Revealing the canalizing structure of Boolean functions: Algorithms and applications [*]

Elena Dimitrova [a], Brandilyn Stigler [b], Claus Kadelka [c], David Murrugarra [d]

[a] *Department of Mathematics, California Polytechnic State University, San Luis Obispo, CA, USA*

[b] *Department of Mathematics, Southern Methodist University, Dallas, TX, USA*

[c] *Department of Mathematics, Iowa State University, Ames, IA, USA*

[d] *Department of Mathematics, University of Kentucky, Lexington, KY, USA*

Abstract

Boolean functions can be represented in many ways including logical forms, truth tables, and polynomials. Additionally, Boolean functions have different canonical representations such as minimal disjunctive normal forms. Another canonical representation is based on the polynomial representation of Boolean functions and the biologically motivated concept of canalization: any Boolean function can be written as a nested product of canalizing layers and a polynomial that contains the variables that are never canalizing. In this paper we study the problem of identifying the canalizing layers of a Boolean function. First, we show that the problem of finding the canalizing layers is **NP**-hard. Second, we present several algorithms for finding the canalizing layers, discuss their complexities, and compare their performances. Third, we exhibit how the canalizing layers format can be used to find a disjunctive normal form for any nested canalizing function. Another application deals with the reverse engineering of Boolean networks with a prescribed layering format. Implementations of the developed algorithms in Python and in the computer algebra system Macaulay2 are available at https://github.com/ckadelka/BooleanCanalization.

Key words: Boolean functions; Canalizing layers; **NP**-hard; Disjunctive normal forms; Reverse engineering.

1 Introduction

First introduced in the 1940s, canalization refers to a concept from evolutionary and developmental biology used to describe the ability of organisms to produce and maintain the same phenotype in the face of widespread variability in their genotypes [28,27]. S. Kauffman introduced Boolean canalizing functions and studied their role in gene regulatory network models [14]. Over the years, more evidence emerged that canalization is ubiquitous in gene regulation [6,1]: most rules used in over 130 expert-curated published gene regulatory network models are even so-called nested canalizing [9]. These findings intensified the study of Boolean canalization and its impact on the dynamics and controllability of gene regulatory network models [8,20]. Boolean networks governed by canalizing functions are typically more stable than networks governed by random functions [15,11]. That is, they have fewer and shorter attractors and are more robust to perturbations. The amount of canalization exhibited by a Boolean function can be quantified using concepts such as the canalizing depth [16,7], the canalizing strength [10], or the logical reduncancy [19,4]. As a rule of thumb, the higher the amount and prevalence of canalization the more stable are the dynamics [13].

A canalizing function possesses at least one "canalizing" variable such that, if this variable takes on its "canalizing" value, then the output value is already determined, regardless of the values of the remaining variables. If the canalizing variable does not take on its canalizing value, but there is a second variable with the canalization property, the function is 2-canalizing. If k variables follow this pattern, the function is k-canalizing. If all variables are canalizing in this iterative way, the function is "nested canalizing"; the Boolean AND function is such an ex-

[*] Corresponding author David Murrugarra. Tel. 859 257 4734.

Email addresses: edimitro@calpoly.edu (Elena Dimitrova), bstigler@mail.smu.edu (Brandilyn Stigler), ckadelka@iastate.edu (Claus Kadelka), murrugarra@uky.edu (David Murrugarra).

ample. Boolean nested canalizing functions have been shown to be equivalent to unate cascade functions [8], a class of functions that have been extensively studied by computer scientists [18,26]. The concept of canalization has also been extended to the multistate case where variables can take on three or more input values and the dynamics of networks governed by multistate canalizing functions have been studied [22,23,12,21].

The concept of canalizing layers was first introduced for nested canalizing functions [17]. Later, a full stratification of the variables of any Boolean function into layers was provided [7]. This layer structure provides a unique polynomial form (up to permutations within the layers) of representing any Boolean function and contains various important information about the dynamical properties of the function. For instance, the canalizing layers of a nested canalizing function are related to the function's symmetry groups [25]; all variables in the same layer are contained within at most two symmetry groups. Further, the layer structure has been used for quantifying the total effect of network perturbations and control [20,21], as well as for calculating exact Derrida values, a stability measure of Boolean networks, for networks based on nested canalizing functions [11].

In this paper we present multiple algorithms for finding the canalizing layers of a Boolean function. We also provide implementations of the algorithms in Python and in the computer algebra system Macaulay2 [5]. Although our implementations can handle finding the layer structure for most functions in published models [9], the problem is **NP**-hard, as we prove in Section 4.

The canalizing layer structure can be used to obtain useful information about the Boolean functions in a model. In this paper, we focus on two practical applications. First, we show how to obtain a disjunctive normal form for a nested canalizing function from the layers. Second, we provide an application for how to reverse engineer a family of Boolean functions with a prescribed layering format.

This paper is structured in the following way. We provide the theoretical background of Boolean canalization and the layer structure in Section 2 and show that the problem of finding the canalizing layers is **NP**-hard in Section 4. Then in Section 5 we present multiple algorithms along with their performance and conclude with applications. We describe how to obtain disjunctive normal forms from canalization layers of Boolean functions in Section 6.1, and show how to use layers for reverse engineering in Section 7. In Section 8 we briefly summarize the main findings and their significance.

2 Background

In this section, we review some concepts and definitions and introduce the concept of *canalization* as well as *layers of canalization*. Throughout the manuscript, we consider functions over a two-element field $\mathbb{F}_2 = \{0, 1\}$ (i.e., Boolean functions). We frequently use the fact that any Boolean function $f \colon \mathbb{F}_2 \times \cdots \times \mathbb{F}_2 \to \mathbb{F}_2$ can be represented by a polynomial function with coefficients in $\mathbb{F}_2$, which enables the use of tools from algebraic geometry.

Definition 1 *Given a Boolean function* $f(x_1, \ldots, x_n)$, *variable* x_i *is* essential *if the two functions* $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ *and* $f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ *are not equivalent. Otherwise,* x_i *is* non-essential.

Definition 2 *A Boolean function* $f(x_1, \ldots, x_n)$ *is* canalizing *if there exists a variable* x_i, *values* $a, b \in \{0, 1\}$, *and a Boolean function* $g(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \not\equiv b$ *such that*

$$f(x_1, \ldots, x_n) = \begin{cases} b, & \text{if } x_i = a \\ g(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n), & \text{if } x_i \neq a. \end{cases}$$

In this case, x_i *is called a* canalizing variable, *a its* canalizing input, *and b the* canalized output.

Definition 3 *[7] A Boolean function* $f(x_1, \ldots, x_n)$ *is* k-canalizing, *where* $1 \leq k \leq n$, *with respect to the permutation* $\sigma \in \mathcal{S}_n$, *inputs* $a_1, \ldots, a_k$, *and outputs* $b_1, \ldots, b_k$ *if*

$$f(x_1, \ldots, x_n) =$$
$$\begin{cases} b_1 & x_{\sigma(1)} = a_1, \\ b_2 & x_{\sigma(1)} \neq a_1, x_{\sigma(2)} = a_2, \\ b_3 & x_{\sigma(1)} \neq a_1, x_{\sigma(2)} \neq a_2, x_{\sigma(3)} = a_3, \\ \vdots & \vdots \\ b_k & x_{\sigma(1)} \neq a_1, \ldots, x_{\sigma(k-1)} \neq a_{k-1}, x_{\sigma(k)} = a_k, \\ f_C \not\equiv b_k & x_{\sigma(1)} \neq a_1, \ldots, x_{\sigma(k-1)} \neq a_{k-1}, x_{\sigma(k)} \neq a_k, \end{cases}$$

where $f_C = f_C(x_{\sigma(k+1)}, \ldots, x_{\sigma(n)})$ *is the* core function, *a Boolean function on* $n - k$ *variables. When* f_C *is not canalizing, then the integer* k *is the* canalizing depth *of* f *[16].*

An n-canalizing function is also called a *nested canalizing function* (NCF), and by definition all Boolean functions are 0-canalizing.

Authors He and Macauley provided the following powerful stratification theorem.

