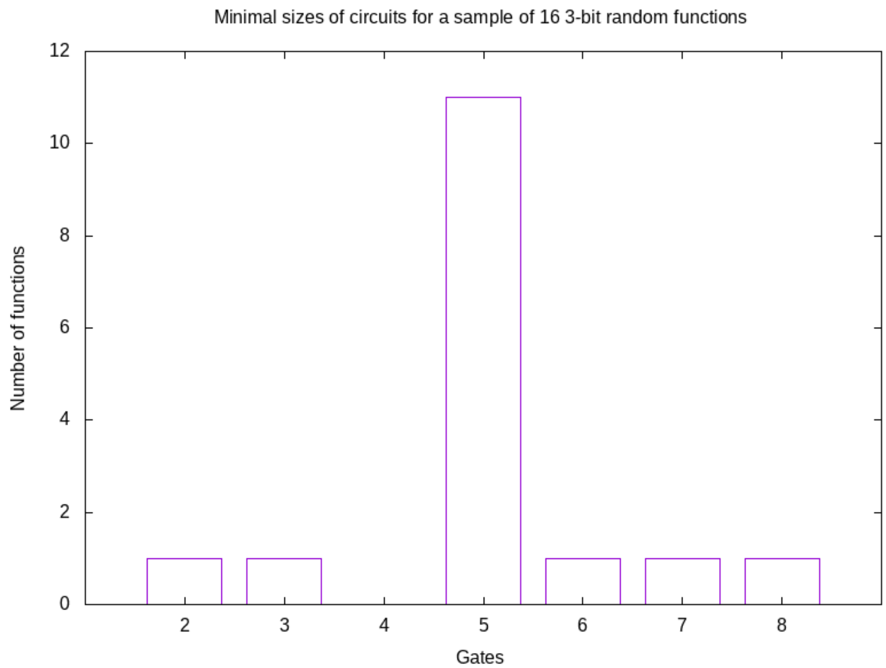

Figure 2: Sizes of optimal circuits for 3-bit functions with randomly chosen truth tables

using the SAT solver we can check the number of hard circuits directly.

image

3.4 Extension axioms

4 Discussion

4.1 Reflection

4.2 Conclusions

4.3 Further directions

References

[1] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1), feb 2009.

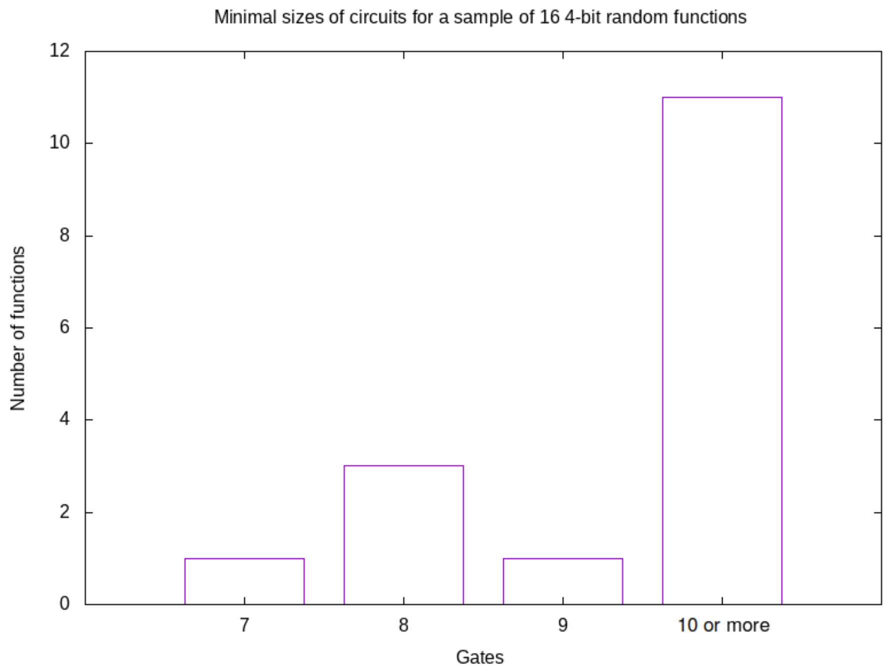

Figure 3: Sizes of optimal circuits for 4-bit functions with randomly chosen truth tables

[2] M. Ajtai. 11-formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983.

[3] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[4] Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $P = ?NP$ question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[5] Armin Biere. Picosat essentials. *JSAT*, 4:75–97, 05 2008.

[6] Ravi B Boppana and Michael Sipser. The complexity of finite functions. In *Algorithms and complexity*, pages 757–804. Elsevier, 1990.

[7] Joshua Brakensiek, Marijn Heule, John Mackey, and David Narváez. The resolution of keller's conjecture. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 48–65, Cham, 2020. Springer International Publishing.

[8] Marco Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Learning algorithms from natural proofs. In *Computational Complexity Conference*, 2016.

[9] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, pages 502–518, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[10] Niklas Eén and Niklas Sörensson. minisat github repository. https://github.com/niklasso/minisat/tree/master/doc, 2010.

[11] Giovani Gomez Estrada. A note on designing logical circuits using sat. In *International Conference on Evolvable Systems*, pages 410–421. Springer, 2003.

[12] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*, pages 260–270, 1981.

[13] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer. In *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 228–245. Springer International Publishing, 2016.

[14] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of 5n o(n) for boolean circuits. pages 353–364, 08 2002.

[15] AP Kamath, NK Karmarkar, KG Ramakrishnan, and MGC Resende. An interior point approach to boolean vector function synthesis. In *Proceedings of 36th Midwest Symposium on Circuits and Systems*, pages 185–189. IEEE, 1993.

[16] Donald E Knuth. *The art of computer programming, Volume 4, Fascicle 6: Satisfiability*. Addison-Wesley Professional, 2015.

[17] Arist Kojevnikov, Alexander S Kulikov, and Grigory Yaroslavtsev. Finding efficient circuits using sat-solvers. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 32–44. Springer, 2009.

[18] Jan Krajíček. *Proof complexity*, volume 170. Cambridge University Press, 2019.

[19] Alexander S Kulikov. Improving circuit size upper bounds using sat-solvers. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 305–308. IEEE, 2018.

[20] Alexander S. Kulikov and Nikita Slezkin. Sat-based circuit local improvement, 2021.

[21] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and K. Czarnecki. Learning rate based branching heuristic for sat solvers. In *SAT*, 2016.

[22] Joao Marques-Silva, Ines Lynce, and Sharad Malik. *The Handbook of Satisfiability, Chapter 4: Conflict-driven clause learning SAT solvers*, pages 133–182. Frontiers in Artificial Intelligence and Applications. IOS Press BV, 2021. Publisher Copyright: © 2021 The authors and IOS Press. All rights reserved.

[23] Alexander A Razborov. Lower bounds for the monotone complexity of some boolean functions. In *Soviet Math. Dokl.*, volume 31, pages 354–357, 1985.

[24] Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical notes of the Academy of Sciences of the USSR*, 41:333–338, 1987.

[25] Alexander A Razborov. Pseudorandom generators hard for k-dnf resolution and polynomial calculus resolution. *Annals of Mathematics*, pages 415–472, 2015.

[26] Alexander A. Razborov and Steven Rudich. Natural proofs. *Electron. Colloquium Comput. Complex.*, 1, 1994.

[27] Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *IN PROCEEDINGS OF THE 19TH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, STOC '87*, pages 77–82, 1987.

[28] Ryan Williams. Applying practice to theory, 2008.