National Institute of Standards and Technology
U.S. Department of Commerce

Optimizing Implementations of Boolean Functions

Meltem Sönmez Turan

National Institute of Standards and Technology

Presented at BFA2023 – September 2023

NIST Circuit Complexity Project

Goal:

- improve the understanding of the circuit complexity of Boolean functions and vectorial Boolean functions;
- develop new techniques for constructing better circuits for use by academia and industry.

Example circuits: [1]

Circuit	Gate count					Depth	
	All	`AND`	`XOR`	`XNOR`	NOT	Total	`AND`
AES S-Box	113	32	77	4	0	27	6
AES-128(k,m)	28 600	6400	21 356	844	0	326	60
SHA-256(m)	115 882	22 385	89 248	3894	355	5403	1604

[1] Project webpage: https://csrc.nist.gov/Projects/circuit-complexity

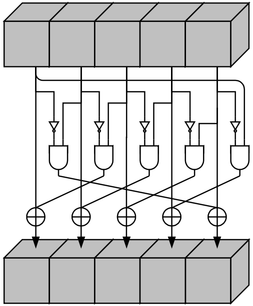
Overview

- Boolean circuits
- Optimizing linear circuits & Paar's heuristic
- Extending Paar's heuristic

Boolean Circuits

A *Boolean circuit* with n inputs and m outputs is a **directed acyclic graph** (DAG), where

- the inputs and the gates are *nodes*,
- the edges correspond to Boolean-valued *wires*,
- *fanin*/*fanout* of a node is the number of wires going in/out the node,
- the nodes with fanin zero are called *input nodes*,
- the nodes with fanout zero are called *output nodes*.



Circuit for Keccak s-box
https://keccak.team/figures.html

Optimizing Boolean Circuits

Problem: Given a set of Boolean gates (e.g., `AND`, `NAND`, `XOR`, `NOR`), construct a circuit that computes a Boolean function that is optimal w.r.t. a target metric.

Target metric depends on the application.

- *Number of gates:* for *lightweight cryptography applications* running on constrained devices.
- *Number of nonlinear gates:* for *secure multi-party computation*, *zero-knowledge proofs* and *side channel protection*.
- *AND-depth:* for homomorphic encryption schemes.
- etc.

Linear vs. Nonlinear Layers

- Linear layers
  - provides **diffusion**
  - e.g., bit permutations, multiplication with a binary matrix
  - implementations by `XOR`,`NOT` gates
- Nonlinear layers
  - provides **confusion**
  - e.g., s-boxes
  - implementations by `AND`,`NAND`,`XOR`,`NOT` gates

Constructing efficient circuits for these layers are challenging, even for the linear ones.

Linear Optimization

Linear layers can be represented using a an $m \times n$ binary matrix M, applied to n input variables $(x_1, \ldots, x_n)$ to calculate m output variables $(y_1, \ldots, y_m)$.

The linear layer

$$
\begin{aligned}
x_0 + x_1 + x_2 &= y_0 \\
x_1 + x_3 + x_4 &= y_1 \\
x_0 + x_2 + x_3 + x_4 &= y_2 \\
x_1 + x_2 + x_3 &= y_3 \\
x_0 + x_1 + x_3 &= y_4 \\
x_1 + x_2 + x_3 + x_4 &= y_5
\end{aligned}
$$

Matrix representation

$$
\begin{pmatrix}
1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4
\end{pmatrix}
=
\begin{pmatrix}
y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5
\end{pmatrix}
$$

For a given binary $m \times n$ matrix M, the goal is to minimize the number of `XOR` operations.

Three Metrics: `d-XOR`, `s-XOR` and `g-XOR`

- **Direct `XOR` (`d-XOR`)**
  - Implements each row individually, corresponds to $(weight(M) - m)$ XORs.
- **Sequential `XOR` (`s-XOR`)**
  - Counts the number of `XOR` operations of the form $x_i = x_i \oplus x_j$, that updates the value of input x_i
  - Relevant for quantum implementations
  - Known techniques (e.g., Gauss-Jordan elimination)
- **General `XOR` (`g-XOR`)**
  - Corresponds to the number of operations of the form $x_i = x_j \oplus x_k$
  - *The Shortest Linear Program (SLP) problem:* Minimizing the number of XORs (i.e., determining `g-XOR`) to compute Mx is known to be NP-hard. (Boyar et al., 2013)

Paar's Heuristic (1997)