Theorem 4 ([7]) *Every Boolean function* $f(x_1, \ldots, x_n) \not\equiv$

0 *can be uniquely written as*

$$f(x_1, \ldots, x_n) = \\ M_1(M_2(\cdots(M_{r-1}(M_r p_C + 1) + 1)\cdots) + 1) + q, \quad (1)$$

where each $M_i = \prod_{j=1}^{k_i}(x_{i_j} + a_{i_j})$ *is a nonconstant extended monomial,* p_C *is the* core polynomial *of* f*, and* $k = \sum_{i=1}^{r} k_i$ *is the canalizing depth. Each* x_i *appears in exactly one of* $\{M_1, \ldots, M_r, p_C\}$*, and the only restrictions are the following "exceptional cases":*

(1) If $p_C \equiv 1$ *and* $r \neq 1$*, then* $k_r \geq 2$*;*
(2) If $p_C \equiv 1$ *and* $r = 1$ *and* $k_1 = 1$*, then* $q = 0$*.*

When f *is not canalizing (i.e., when* $k = 0$*), we simply have* $p_C = f$*.*

Remark 5 *Note the following properties of canalization.*

(1) Theorem 4 shows that any Boolean function has a unique extended monomial form, in which the variables are partitioned into different layers based on their dominance. Any variable that is canalizing (independent of the values of other variables) is in the first layer. Any variable that "becomes" canalizing when excluding all variables from the first layer is in the second layer, etc. Variables in any layer will be referred to as conditionally canalizing. *All remaining variables that never become canalizing are part of the core polynomial. The number of variables that eventually become canalizing, i.e., the number of conditionally canalizing variables, is the canalizing depth of the function. NCFs are exactly those functions where all variables are conditionally canalizing.*

(2) While variables in the same layer may have different canalizing input values, they all share the same canalized output value, i.e., they all canalize a function to the same output. On the other hand, the outputs of two consecutive layers are distinct. Therefore, the number of layers of a k-canalizing function expressed as in Definition 3 is simply one plus the number of changes in the vector of canalized outputs, $(b_1, b_2, \ldots, b_k)$*.*

Example 6 *The function*
$f = (x_1 + 1)x_2[(x_3 + 1)x_4(x_5x_6 + x_7 + 1) + 1]$ *has the unique standard extended monomial form* $]f = M_1(M_2 p_C + 1)$*, where* $M_1 = (x_1 + 1)x_2$*,* $M_2 = (x_3 + 1)x_4$*, and* $p_C = x_5x_6 + x_7 + 1$*. The variables* x_1 *and* x_2 *are canalizing and are in the first (outermost) layer, while*

x_3 *and* x_4 *are conditionally canalizing and are in the second (and consequently last) layer. In brief,* f *has 2 layers and canalizing depth 4.*

We modify Equation 1 to present the *layer structure* [1] of a Boolean function; that is complete information about canalizing and conditionally canalizing variables, their inputs and outputs, and the core function.

Definition 7 *Let* $f(x_1, \ldots, x_n)$ *be a Boolean function with r layers. The* layer structure *of* f *is an ordered pair (`layers`, `core function`) where*

- `layers` *is an ordered set of the layers (as a list) and corresponding canalized outputs (as an integer) of the form*
  {(layer 1, output 1), ..., (layer r, output r)},
  where each layer i*,* $1 \leq i \leq r$*, is an unordered set of conditionally canalizing variables and their canalizing inputs of the form*
  {(variable 1, input 1), ..., (variable k_i*, input* k_i*)}.*
- `core function` *is the function* f_C *from Definition 3. The core function and core polynomial from Theorem 4 are intrinsically related,* $f_C = p_C$ *or* $f_C = p_C + 1$ *[7].*

Example 8 *Consider the function* f *in Example 6. The layer structure of* f *is*

$$\left(\left\{ \Big(\{(x_1, 1), (x_2, 0)\}, 0 \Big), \Big(\{(x_3, 1), (x_4, 0)\}, 1 \Big) \right\}, \\ x_5x_6 + x_7 + 1 \right).$$

Remark 9 *Any nonzero Boolean function has a unique layer structure by Theorem 4. Note that functions with a single essential variable will have two canalizing input and canalized output values, yet they still have a unique layer structure, forced by "exceptional case" 2 in Theorem 4. For example, the function* x_1 *has canalizing input value* 0 *with corresponding canalized output value* 0 *but also canalizing input value* 1 *with corresponding canalized output value* 1*. However, its unique layer structure is*

$$\left(\left\{ \Big(\{(x_1, 1)\}, 1 \Big) \right\}, 1 \right).$$

Likewise the unique layer structure for $f = x_1 + 1$ *is*

$$\left(\left\{ \Big(\{(x_1, 0)\}, 1 \Big) \right\}, 1 \right)$$

---

[1] In [11], the authors referred to the summary vector $(k_1, \ldots, k_r)$, where r is the number of layers and k_i is the number of variables in the i-th layer, as the *layer structure* of f.

since f is written as $((x_1 + a_1) \cdot p_C + 1) + q$. Having one layer requires $p_C = 1$. By "exceptional case" 2, we have $q = 0$. This now forces a canalizing input $a_1 = 0$ with corresponding canalized output of 1.

3 Enumeration of Boolean functions with a given number of layers

In this section, we count the number of Boolean functions on n variables that contain a certain number of layers. This extends the results in [7, Section 5]. We start with a remark that summarizes the known results, which are all direct consequences of Theorem 4.

Remark 10 *The following has been proved in [7].*

(1) The number C_n of canalizing Boolean functions on $n \geq 0$ variables is

$$C_n = 2((-1)^n - n - 1) + \sum_{k=1}^{n} (-1)^{k+1} \binom{n}{k} 2^{k+1} 2^{2^{n-k}}.$$

(2) The number $B^\star(n, 0)$ of Boolean non-constant core polynomials on $n \geq 0$ variables (i.e., the number of Boolean non-constant, non-canalizing functions) is

$$B^\star(n, 0) = 2^{2^n} - C_n - 2.$$

(3) The number $B(n, n)$ of Boolean nested canalizing functions on $n \geq 2$ variables is

$$B(n, n) = 2^{n+1} \sum_{r=1}^{n-1} \sum_{\substack{k_1 + \cdots k_r = n \\ k_i \geq 1, k_r \geq 2}} \binom{n}{k_1, \ldots, k_r},$$

where $\binom{n}{k_1, \ldots, k_r} = \frac{n!}{k_1! k_2! \cdots k_r!}$. Further, $B(1, 1) = 2$.

(4) The number $B(n, k)$ of Boolean functions on n variables with exact canalizing depth k, for $1 \leq k \leq n$, is

$$B(n, k) = \binom{n}{k} \Big[B(k, k) + B^\star(n - k, 0) \cdot 2^{k+1} \sum_{r=1}^{k} \sum_{\substack{k_1 + \cdots k_r = k \\ k_i \geq 1}} \binom{n}{k_1, \ldots, k_r} \Big].$$

Corollary 11 *For $n \geq 2$ and $1 \leq r \leq n - 1$, the number $B(n, n, r)$ of Boolean nested canalizing functions on n variables with exactly r layers is*

$$B(n, n, r) = 2^{n+1} \sum_{\substack{k_1 + \cdots k_r = n \\ k_i \geq 1, k_r \geq 2}} \binom{n}{k_1, \ldots, k_r},$$

and we have $B(n, n) = \sum_{r=1}^{n-1} B(n, n, r)$. Further, $B(1, 1, 1) = 2$.

The number $B(n, k, r)$ of Boolean functions on n variables with exact canalizing depth k and exactly r layers, for $1 \leq r \leq k \leq n$, is

$$B(n, k, r) = \binom{n}{k} \Big[B(k, k, r) + B^\star(n - k, 0) \cdot 2^{k+1} \sum_{\substack{k_1 + \cdots k_r = k \\ k_i \geq 1}} \binom{n}{k_1, \ldots, k_r} \Big],$$

and we have $B(n, k) = \sum_{r=1}^{k} B(n, k, r)$. Note that if $n = k$ (i.e., for nested canalizing functions), we have $B^\star(n - k, 0) = B^\star(0, 0) = 0$ so that the provided formula does not contradict the $k_r \geq 2$ requirement.

Finally, the number of Boolean functions with exactly r layers, for $1 \leq r \leq n - 1$, is $\sum_{k=1}^{n} B(n, k, r)$.

PROOF. To count the number of Boolean functions on n variables that contain a fixed number of layers, we separate the summations in Remark 10 (3) and (4) based on r, the number of layers.

