An Iterative Approach for Counting Reduced Ordered Binary Decision Diagrams

Julien Clément, Antoine Genitrini

An Iterative Approach for Counting Reduced Ordered Binary Decision Diagrams

Julien Clément
Normandie Université, UNICAEN, ENSICAEN, CNRS, GREYC – UMR 6072, France

Antoine Genitrini
Sorbonne Université, CNRS, LIP6 – UMR 7606, F-75005 Paris, France

Abstract

For three decades *binary decision diagrams*, a data structure efficiently representing Boolean functions, have been widely used in many distinct contexts like model verification, machine learning, cryptography and also resolution of combinatorial problems. The most famous variant, called reduced ordered binary decision diagram (ROBDD for short), can be viewed as the result of a compaction procedure on the full decision tree. A useful property is that once an order over the Boolean variables is fixed, each Boolean function is represented by exactly one ROBDD. In this paper we aim at computing the *exact distribution of the Boolean functions in k variables according to the ROBDD size*, where the ROBDD size is equal to the number of decision nodes of the underlying directed acyclic graph (DAG) structure. Recall the number of Boolean functions with k variables is equal to 2^{2^k}, which is of double exponential growth with respect to the number of variables. The maximal size of a ROBDD with k variables is $M_k \approx 2^k/k$. Apart from the natural combinatorial explosion observed, another difficulty for computing the distribution according to size is to take into account dependencies within the DAG structure of ROBDDs. In this paper, we develop the first polynomial algorithm to derive the distribution of Boolean functions over k variables with respect to ROBDD size denoted by n. The algorithm computes the (enumerative) generating function of ROBDDs with k variables up to size n. It performs $O(k\ n^4)$ arithmetical operations on integers and necessitates storing $O((k+n)n^2)$ integers with bit length $O(n \log n)$. Our new approach relies on a decomposition of ROBDDs layer by layer and on an inclusion-exclusion argument.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Generating functions; Mathematics of computing → Combinatoric problems; Theory of computation → Generating random combinatorial structures; Information systems → Data compression; Theory of computation → Data compression

Keywords and phrases Boolean Function, Reduced Ordered Binary Decision Diagram ({robdd}), Enumerative Combinatorics, Directed Acyclic Graph

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.36

Supplementary Material *Software*: `https://github.com/agenitrini/BDDgen`
archived at `swh:1:dir:dc717703d5409305685bff27e67735eba792508d`

Funding *Julien Clément*: PING/ACK [ANR-18-CE40-0011], C_SYDISI [ANR-19-CE48-0007]

Acknowledgements The authors thank the anonymous referees for their comments and suggested improvements. All these remarks have increased the quality of the paper.

1 Introduction

Three decades ago a central data structure in computer science, designed to represent Boolean functions, emerged under the name of Binary Decision Diagrams (or BDDs) [1]. Their algorithmic paradigm gives great advantages: it is based on a *divide-and-conquer* approach combined with a compaction process. Their benefits compared to other Boolean representations are so obvious that several dozens of BDD variants have been developed

in recent years. In his monograph [16], Wegener presents several ones like ROBDDs [2], OKFBDDs [5], QOBDDs [15], ZBDDs [10], and others. While most of these data structures are used in the context of verification [16], they also appear, for example, in the context of cryptography [9] or knowledge compilation [4]. Also, the size of the structure, depending on the compaction of a decision tree, allows improving classification in the context of machine learning [11]. Finally, some specific BDDs are relevant to strategies for the resolution of combinatorial problems, cf. [8, vol. 4], like the classical satisfiability count problem.

The classical way to represent the different diagrams consists in their embedding as directed acyclic graphs (or DAGs). In the following we are interested in the original form of decision diagrams that are ROBDDs, for *Reduced Ordered Binary Decision Diagrams*. One of their fundamental properties relies on the single, thus canonical, representative property for each Boolean function (with a given order over the Boolean variables). In his book [8] Knuth recalls and proves several combinatorial results for ROBDDs. He is, for example, interested in the profile of a typical ROBDD, or in the way to combine two structures to represent a more complex Boolean function. However, thirty years after the takeoff of ROBDDs, the study of the distribution of Boolean functions with respect to the size, defined as the number of decision nodes (see Figure 2), of the DAG structure is not totally understood. The main problem is that no recursive characterization describing the structure of ROBDDs is known, as opposed, for instance, to the recursive decomposition of binary trees which is the core approach in their combinatorial studies (profile, width, depth).

Related work. An important step in the comprehension of the distribution of the Boolean functions according to their ROBDD size has been achieved by Wegener [15] and improved by Gröpl *et al.* [7]. These authors proved that almost all functions have the same ROBDD size up to a factor of $1 + o(1)$ when the number of variables k tends to infinity, exhibiting the Shannon effect (strong or weak depending on the value of k). The strong (respectively weak) Shannon effect states that almost all functions have the same ROBDD size as the largest ROBDDs up to a factor of $1 + f(k)$ with $f(k) = o(1)$ (resp. $f(k) = \Omega(1)$) as k tends to infinity (see also [14]). The reader may find an illustration of this phenomenon in Figure 1 with the plot of the exact distribution for $k = 13$ variables. A consequence of these first analyses is that picking *uniformly at random* a Boolean function whose ROBDD is small is not an easy task, although in practice ROBDDs are often not of exponential size (with respect to k).

In [12], the authors study, experimentally, numerically, and theoretically, the size of ROBDDs when the number k of variables is increasing. However, their main approach relies on an exhaustive enumeration of the decision trees of all Boolean functions, that are in a second step compressed into ROBDDs. The doubly exponential growth of Boolean functions over k variables, equal to 2^{2^k}, allows only to compute the first values for $k = 1, \ldots, 4$. Then the authors extrapolate the distributions by sampling decision trees (uniformly at random).