Main idea:

- Determines the frequency for each possible pairs of input variable x_i, x_j $(i \neq j)$ that are XORed together in m linear functions
- Compute the pair with highest frequency and place it to the matrix as a new variable
- Repeat until all outputs have been computed

Two options in a tie:

- Choose the first pair in lexicographical order
- Exhaust all equally frequent options

Example: Paar's Heuristic (1)

Matrix representation

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}$$

Pair	Frequency	Pair	Frequency
(x_0, x_1)	2	(x_1, x_3)	**4**
(x_0, x_2)	2	(x_1, x_4)	2
(x_0, x_3)	2	(x_2, x_3)	3
(x_0, x_4)	1	(x_2, x_4)	2
(x_1, x_2)	3	(x_3, x_4)	3

The first selected pair is (x_1, x_3) with frequency 4. So, the first step of the implementation is $t_0 = x_1 \oplus x_3$.

Example: Paar's Heuristic (2)

Updated matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & \color{red}{0} \\ 0 & \color{red}{0} & 0 & \color{red}{0} & 1 & \color{red}{1} \\ 1 & 0 & 1 & 1 & 1 & \color{red}{0} \\ 0 & \color{red}{0} & 1 & \color{red}{0} & 0 & \color{red}{1} \\ 1 & \color{red}{0} & 0 & \color{red}{0} & 0 & \color{red}{1} \\ 0 & \color{red}{0} & 1 & \color{red}{0} & 1 & \color{red}{1} \end{pmatrix}$$

Updated frequency table

Pair	Frequency	Pair	Frequency
(x_0, x_1)	1	(x_1, t_0)	0
(x_0, x_2)	2	(x_2, x_3)	1
(x_0, x_3)	1	(x_2, x_4)	2
(x_0, x_4)	1	(x_2, t_0)	2
(x_0, t_0)	1	(x_3, x_4)	1
(x_1, x_2)	1	(x_3, t_0)	0
(x_1, x_3)	0	(x_4, t_0)	2
(x_1, x_4)	0	-	-

Implementation:

$$t_0 = x_1 \oplus x_3 \qquad t_3 = x_1 \oplus t_1 \qquad t_6 = x_2 \oplus t_0$$

$$t_1 = x_0 \oplus x_2 \qquad t_4 = x_3 \oplus x_4 \qquad t_7 = x_0 \oplus t_0$$

$$t_2 = x_4 \oplus t_0 \qquad t_5 = t_1 \oplus t_4 \qquad t_8 = x_2 \oplus t_2$$

The output $(y_0, y_1, y_2, y_3, y_4, y_5)$ is obtained as $(t_3, t_2, t_5, t_6, t_7, t_8)$.

Cancellation-free

Cancellations in circuits happen when the inputs to an `XOR` gate are of the form $(x_1 \oplus x_3, x_2 \oplus x_3)$. The `XOR` gate computes $x_1 \oplus x_2$, and cancels x_3.

Paar's heuristic is **cancellation-free**, which leads to generating sub-optimal circuits (Boyar et al., 2019).

New heuristics with cancellation property, such as Maximov & Ekdahl, 2019, Banik et al. 2019, Xiang et al, 2020.

Observation

- Due to cancellation-free property, a modification of Paar's algorithm can be applied to nonlinear Boolean functions.

Paar's heuristic for Nonlinear Boolean Functions

Represent n-variable Boolean function with m monomials using a $m \times n$ binary matrix.

Example. $f = x_1 + x_2.x_3 + x_0x_1x_3x_4$. Matrix representation is $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix}$

With this representation, it is possible to apply Paar's heuristic. Note that `XOR` operations now corresponds to `AND`.

Not promising approach, as it implements each monomial independently.

Main idea: Decompose Boolean function into homogeneous Boolean functions, and exploit affine equivalence relations to find low-weight matrix representations.

Proposal

1. Decompose f into d homogeneous Boolean functions,

$$f = a + f_1 \oplus f_2 \oplus \ldots \oplus f_d,$$

where f_i is the sum of monomials of f with degree i, and a is the constant term.