These formulas enable a simple exploration of the prevalence of canalizing functions. As known, the property of canalization, and in particular nested canalization, becomes increasingly rare for functions with more variables (Figure 3). Functions with $n - 1$ canalizing variables all possess an nth non-essential variable and are, just like functions with $n - 2$ canalizing variables, even rarer than nested canalizing functions (Figure 3A). When looking at the number of functions with a specific number of canalizing layers (instead of a specific number of canalizing variables), a monotonic relationship appears for $n \geq 3$: there exist fewer functions with r layers than $r - 1$ layers, $r = 1, \ldots, n - 1$.

4 Finding canalizing layers is NP-hard

In this section we show that the problem of determining the specific layer structure of a Boolean function is **NP**-hard, thus requiring efficient algorithms. To show that the problem of determining the canalizing layers is **NP**-hard we exploit the connection between the canalizing layers of a nested canalizing function and the function's symmetry groups given in [25], which says that all variables in the same layer are contained within at most two symmetry groups. Here we reproduce the relevant result from [25] to make our description self-contained.

Theorem 12 *(adapted version of Theorem 2.5 in [25]) Unless $\mathbf{P} = \mathbf{NP}$, there is no polynomial time algorithm for determining the symmetry level of a Boolean function f specified as a Boolean expression, even if f is a CNF expression.*
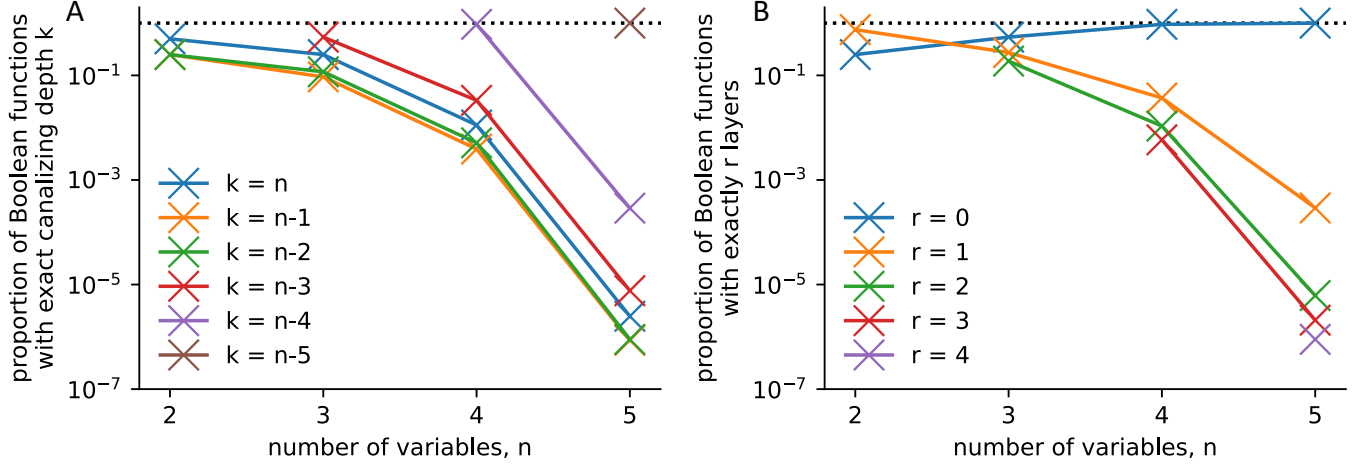
Fig. 1. Enumeration of Boolean functions. Using the formulas in Remark 10 and Corollary 11, this figure shows the proportion of Boolean functions with $n = 2 - 5$ variables, which have (A) a specific canalizing depth or (B) a specific number of layers.

This theorem was proved by showing that determining the symmetry level of a Boolean function is equivalent to solving the CNF Satisfiability Problem (SAT), which is known to be **NP**-hard (Cook-Levin Theorem). We will now show that a polynomial time algorithm (with respect to the number of variables) for the identification of layers and core function of a Boolean function would also yield the symmetry level in polynomial time.

Theorem 13 *Unless* **P** = **NP**, *there is no polynomial time algorithm for determining the standard extended monomial form (Theorem 4; i.e., the layers and the core function) of a Boolean function f specified as a Boolean expression, even if f is a CNF expression.*

The following proof was adapted from [25].

PROOF. [Proof of Theorem 13] Assume there is a polynomial time algorithm $\mathcal{A}$ (in the size of the inputs) that yields the canonical monomial form, as stated in Theorem 4, for any Boolean function. We will show that $\mathcal{A}$ can be used to efficiently solve (in polynomial time) the CNF Satisfiability problem (SAT) which is known to be **NP**-hard (Cook-Levin Theorem, see Section 2.6 of [3]).

Let g be a CNF formula representing an instance of SAT. Let $X = \{x_1, \ldots, x_n\}$ denote the set of Boolean variables used in g. Let $y_1 \notin \{x_1, \ldots, x_n\}$ be a new variable and h be the Boolean function with 1 layer: $h = y_1$. Now, we create another CNF formula f as $f = g \wedge h$. Note that f is a function of $n + 1$ variables, namely $x_1, \ldots, x_n, y_1$. Since g is a CNF formula, so is f.

Consider the following cases for the satisfiability of g. If g is not satisfiable, then f is also not satisfiable; that is, for all inputs, the value of f is 0. By Definition 2, constant

functions do not possess any canalizing variables. Thus the number of layers of f is 0. On the other hand, if g is satisfiable, this implies that $y_1 = 0$ canalizes f to 0, *i.e.*, y_1 is a canalizing variable of f. Therefore f has at least one layer.

Suppose we execute the polynomial time algorithm $\mathcal{A}$ on the function f defined above. If g is not satisfiable, then the number of layers of f is 0. Thus $\mathcal{A}$ will produce zero layers. If g is satisfiable, then the number of layers of f is at least 1; so, $\mathcal{A}$ will produce at least one layer. In other words, g is not satisfiable if and only if the number of layers produced by $\mathcal{A}$ is 0. Since $\mathcal{A}$ runs in polynomial time, we have an efficient algorithm for SAT, contradicting the assumption that **P** $\neq$ **NP**.

5 Algorithms for revealing the canalizing layer structure

In this section we present algorithms for finding the layers of canalization of a Boolean function in three different cases:

(1) The function is presented as a polynomial $f : \{0,1\}^n \rightarrow \{0,1\}$ (see Section 5.1).
(2) The function is given in truth table format, *i.e.*, as a vector (see Section 5.2).
(3) The function is nested canalizing with a known order of conditionally canalizing variables (see Section 5.3).

The output produced by Algorithms 1 and 2 follows the layer structure format in Definition 7.

5

5.1 Algorithm for Boolean functions represented as polynomials

The first algorithm, implemented in Macaulay2, takes as input *any* Boolean function and returns its canalizing layers. The algorithm is based on successive evaluations of the inputs of the function to check for canalizing inputs and subsequent removal of these from the search.

5.2 Algorithm for Boolean functions represented as truth tables

The second algorithm, implemented in Python, uses elementary linear algebra and recursion to compute the layer structure of any Boolean function $f(x_1, \ldots, x_n)$ in n variables. Consider a *binary truth table* represented as a $2^n \times (n+1)$-matrix over $\{0, 1\}$ where the *left-hand side* of dimension $2^n \times n$ corresponds to the inputs of a function $f : \{0, 1\}^n \to \{0, 1\}$ and the $(2^n \times 1)$-dimensional *right-hand side* corresponds to the outputs of f when evaluated on the inputs. Like most authors (e.g., [7]), we exclude constant functions from the set of canalizing functions (Definition 2). In Algorithm 2, we therefore first check if f is constant (row 3). In the following, assume f is not constant.

Let $\mathbf{T}_n = (\mathbf{t}_1, \ldots, \mathbf{t}_n) \in \{0, 1\}^{2^n \times n}$ be the left-hand side of a binary truth table. That is, $\mathbf{t}_i$ is a binary vector of length 2^n, which is 1 in the rows of $\mathbf{T}_n$ where $x_i = 1$, and 0 otherwise. Likewise, we will think of the right-hand side of a binary truth table as a vector f. With $\langle \cdot, \cdot \rangle$ denoting the dot product, we have the following for any non-constant Boolean function f:

- $x_i = 1$ canalizes f to the value 1 if and only if $\langle \mathbf{t}_i, f \rangle = 2^{n-1}$;
- $x_i = 0$ canalizes f to the value 1 if and only if $\langle \mathbf{1} - \mathbf{t}_i, f \rangle = 2^{n-1}$;
- $x_i = 1$ canalizes f to the value 0 if and only if $\langle \mathbf{t}_i, \mathbf{1} - f \rangle = 2^{n-1}$;
- $x_i = 0$ canalizes f to the value 0 if and only if $\langle \mathbf{1} - \mathbf{t}_i, \mathbf{1} - f \rangle = 2^{n-1}$.

