Searching for circuit complexity lower bounds using SAT solvers

August Taylor

February 25, 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 [2] 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). [1]

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. [3] 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. [1]

A heuristic approach is to find minimal circuits for small instances of the functions. This can be used to prove new bounds [7] and may lead to theoretical insights on the structure of their optimal circuits generally [10]. 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 [6].

1.2 Previous work

Kamath et al [5] 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 [4]. Kojevnikov et al [6] 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 [8] 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.

*Mention valiant universal circuits?

*Should look for other things

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, testing them on a range of different functions.

We also present a new reduction *not sure what to say about this*

*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. [6]

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 $f : \{0,1\}^n \rightarrow \{0,1\}^m$, there exists a circuit of size N computing f.

2.2 Reduction to SAT

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

2.2.1 Existing reductions

The first is due to Kojevnikov et al [6] 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. The number of clauses generated is $O(N^3 \cdot 2^n)$ and the number of variables is $O(2^n \cdot N)$

The second is due to Razborov [9] and requires a circuit over the basis C_2: *not sure how much to describe this one but I definitely need to give the growth rates..*

Also am bothered that this is the only reduction that allows constants in the circuit - it wouldn't be hard to remove that aspect.

2.2.2 A new reduction

The third reduction is straightforward and also requires a circuit over the basis C_2.

It uses the following variables:

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

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

3. g_i^0, g_i^1 $(n \leq i < N)$ 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 $n \leq i < n + N$, $0 \leq j < i$ 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. A clause must be added for each function in the basis. For example, the clause for when the ith gate is an $\wedge$-gate:

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

   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 leads to $O(2^n \cdot N^3)$ 7-clauses.

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

3. For all (i, j) with $n \leq i < n + N$, $0 \leq j < i$, 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 clause for when the ith gate is an $\wedge$-gate:

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

   This leads to $O(N^2)$ $O(N)$-clauses.

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

2.3 Extension axioms

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 mean.

We chose to investigate the existence of circuits up to 9 gates, as these were reliably solved within less than 12 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 [4].

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

3.2 Comparison of SAT solvers and reductions

We used three SAT solvers, each based on DPLL with a variety of heuristics. (give a brief introduction on minisat, picosat, maplesat)

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 PicoSAT, and the naive and Kulikov reductions performing better than the Razborov reduction. **however I should take out the constants and re-run the experiments to be sure it's not that..

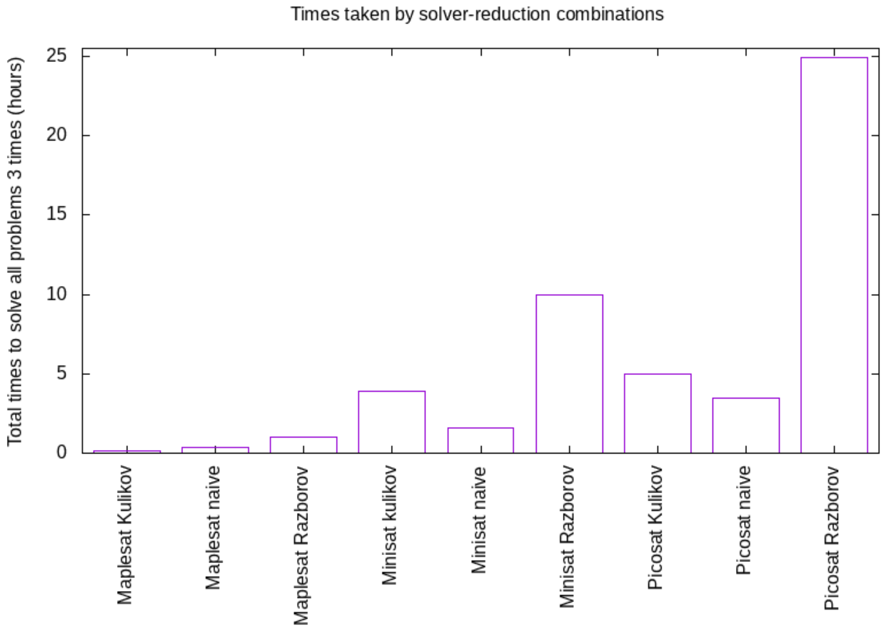

Figure 1: Comparison of times taken by SAT solvers on different reductions

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

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 [1], the usual counting argument breaks down for such small values of n (ask for help - why?). However, using the SAT solver we can check the number of hard circuits directly.