Example. $f = x_1 + x_2.x_3 + x_2.x_4 + x_2x_5 + x_1.x_2.x_4 + x_1.x_2.x_5 + 1$ The decomposition is

$$\begin{aligned} a =& 1 \\ f_1 =& x_1 \\ f_2 =& x_2.x_3 + x_2.x_4 + x_2x_5 \\ f_3 =& x_1.x_2.x_4 + x_1.x_2.x_5 \end{aligned}$$

Proposal

2. Apply affine transformations to the highest-degree homogeneous function, (i.e., f_d) to reduce the $\#$ of monomials. If f'_d includes monomials with degree smaller than d, those monomials are added to the corresponding f_i depending on their degree.

$$
\begin{aligned}
a =& 1 \\
f_1 =& x_1 \rightarrow f'_1 = x_1 \\
f_2 =& x_2.x_3 + x_2.x_4 + x_2 x_5 \rightarrow f'_2 = x_2.x_3 \\
f_3 =& x_1.x_2.x_4 + x_1.x_2.x_5 \rightarrow f'_3 = x_1.x_2.x_3
\end{aligned}
$$

Proposal

3. Apply modified Paar's heuristic to find an implementation for the degree d terms of f'_d. (Note that in modified Paar's heuristic each iteration corresponds to modulo 2 multiplication, instead of modulo 2 addition.) Apply the inverse affine transformation to the circuit to construct an implementation for the degree d monomials of f.
4. Repeat the procedure to find an implementation for f'_{d-1} where f'_{d-1} is the XOR of f_d and the new degree $d-1$ monomials generated during Step 2.
5. Repeat until implementations for all homogoeneous function is generated and combine the sub-circuits.

Experiments and Notes

- Most time consuming phase is finding the *right* affine equivalence class. If a class representative with low degree is available, decomposing functions into homogeneous functions, and reducing the number of monomials of same degree can be done much more efficiently.

  Example. Let $n = 6$. There are 150357 affine equivalance classes.
  - Degree=6, # classes = 74596 $\rightarrow$ # monomial = 1
  - Degree=5, # classes =73262 $\rightarrow$ # monomial = 1
  - Degree=4, # classes =2465 $\rightarrow$ # monomial ≤ 3
  - Degree=3, # classes =30 $\rightarrow$ # monomial ≤ 5
  - Degree=2, # classes =3 $\rightarrow$ # monomial ≤ 3

- We observe that the technique achieves optimal implementations (in terms of nonlinear gates) for some of the classes for small $n \leq 6$, where it is possible to compare with the optimal values.

Conclusion

- Proposed a modification to Paar's algorithm to apply to nonlinear Boolean functions (possible due to the cancellation-free property)
- Technique is currently more efficient when a low-weight representative from the equivalence class of the target function is available ($n \leq 6$).
- For larger n, our goal is to achieve a generic bound for Boolean function complexity (in term of AND gates), which is better than generic bounds.

Thanks! Questions?

- Contact:
  `meltem.turan@nist.gov`
  `circuit_complexity@nist.gov`
- GitHub: https://github.com/usnistgov/Circuits/
- NIST Circuit Complexity Project Webpage:
  https://csrc.nist.gov/Projects/Circuit-Complexity

References

Pa97 Paar, Optimized Arithmetic for Reed-Solomon Encoders. In 1997 IEEE International Symposium on Information Theory, 1997.

BFP19 J. Boyar, MG. Find, R. Peralta, *Small Low-Depth Circuits for Cryptographic Applications* Cryptogr Commun. 2019

BPP00 J. Boyar, R. Peralta, and D. Pochuev, *On the multiplicative complexity of Boolean functions over the basis* $(\wedge, \oplus, 1)$ Theoretical Computer Science, vol. 235, no. 1, pp. 43 – 57, 2000.

XZLB20 Z. Xiang, X. Zeng, D. Lin, Z. Bao, and S. Zhang. Optimizing implementations of linear layers. IACR Transactions on Symmetric Cryptology, 2020(2):120–145, Jul. 2020.

BFI19 S. Banik, Y. Funabiki, and T. Isobe. *More results on shortest linear programs* Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan

FTT17 M. G. Find, D. Smith-Tone, M. Sönmez Turan, *The Number of Boolean Functions with Multiplicative Complexity 2* International Journal of Information and Coding Theory, 2017.

CTP19 Ç. Çalık, M. Sönmez Turan, R. Peralta, *Boolean Functions with Multiplicative Complexity 3 and 4* Cryptography and Communications 2019.

STP21 M. Sönmez Turan and R. Peralta. *On the Multiplicative Complexity of Cubic Boolean Functions* IACR Cryptol. ePrint Arch. 2021: 1041 (2021)