With this we can define four sets

$$I_{1 \to 1}(f) := \left\{ i \in \{1, \ldots, n\} \;\middle|\; \langle \mathbf{t}_i, f \rangle = 2^{n-1} \right\},$$

$$I_{0 \to 1}(f) := \left\{ i \in \{1, \ldots, n\} \;\middle|\; \langle \mathbf{1} - \mathbf{t}_i, f \rangle = 2^{n-1} \right\},$$

$$I_{1 \to 0}(f) := \left\{ i \in \{1, \ldots, n\} \;\middle|\; \langle \mathbf{t}_i, \mathbf{1} - f \rangle = 2^{n-1} \right\},$$

$$I_{0 \to 0}(f) := \left\{ i \in \{1, \ldots, n\} \;\middle|\; \langle \mathbf{1} - \mathbf{t}_i, \mathbf{1} - f \rangle = 2^{n-1} \right\}.$$

Then $I_{a \to b}(f)$ contains all the indices of canalizing variables that canalize f to the canalized output value b if they take on the canalizing input value a, for $a, b \in \{0, 1\}$. For brevity, let $I_b(f) = I_{0 \to b}(f) \cup I_{1 \to b}(f)$. In Algorithm 2, we do not calculate these sets for constant functions, which are by Definition 2 not canalizing. Therefore we have for any non-constant f

- $|I_0(f) \cup I_1(f)|$ is the number of canalizing variables.
- $I_0(f) = I_1(f) = \emptyset \iff f$ is not canalizing.

By Remark 5, all canalizing variables canalize f to the same output value and by Remark 9 each canalizing variable in functions with more than one essential variable possesses a unique canalized output. Thus, for non-trivial functions - with more than one essential variable - we have

- $I_0(f) \neq \emptyset \implies I_1(f) = \emptyset$;
- $I_1(f) \neq \emptyset \implies I_0(f) = \emptyset$.

This allows us to use these sets to define an iterative process that finds all layers of a Boolean function, recording layer by layer the canalizing inputs, canalized outputs and conditionally canalizing variables (see Algorithm 2). Note that Boolean functions with a single essential variable have two choices for the canalized output value (Remark 9). Theorem 4 ("exceptional case" 2) ensures a unique layer structure for these functions by forcing a canalized output of 1. To agree with this default choice in Algorithm 2, we first check for canalizing variables in the set $I_1(f)$.

Example 14 *Let $n = 3$ and consider the left-hand side of the $(2^3 \times 3)$-truth table*

$$\mathbf{T}_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

Let $f(x_1, x_2, x_3) = x_1 \land (x_2 \lor x_3)$. As a vector (with variable order x_1, x_2, x_3), we can express f as $f = (0, 0, 0, 0, 0, 1, 1, 1)$. Then we have

$$f \cdot \mathbf{T}_3 = (3, 2, 2), \quad f \cdot (\mathbf{1} - \mathbf{T}_3) = (0, 1, 1),$$

where $\mathbf{1} - \mathbf{T}_3$ is the difference of the (8×3)-matrix of ones and $\mathbf{T}_3$ modulo 2. None of these entries equal $2^{3-1} = 4$; hence $I_1(f) = \emptyset$.

On the other hand,

$$(1 - f) \cdot \mathbf{T}_3 = (1, 2, 2), \quad (1 - f) \cdot (\mathbf{1} - \mathbf{T}_3) = (4, 3, 3).$$

Algorithm 1 Polynomial Form

Input: A Boolean function $f : \{0,1\}^n \to \{0,1\}$ as a polynomial.
Output: Layer structure of f.

1: **if** $\#support(f) == 0$ **then return** $(\{\}, f)$ $\triangleright$ f is constant
2: **if** $\#support(f) == 1$ **then return** $(\{(\{(x_1, 1)\}, f(1))\}, 1)$ $\triangleright$ Exceptional Cases:
                                                               $\triangleright$ $f = x_1$ or $f = x_1 + 1$

3: $Layers = \{\}$
4: Set $NewVar = support(f)$ $\triangleright$ get variables of f
5: **while** $NewVar \neq \emptyset$ **do**
6:     $NewLayer = \{\}$ and $NegNewLayer = \{\}$
7:     **for all** $x_i \in NewVar$ **do**
8:         **for all** $a \in \{0,1\}$ **do**
9:             $\hat{X} = (x_1, \ldots, x_{i-1}, x_i = a, x_{i+1}, \ldots, x_n)$ $\triangleright$ substitute $x_i = a$
10:            $g = f(\hat{X})$ $\triangleright$ evaluate at $x_i = a$
11:         **if** $support(g) == \emptyset$ **then**
12:            Append (x_i, a) to $NewLayer$ $\triangleright$ store cond. canalizing variable and input
13:            Append $(x_i, a+1)$ to $NegNewLayer$ $\triangleright$ store noncanalizing input
14:     Append $(NewLayer, f(NewLayer))$ to $Layers$ $\triangleright$ evaluate f on canalizing inputs
                                                         $\triangleright$ of variables in $NewLayer$;
                                                         $\triangleright$ append $NewLayer$ and
                                                         $\triangleright$ canalized output to $Layers$

15:     Set $NewVar = NewVar \backslash \{\text{variables in } NewLayer\}$
16:     Set $f = f(NegNewLayer)$ $\triangleright$ evaluate f on noncanalizing inputs
                                                         $\triangleright$ of variables in $NegNewLayer$

17:     **if** $NewLayer == \emptyset$ **then**
18:         Break;
                                                         $\triangleright$ end of while loop
19: **return** $Layers, f(NewVar)$ $\triangleright$ return layers and core function.

Thus $I_0(f) = \{1\}$ and x_1 is therefore the only canalizing variable of f with $x_1 = 0$ canalizing f to 0. Removing the first layer $\{x_1\}$ from f yields the subfunction $g = f(1, x_2, x_3) = x_2 \vee x_3$ or $(0, 1, 1, 1)$ as a vector.

Now we repeat the process and find that $I_0(g) = \emptyset$ since $(1-g) \cdot \mathbf{T}_2 = (0,0)$ and $(1-g) \cdot (\mathbf{1} - \mathbf{T}_2) = (1,1)$. Likewise $I_1(g) = \{2,3\}$ since $g \cdot \mathbf{T}_2 = (2,2)$ and $g \cdot (\mathbf{1} - \mathbf{T}_2) = (1,1)$. Therefore x_2 and x_3 are canalizing variables of g and thus conditionally canalizing variables of f. Thus $\{x_2, x_3\}$ forms the second layer of f with canalized output 1. The process is completed because the new subfunction $h = g(1,1) = 1$ contains no more variables. Note that for non-NCFs, the process completes when no new conditionally canalizing variables are found, i.e., when both I_0 and I_1 are empty.

5.3 Algorithm for nested canalizing functions

The authors of [8] developed a parametrization of NCFs corresponding to points in the affine space $\mathbb{F}_2^{2^n}$ that satisfy a certain collection of polynomial equations. They observed that since the terms of a Boolean polynomial consist of square-free monomials, one can uniquely index monomials by the subsets of $[n] = \{1, \ldots, n\}$ corresponding to the variables appearing in the monomial. Thus the set of all Boolean polynomials can be expressed

as

$$\left\{ \sum_{S \subseteq [n]} c_s \prod_{i \in S} x_i \;\middle|\; c_s \in \mathbb{F}_2 \right\}. \tag{2}$$

We use this parametrization, and in particular Corollary 3.6 in [8], to suggest a linear time algorithm for finding the canalizing layers of a given NCF. The algorithm is based on the following proposition, which is a straightforward consequence of Corollary 3.6 in [8].