image

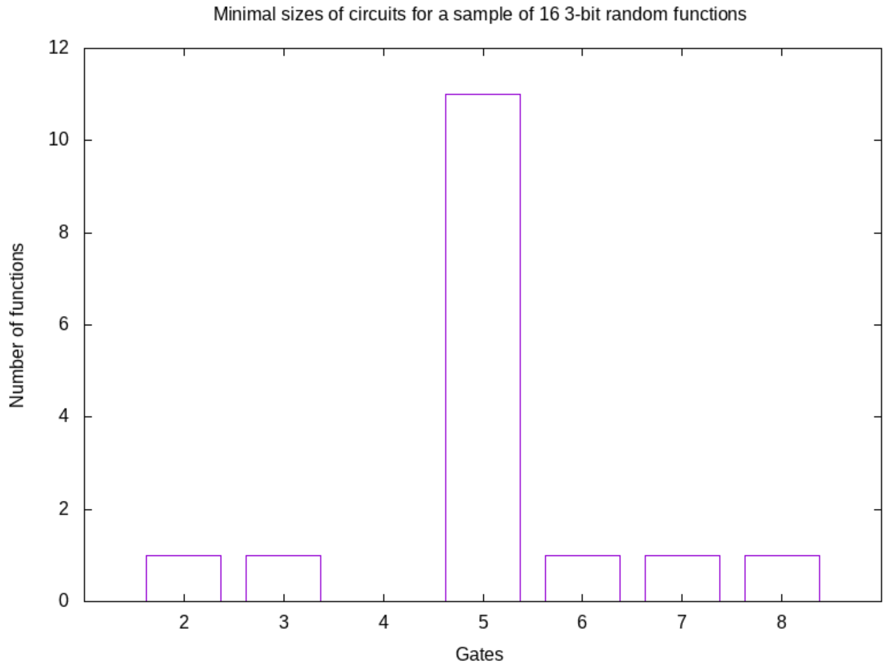

Figure 2: Sizes of optimal circuits for 3-bit functions with randomly chosen truth tables

3.4 Extension axioms

4 Discussion

4.1 Reflection

4.2 Conclusions

4.3 Further directions

References

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

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

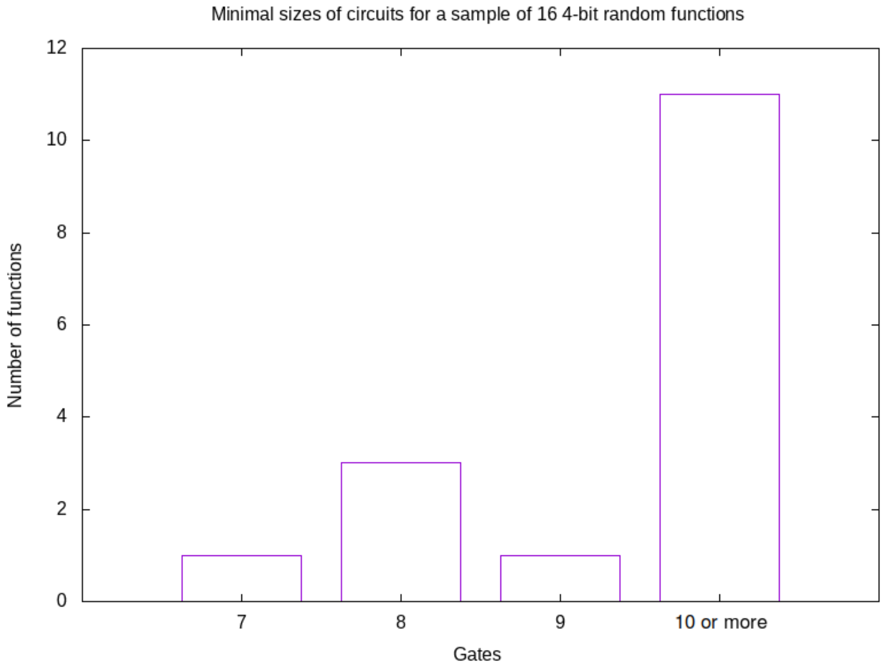

Figure 3: Sizes of optimal circuits for 4-bit functions with randomly chosen truth tables

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

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

[5] 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.

[6] 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.

[7] 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.

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

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

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