Proposition 15 *Let f be a Boolean polynomial written in the form of (2). If*

$$c_{\{1,\ldots,j\}} = c_{\{1,\ldots,j,j+1\}} c_{[n] \backslash \{1,\ldots,j,j+1\}} \text{ for all } j = 1, \ldots, n-1,$$

then x_j and x_{j+1} are in the same layer.

Therefore if the canalizing order of the n variables of an NCF is known, one needs to check at most $n-1$ equalities; that is, the algorithm suggested by Proposition 5.3 will take linear time in the number of variables.

5.4 Time complexity of the algorithms

As an indication for the time complexity, we recorded the average run time of both algorithms for random Boolean functions in 4 to 16 variables. We tested two extremes:

Algorithm 2 Truth Table Form

Input: A Boolean function $f : \{0,1\}^n \to \{0,1\}$ as a vector of length 2^n.
Output: Layer structure of f.

 1: $Layers = \{\}$ $\triangleright$ stores all layers
 2: Set $n = \log_2(\text{length } f)$ $\triangleright$ number of variables of f; n and f will change
 3: **if** $support(f) == \emptyset$ **then** $\triangleright$ f is constant
 4:    **return** $(Layers, f)$ $\triangleright$ constant functions are not canalizing - Def 2
 5: Initialize $\mathbf{T}_n$ $\triangleright$ left-hand side of a truth table
 6: Set $v = (1, \ldots, 1)$ $\triangleright$ vector of 2^n ones
 7: Set $M = [v', \ldots, v']$ $\triangleright$ matrix of $2^n \times n$ ones; $v' = $ transpose of v
 8: $NewLayer = \{\}$ $\triangleright$ stores the current layer
 9: $NegNewLayer = \{\}$ $\triangleright$ stores noncanalizing input of current layer
10: Compute $(v - f) \cdot \mathbf{T}_n$ and $(v - f) \cdot (M - \mathbf{T}_n)$
11: Derive $I_{0 \to 1}(f)$, $I_{1 \to 1}(f)$, and $I_1(f)$
12: **if** $I_1(f) \neq \emptyset$ **then**
13:    **for all** $a \in \{0,1\}$ **do**
14:      **for all** $x_i \in I_{a \to 1}$ **do**
15:        Append (x_i, a) to $NewLayer$ $\triangleright$ store cond. canalizing variable and input
16:        Append $(x_i, a+1)$ to $NegNewLayer$ $\triangleright$ store noncanalizing input
17:    Append $(NewLayer, 1)$ to $Layers$ $\triangleright$ store canalized output
18: **else**
19:    Compute $f \cdot \mathbf{T}_n$ and $f \cdot (M - \mathbf{T}_n)$
20:    Derive $I_{0 \to 0}(f)$, $I_{1 \to 0}(f)$, and $I_0(f)$
21:    **if** $I_0(f) \neq \emptyset$ **then**
22:      **for all** $a \in \{0,1\}$ **do**
23:        **for all** $x_i \in I_{a \to 0}$ **do**
24:          Append (x_i, a) to $NewLayer$ $\triangleright$ store cond. canalizing variable and input
25:          Append $(x_i, a+1)$ to $NegNewLayer$ $\triangleright$ store noncanalizing input
26:      Append $(NewLayer, 0)$ to $Layers$ $\triangleright$ store canalized output
27:    **else return** $(Layers, f)$ $\triangleright$ return layers and core function
28: Set $f = f(NegNewLayer)$ $\triangleright$ evaluate f on noncanalizing inputs
                   $\triangleright$ of variables in $NegNewLayer$;
29: **Goto** Step 2 $\triangleright$ repeat using the new f

---

the run time for random noncanalizing functions and for random nested canalizing (*i.e.*, most canalizing) functions. For each even number of variables $n \in [4, 16]$ and for both types of functions, we randomly generated five Boolean functions of that type as a vector of length 2^n, which served as the input for Algorithm 2. To use the same sets of random functions in Algorithm 1, we transformed the vectors into polynomials in disjunctive normal form.

At small numbers of variables ($n \leq 10$), both algorithms were on average faster at determining that there were no canalizing variables than at finding all layers of an NCF (Figure 2A). Interestingly for functions in many variables, this switched for Algorithm 1; its run time increased only marginally for NCFs, much slower than for noncanalizing functions. This is likely because Algorithm 1 benefits from a high amount of structure in nested canalizing polynomials, highlighting the advantage of the use of the polynomial representation over truth tables (*i.e.*, vectors). Note that the algorithms are implemented in two different programming languages, which prohibits a direct comparison of the run time.

Algorithm 2 can be sped up by pre-computing the left-hand side of the truth table $\mathbf{T}_m$ for all m with $1 \leq m \leq n$. In our limited experiment this reduced the average run time by 38.6% ($n = 4$) to 95.0% ($n = 20$) for noncanalizing functions and by 12.8% ($n = 4$) to 83.4% ($n = 20$) for NCFs (Figure 2B).

The findings from these bench-marking experiments align with theoretical computations of the worst-case time complexity, which is $O(n^2 2^n)$ for both algorithms. For Algorithm 1, the number of checks (in the worst case where, in each iteration, the last evaluated variable is the only canalizing) is

$$n + n - 1 + \cdots + 2 + 0 = \frac{n(n+1)}{2} - 1 = O(n^2)$$

(notice the zero because the last layer of an NCF contains at least two variables, Theorem 4) and each of these checks requires $O(2^n)$ polynomial evaluations. Thus, Algorithm 1 is $O(n^2 2^n)$. For Algorithm 2, the matrices that are evaluated to find canalizing variables are of size $n 2^n$ (Lines 11 and 20 in Algorithm 2) and in the worst case,
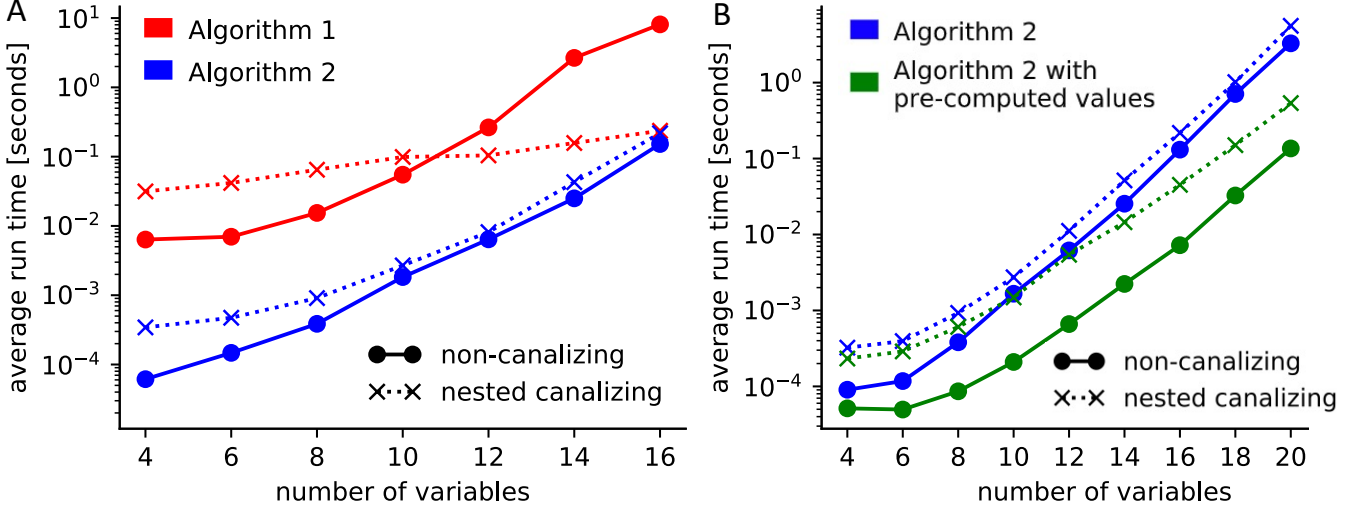
Fig. 2. **Comparison of the run time of the algorithms.** (A) For both algorithms, we compared the average computation time of the layers for noncanalizing (solid lines with circles) and nested canalizing (dotted lines with x) Boolean functions with a fixed number of variables, n (x-axis). (B) Comparison of the basic version of Algorithm 2 (blue) and a modified version of the algorithm where the left-hand side of the truth table $\mathbf{T}_m$ has been pre-computed for all m with $1 \leq m \leq n$.

we find n canalizing variables at once yielding another iteration over n variables (Lines 14 and 23 Algorithm 2). Thus, Algorithm 2 is $O(n^2 2^n)$.

6 Applications

In this section, we present two applications that utilize the canalizing structure of an NCF: converting the function into disjunctive normal form (Section 6.1), and reverse-engineering an unknown function from (partial) knowledge of its layers (Section 7).

6.1 Finding disjunctive normal forms using canalizing layers

A disjunctive normal form (DNF) is a canonical representation of Boolean functions, which consists of a disjunction of conjunctions. A disjunctive normal form is a useful representation of Boolean functions that has been used for many applications. For instance, a minimal DNF is needed for the identification of stable motifs [30,29]. In this section we will use the canalizing layers to obtain a disjunctive normal form expression for a nested canalizing function. In general, the Quine-McCluskey algorithm [24] can be used to find a disjunctive normal form for any Boolean function.

The following proposition is a special case of Theorem 4 where we assume that the Boolean function is nested canalizing.

Proposition 16 *Every nested canalizing function*

$f(x_1, \ldots, x_n)$ *can be uniquely written as*

$$
\begin{aligned}
&f(x_1, \ldots, x_n) = \\
&M_1(M_2(\ldots(M_{r-1}(M_r + 1) + 1)\ldots) + 1) + b,
\end{aligned}
\tag{3}
$$

where $M_i = \prod_{j=1}^{k_i}(y_{i_j} + 1)$, $y_{i_j} \in \{x_{i_j}, \overline{x}_{i_j}\}$, *and* $n = k_1 + \cdots + k_r$. *Each variable* y_i *appears in exactly one of the* $M_1, \ldots, M_r$.

Remark 17

(1) *Note that the extended monomials in Proposition 16 can be written as*

$$
M_i = \prod_{j=1}^{k_i}(y_{i_j} + 1) = \overline{y}_{i_1} \wedge \cdots \wedge \overline{y}_{i_{k_i}}.
$$

(2) *The negated form of* f *is* $f + 1$. *Thus to obtain the canalizing layers description of* $\overline{f}$ *we add 1 to Equation 3:*

$$
\begin{aligned}
&\overline{f}(x_1, \ldots, x_n) = \\
&M_1(M_2(\cdots(M_{r-1}(M_r + 1) + 1)\cdots) + 1) + b + 1.
\end{aligned}
\tag{4}
$$

Now using the first part of Remark 17, we introduce the following notation

$$
\overline{M}_i = y_{i_1} \vee \cdots \vee y_{i_{k_i}}.
$$

9

Given a nested canalizing function f written in the format of Equation 3, one can obtain a disjunctive normal form for f.

Theorem 18 *Let f be a nested canalizing function that is written as in Equation 3. Then the following formulas provide a disjunctive normal form for f depending of the value of b in Equation 3.*

(1) If $b = 0$, then a DNF of f is given by

$$f = \bigvee_{i=0}^{\lfloor r/2 \rfloor} \left(M_{2i+1} \wedge \bigwedge_{j=1}^{i} \overline{M}_{2j} \right), \qquad (5)$$

where $M_{r+1} = 1$ if needed.
(2) If $b = 1$, then a DNF of f is given by

$$f = \bigvee_{i=0}^{\lfloor r/2 \rfloor} \left(\overline{M}_{2i+1} \wedge \bigwedge_{j=1}^{i} M_{2j} \right), \qquad (6)$$

where $\overline{M}_{r+1} = 1$ if needed.

PROOF. We consider two cases for the value of b.

(1) If $b = 0$, then from Equation 3 f can be written as

$$f(x_1, \ldots, x_n) =$$
$$M_1(M_2(\ldots(M_{r-1}(M_r + 1) + 1)\ldots) + 1),$$

which can also be written as

$$f = \bigvee_{i=0}^{\lfloor r/2 \rfloor} \left(M_{2i+1} \wedge \bigwedge_{j=1}^{i} \overline{M}_{2j} \right).$$

Then notice that

$$M_{2i+1} \wedge \overline{M}_{2j} = M_{2i+1} \wedge (y_{i_1} \vee \cdots \vee y_{i_{k_i}})$$
$$= \bigvee_{k=1}^{k_i} M_{2i+1} \wedge y_{i_j}.$$

Thus Equation 5 is in disjunctive normal form.
(2) If $b = 1$, then from Equation 3 f can be written as

$$f(x_1, \ldots, x_n) =$$
$$M_1(M_2(\ldots(M_{r-1}(M_r + 1) + 1)\ldots) + 1) + 1,$$

which can also be written as

$$f = \bigvee_{i=0}^{\lfloor r/2 \rfloor} \left(\overline{M}_{2i+1} \wedge \bigwedge_{j=1}^{i} M_{2j} \right),$$

Then notice that

$$\overline{M}_{2i+1} \wedge M_{2j} = (y_{i_1} \vee \cdots \vee y_{i_{k_i}}) \wedge M_{2j} = \bigvee_{k=1}^{k_i} y_{i_j} \wedge M_{2j}.$$

Thus Equation 6 is in disjunctive normal form.

Remark 19 *For Theorem 18, when $i = 0$, the term inside the parentheses of Equation 5 (Equation 6) reduces to M_1 $(\overline{M}_1)$.*

Example 20 *Let $f = x_1x_2x_3x_4 + x_1x_2x_3 + x_1x_2x_4 + x_2x_3x_4 + x_1x_2 + x_1x_3 + x_2x_3 + x_2x_4 + x_1 + x_2 + x_3 + 1$. Using Algorithm 5.1, we find that the layer structure of this function is $f = M_1(M_2 + 1)$ where $M_1 = (x_1 + 1)(x_3 + 1)$ and $M_2 = x_2(x_4 + 1)$.*

Thus the value of b in Equation 3 is $b = 0$. Then from Equation 5, the disjunctive normal form is $f = M_1 \wedge \overline{M}_2 = (M_1 \wedge \overline{x}_2) \vee (M_1 \wedge x_4) = (\overline{x}_1 \wedge \overline{x}_3 \wedge \overline{x}_2) \vee (\overline{x}_1 \wedge \overline{x}_3 \wedge x_4)$.

Example 21 *Let $f = x_1x_2x_3x_4 + x_1x_2x_3 + x_2x_3x_4 + x_2x_3 + x_4$. Using Algorithm 5.1, we find that the layer structure of this function is $f = M_1(M_2 + 1) + 1$ where $M_1 = x_4 + 1$ and $M_2 = (x_1 + 1)x_2x_3$. Thus the value of b in Equation 3 is $b = 1$. Then from Equation 6, the disjunctive normal form is $f = \overline{M}_1 \vee M_2 = x_4 \vee (\overline{x}_1 \wedge x_2 \wedge x_3)$.*

6.2 Reverse engineering nested canalizing functions

When part of the canalizing layer structure of an NCF is known, one can generate all NCFs with that structure using the parametrization of NCFs given in (2), together with the following corollary to Theorem 4.

Corollary 22 *Let $f(x_1, \ldots, x_n)$ be a polynomial over $\mathbb{F}_2$.*

(1) A variable x_i is canalizing in the first layer of f with canalizing input 0 if and only if $f(x_1, \ldots, x_n) - f(0, \ldots, 0) = x_i \cdot g$ for some function g.
(2) A variable x_i is canalizing in the first layer of f with canalizing input 1 if and only if $f(x_1, \ldots, x_i + 1, \ldots, x_n) - f(0, \ldots, 0, x_i = 1, 0, \ldots, 0) = x_i \cdot g$ for some function g.

PROOF. Part 1 follows directly from Theorem 4. Part 2 also follows from Theorem 4 by noticing that the constant term of $f(x_1, \ldots, x_i + 1, \ldots, x_n)$ is obtained by setting $x_i = 1$ and $x_j = 0$ for $j \neq i$, so it is given by $f(0, \ldots, 0, x_i = 1, 0, \ldots, 0)$.

Example 23 *Let $f = x_1x_2 + x_1 + 1 \in \mathbb{F}_2[x_1, x_2]$. Then $f(x_1, x_2) - f(0, 0) = x_1x_2 + x_1 + 1 - 1 = x_1x_2 + x_1$ and since it can be factored as $x_1(x_2 + 1)$, we see that x_1 is a*

canalizing variable of f with canalizing input 0. Similarly, $f(x_1, x_2 + 1) - f(0, 1) = x_1 x_2$, so we have that x_2 is another canalizing variable of f with canalizing input 1.

7 Reverse engineering of the mammalian cell cycle

Corollary 22 allows for quick identification of the canalizing variables in the outermost layer of a function and we will use it as we "reverse engineer" an NCF from partial information about its canalizing structure. To illustrate the process we will consider the regulation of the mammalian cell cycle and specifically how the members of the E2F family of transcription factors are regulated by other elements. This example is based on the work in [2] in which the mammalian cell cycle is modeled using a logical framework.

Mammalian cell division is tightly controlled and coordinated with the overall growth of the organism. The protein Rb forms a complex with members of the E2F family of transcription factors, turning them from transcriptional activators to repressors. As a result, Rb is considered an inhibitor of E2F and so is CycB. CycA plays a more complicated role in the regulation of E2F. While generally an inhibitor of E2F, CycA's effect on E2F is known to be altered by p27 but we will assume we do not know how.

Thus the layer structure of the polynomial that models the dynamics of E2F will be an NCF in variables $x_1 = $ CycB, $x_2 = $ Rb, $x_3 = $ p27, and $x_4 = $ CycA distributed across two layers as follows: *Layer 1* has canalizing output 0 and contains x_1 with canalizing input 1 and x_2 also with canalizing input 1; *Layer 2* has unknown canalizing output and contains x_3 with unknown canalizing input and x_4 with canalizing input 0. The goal is to find *all* NCFs with this prescribed canalizing structure.

First, we will use Theorem 3.3 in [8] which provides relationships among the coefficients in the NCF. (For ease of reading, we write below, for example, c_{123} instead of $c_{\{1,2,3\}}$.) Notice that not all the equations that follow from the theorem carry useful information. Here we present only those that are not trivially true.

$$c_{1234} = 1 \tag{7}$$
$$c_2 = c_{12}c_{234} \tag{8}$$
$$c_3 = c_{123}c_{234}c_{134} \tag{9}$$
$$c_4 = c_{234}c_{134}c_{124} \tag{10}$$
$$c_{13} = c_{123}c_{134} \tag{11}$$
$$c_{14} = c_{134}c_{124} \tag{12}$$
$$c_{23} = c_{123}c_{234} \tag{13}$$
$$c_{24} = c_{234}c_{124} \tag{14}$$
$$c_{34} = c_{234}c_{134} \tag{15}$$

Second, we use Corollary 3.6 in [8] and the fact that variables in the same layer have the same canalizing output, while those in adjacent layers have different canalizing output to ensure that x_1 and x_2 are in one layer and x_3 and x_4 and in a different layer. The corollary produces the following nontrivial equations:

$$c_1 = c_{12}c_{134} \tag{16}$$
$$c_{12} = c_{123}c_{124} + 1 \tag{17}$$

We solve the system of eleven equations (7)-(17) using Gröbner bases with lexicographic order to find that the system has five free parameters, $c_{134} = a, c_{234} = b, c_{124} = c, c_{123} = d$, and $c_\emptyset = e$. The rest of the coefficients can be expressed in terms of these free parameters as $c_1 = acd + a, c_2 = bcd + b, c_3 = abd, c_4 = abc, c_{12} = cd + 1, c_{13} = ad, c_{14} = ac, c_{23} = bd, c_{24} = bc$, and $c_{34} = ab$. Thus the family of functions that satisfy the prescribed canalizing structure is of the form

$$f = x_1 x_2 x_3 x_4 + (acd + a)x_1 + (bcd + b)x_2 + abdx_3 +$$
$$+abcx_4 + (cd + 1)x_1 x_2 + adx_1 x_3 + acx_1 x_4 + bdx_2 x_3 +$$
$$bcx_2 x_4 + abx_3 x_4 + ax_1 x_3 x_4 + bx_2 x_3 x_4 + cx_1 x_2 x_4 +$$
$$dx_1 x_2 x_3 + e.$$

So far, we have ensured that the above family of functions are nested canalizing in canalizing order x_1, x_2, x_3, x_4 and that x_1 and x_2 are in one layer and x_3 and x_4 are in a different layer. We have not guaranteed x_1 and x_2 will be in the first layer or that the variables will have the required canalizing input and corresponding output.

To ensure that variables x_1 and x_2 will be in the first layer and both have canalizing input 1, we will use Corollary 22 and consider $f(x_1 + 1, x_2 + 1, x_3, x_4) - f(1, 1, 0, 0)$, which expanded is

$$x_1 + ax_1 + cdx_1 + acdx_1 + x_2 + bx_2 + cdx_2 + bcdx_2 +$$
$$x_1 x_2 + cdx_1 x_2 + dx_3 + adx_3 + bdx_3 + abdx_3 + dx_1 x_3 +$$
$$adx_1 x_3 + dx_2 x_3 + bdx_2 x_3 + dx_1 x_2 x_3 + cx_4 + acx_4 +$$
$$bcx_4 + abcx_4 + cx_1 x_4 + acx_1 x_4 + cx_2 x_4 + bcx_2 x_4 +$$
$$cx_1 x_2 x_4 + x_3 x_4 + ax_3 x_4 + bx_3 x_4 + abx_3 x_4 + x_1 x_3 x_4 +$$
$$ax_1 x_3 x_4 + x_2 x_3 x_4 + bx_2 x_3 x_4 + x_1 x_2 x_3 x_4.$$

In order to be able to factor out x_1 and x_2 from all nonconstant terms, we need to select zero coefficients for all terms that do not have both x_1 and x_2, that is, set to zero the coefficients for the terms $x_1, x_2, x_3, x_4, x_1 x_3, x_2 x_3$, etc. This generates a system of 11 nonlinear equations

in variables a, b, c, d over $\mathbb{F}_2$ which has (lexicographic) Gröbner basis $\{a + 1, b + 1\}$. So we set $a = b = 1$ in f to obtain

$$f' = e + x_1 + cdx_1 + x_2 + cdx_2 + x_1x_2 + cdx_1x_2 +$$
$$dx_3 + dx_1x_3 + dx_2x_3 + dx_1x_2x_3 + cx_4 + cx_1x_4 +$$
$$cx_2x_4 + cx_1x_2x_4 + x_3x_4 + x_1x_3x_4 + x_2x_3x_4 +$$
$$x_1x_2x_3x_4.$$

Now notice that when we plug x_1's or x_2's canalizing value into f', we get $f'(1, x_2, x_3, x_4) = f'(x_1, 1, x_3, x_4) = 1 + cd + e$. Since we want the output to be 0, we set $e = cd + 1$. Furthermore, since we have already guaranteed that x_1 and x_2 will be in the first layer with canalizing input 1, we can now set them equal to their noncanalizing input, 0, and proceed with the next layer, working with the much simpler function $f'(0, 0, x_3, x_4) = 1 + cd + dx_3 + cx_4 + x_3x_4$.

We now want to make sure that x_4 has canalizing input 0. (Recall that we made sure that x_3 and x_4 are in a different layer by using Corollary 3.6 in [8].) Recalling Corollary 22, we need to be able to factor out x_4 from all nonconstant terms of $f'(0, 0, x_3, x_4)$ which requires us to set $d = 0$. The resulting function is $1 + cx_4 + x_3x_4$. We can quickly check that, indeed, when x_4 takes on its canalizing input of 0, the output is a constant, namely 1. Notice that although we did not explicitly prescribe the canalizing output of x_4, we made sure it is different from the canalizing output of Layer 1 or else x_4 would not be in a different layer from x_1 and x_2. The function $1 + cx_4 + x_3x_4$ depends on a single parameter c whose value decides the canalizing input of x_3 (which we kept undetermined). When $x_3 = 0$, the expression $1 + cx_4 + x_3x_4$ evaluates to $1 + cx_4$, meaning that if the canalizing input of x_3 is 0, then we must set $c = 0$; if $x_3 = 1$, the function evaluates to $1 + (c + 1)x_4$, *i.e.*, if the canalizing input of x_3 is 1, then we must set $c = 1$. Since the canalizing input of x_3 is to be left undetermined, we will keep c as a parameter and conclude that the one-parameter family of nested canalizing functions with the required canalizing structure is the set

$$\{1 + x_1 + x_2 + x_1x_2 + cx_4 + cx_1x_4 + cx_2x_4 + cx_1x_2x_4 +$$
$$x_3x_4 + x_1x_3x_4 + x_2x_3x_4 + x_1x_2x_3x_4 \mid c \in \mathbb{F}_2\}.$$

This family of NCFs is consistent with the model proposed in [2]. With the same variable naming convention, their Boolean equation for E2F is $(\overline{x_2} \wedge \overline{x_4} \wedge \overline{x_1}) \vee (x_3 \wedge \overline{x_2} \wedge \overline{x_1})$, which written as a polynomial over $\mathbb{F}_2$ is

$$1 + x_1 + x_2 + x_1x_2 + x_4 + x_1x_4 + x_2x_4 + x_1x_2x_4 +$$
$$x_3x_4 + x_1x_3x_4 + x_2x_3x_4 + x_1x_2x_3x_4,$$

matching the form of our family of NCFs when $c = 1$. Recall that for the sake of our example we chose to assume that we did not know the role of p27 in altering the regulatory effect of CycA on E2F. Suppose that we now learn that, when present, p27 does not allow the inhibition of E2F by CycA. Equipped with this information we set $c = 1$ so that when both CycA and p27 are present, E2F remains present, arriving to the same model for E2F as the one proposed in [2].

8   Conclusion

This work focuses on the canalization structure of Boolean functions, specifically through studying the layer structure of a function. Knowing the layer structure of a Boolean function may aid researchers in multiple ways. It may be used to inform which experiments to prioritize when the goal is the identification of the correct update rule. The hierarchical layer structure may further reveal the level of importance of specific input variables, guiding experimentalists on which aspects of genetic control to focus.

In this paper we developed and implemented two algorithms for finding the layer structure of a general Boolean function, and proposed a third algorithm for the special case of nested canalizing functions. The implementation in Python requires the truth table format of the function while the Macaulay2 version requires a polynomial version of the function. We showed that the use of the polynomial version is advantageous when the function exhibits canalization; if the function is noncanalizing, the run time of both algorithms increases very similarly when the number of variables increases, at an exponential speed. All code is available at the GitHub repository https://github.com/ckadelka/BooleanCanalization. We demonstrated two uses of canalizing layers. In one application given the layer structure of a nested canalizing function, we showed how to find its disjunctive normal form, an **NP**-hard problem in the general case. In a second application we showed how to recover the complete family of functions that satisfy given partial layer information.

Improving the space- and time-efficiency of the algorithms for special classes of functions that take advantage of their structure (such as symmetric functions) or developing multi-state implementations may be conducive for increasing the scope of applications.

Acknowledgements

References

[1] Bryan C Daniels, Hyunju Kim, Douglas Moore, Siyu Zhou, Harrison B Smith, Bradley Karas, Stuart A Kauffman, and Sara I Walker. Criticality distinguishes the ensemble of biological regulatory networks. *Physical review letters*, 121(13):138102, 2018.

[2] Adrien Fauré, Aurélien Naldi, Claudine Chaouiya, and Denis Thieffry. Dynamic analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics (Oxford, England)*, 22:e124–31, 08 2006.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[4] Alexander J Gates, Rion Brattig Correia, Xuan Wang, and Luis M Rocha. The effective graph reveals redundancy, canalization, and control pathways in biochemical regulation and signaling. *Proceedings of the National Academy of Sciences*, 118(12), 2021.

[5] Daniel R. Grayson and Michael E. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at http://www.math.uiuc.edu/Macaulay2/.

[6] Stephen E Harris, Bruce K Sawhill, Andrew Wuensche, and Stuart Kauffman. A model of transcriptional regulatory networks based on biases in the observed regulation rules. *Complexity*, 7(4):23–40, 2002.

[7] Qijun He and Matthew Macauley. Stratification and enumeration of Boolean functions by canalizing depth. *Physica D: Nonlinear Phenomena*, 314:1–8, 2016.

[8] Abdul Salam Jarrah, Blessilda Raposa, and Reinhard Laubenbacher. Nested canalyzing, unate cascade, and polynomial functions. *Physica D: Nonlinear Phenomena*, 233(2):167–174, 2007.

[9] Claus Kadelka, Taras-Michael Butrie, Evan Hilton, Jack Kinseth, and Haris Serdarevic. A meta-analysis of Boolean network models reveals design principles of gene regulatory networks. *arXiv:2009.01216*, 2020.

[10] Claus Kadelka, Benjamin Keilty, and Reinhard Laubenbacher. Collectively canalizing Boolean functions. *arXiv:2008.13741*, 2020.

[11] Claus Kadelka, Jack Kuipers, and Reinhard Laubenbacher. The influence of canalization on the robustness of Boolean networks. *Physica D: Nonlinear Phenomena*, 353:39–47, 2017.

[12] Claus Kadelka, Yuan Li, Jack Kuipers, John O Adeyeye, and Reinhard Laubenbacher. Multistate nested canalizing functions and their networks. *Theoretical Computer Science*, 675:1–14, 2017.

[13] Fredrik Karlsson and Michael Hörnquist. Order or chaos in Boolean gene networks depends on the mean fraction of canalizing functions. *Physica A: Statistical Mechanics and its Applications*, 384(2):747–757, 2007.

[14] Stuart Kauffman. The large scale structure and dynamics of gene control circuits: An ensemble approach. *Journal of Theoretical Biology*, 44(1):167–190, 1974.

[15] Stuart Kauffman, Carsten Peterson, Björn Samuelsson, and Carl Troein. Genetic networks with canalyzing Boolean rules are always stable. *Proceedings of the National Academy of Sciences*, 101(49):17102–17107, 2004.

[16] Lori Layne, Elena S Dimitrova, and Matthew Macauley. Nested canalyzing depth and network stability. *Bulletin of Mathematical Biology*, 74(2):422–433, 2012.

[17] Yuan Li, John O. Adeyeye, David Murrugarra, Boris Aguilar, and Reinhard Laubenbacher. Boolean nested canalizing functions: A comprehensive analysis. *Theoretical Computer Science*, 481(0):24–36, 4 2013.

[18] K. K. Maitra. Cascaded switching networks of two-input flexible cells. *IRE Trans. Electron. Comput.*, EC-11:136–143, 1962.

[19] Manuel Marques-Pita and Luis M Rocha. Canalization and control in automata networks: Body segmentation in *Drosophila melanogaster*. *PLoS one*, 8(3):e55946, 2013.

[20] David Murrugarra and Elena S Dimitrova. Molecular network control through Boolean canalization. *EURASIP J Bioinform Syst Biol*, 2015(1):9, Dec 2015.

[21] David Murrugarra and Elena S Dimitrova. Quantifying the total effect of edge interventions in discrete multistate networks. *Automatica*, 125:109453, 2021.

[22] David Murrugarra and Reinhard Laubenbacher. Regulatory patterns in molecular interaction networks. *Journal of Theoretical Biology*, 288(0):66–72, 11 2011.

[23] David Murrugarra and Reinhard Laubenbacher. The number of multistate nested canalyzing functions. *Physica D: Nonlinear Phenomena*, 241(10):929–938, 5 2012.

[24] Willard Van Orman Quine. The problem of simplifying truth functions. *The American Mathematical Monthly*, 59(8):521–531, 1952.

[25] Daniel J Rosenkrantz, Madhav V Marathe, SS Ravi, and Richard E Stearns. Symmetry properties of nested canalyzing functions. *Discrete Mathematics and Theoretical Computer Science*, 21(4):1O–1O, 2019.

[26] Tsutomu Sasao and Kozo Kinoshita. On the number of fanout-free functions and unate cascade functions. *IEEE Transactions on Computers*, C-28(1):66–72, jan 1979.

[27] Mark L Siegal and Aviv Bergman. Waddington's canalization revisited: Developmental stability and evolution. *Proc Natl Acad Sci USA*, 99(16):10528–32, Aug 2002.

[28] Conrad H Waddington. Canalization of development and the inheritance of acquired characters. *Nature*, 150(3811):563–565, 1942.

[29] Gang Yang, Jorge Gómez Tejeda Zañudo, and Réka Albert. Target control in logical models using the domain of influence of nodes. *Frontiers in Physiology*, 9:454, 2018.

[30] Jorge G T Zañudo and Réka Albert. Cell fate reprogramming by control of intracellular network dynamics. *PLoS Comput Biol*, 11(4):e1004193, Apr 2015.