Later in the paper [3] we obtain similar combinatorial results. Using a new approach based on a partial recursive decomposition, we partition the ROBDDs according to their profile (which describes the number of nodes per level in the DAG). Another key feature of [3] is that we can restrict ourselves to a maximal size n for ROBDDs, as opposed to the exhaustive-oriented approach of [12]. Although more efficient, still algorithms with this approach are bound to be at least of complexity $\Omega(n\, k^{3/2 \cdot k^2 / \log k})$, while using a huge amount of extra memory. However, after a lengthy computation we obtain the exact distributions of the size of ROBDDs up to $k = 9$, thus partitioning the set of 2^{512} Boolean functions into ROBDDs of sizes ranging from 0 to 141.

Main results. In this paper, we describe an algorithm that calculates the exact distribution for ROBDDs up to size n in time complexity $O(k\ n^4)$ using $O((n+k)n^2)$ extra storage memory for integers. To the best of our knowledge, this is the first polynomial complexity algorithm computing the distribution of the ROBDD size for Boolean functions. Our combinatorial approach is based on an iteration process instead of a recursive approach.

We improve drastically on previous work and all the extrapolated results presented in [12] for Boolean functions up to 13 variables are now fully and exactly described. Using a personal computer, in a couple of minutes we obtain an exhaustive counting of the ROBDDs representing functions over 11 variables. With a computer with several hundreds gigabytes of RAM we compute the distribution over 13 variables in about a day. Indeed, we partition the 2^{8192} Boolean functions according to their ROBDDs size (which ranges from 0 to 1277).

In Figure 1 the exact distribution is depicted in two ways of presentation: a red point (x, y) states that 2^y functions have a ROBDD size x, in logarithmic scale; the blue curve is the probability distribution.

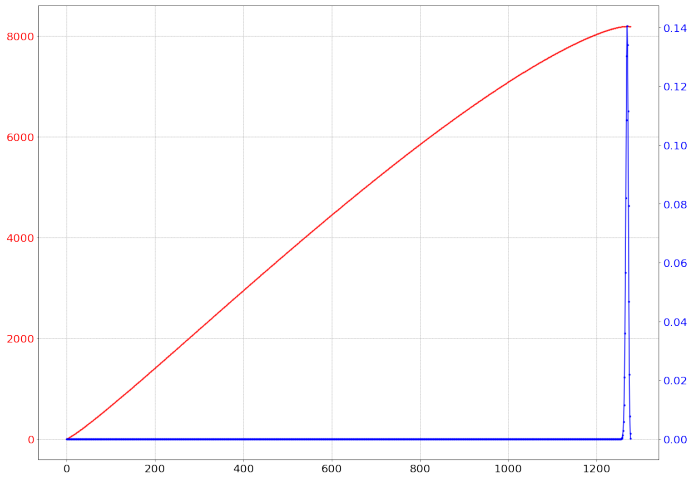

Figure 1 The ROBDD size distribution of Boolean functions in 13 variables. The exact distribution is depicted in two ways of presentation: the red curve is the logarithmic scale of the distribution; the blue curve is the probability distribution.

Organization of the paper. In Section 2 we present the formal notions and objects necessary for the description of our counting approach. Section 3 presents the iterative process for computing the number of ROBDDs having a given profile (the profile describes the number of decision nodes labelled with each variable). Section 3 also states the main result of this paper which is a formula for computing the number of ROBDDs involving linear maps over a polynomial ring. Finally, Section 4 presents the algorithmic context for computing the complete distribution (under the form of the enumerative generating function [6] of ROBDDs).

2 Preliminaries

A Boolean function in k variables is a function from the set $\{0,1\}^k$ into the set $\{0,1\}$. The set of functions is denoted by $\mathcal{B}_k$ and its cardinality is 2^{2^k}. Furthermore, for the rest of this paper, we choose an ordering of the sequence of variables that corresponds to $x_1, x_2, \ldots, x_k$. Any other ordering could be chosen, but one must be fixed.

2.1 Boolean functions representation

Figure 2 shows a decision tree representing a 4-variable Boolean function and its associated ROBDD. In both structures, traversing a path from the root to a leaf allows to evaluate the function for a given assignment. Being in a node labelled by x_i and going to its low child (using the dotted edge) corresponds with evaluating x_i to 0; going to its high child (using the solid edge) corresponds with evaluating x_i to 1.

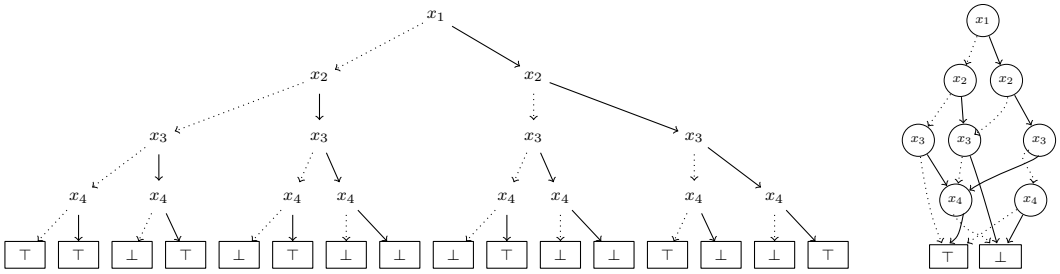

Figure 2 (left) A decision tree and **(right)** its associated ROBDD.

▶ **Definition 1** (DAG representation). *Let $f \in \mathcal{B}_k$ be a Boolean function in variables $x_1, \ldots, x_k$. The function f can be represented as a rooted* directed acyclic graph *(DAG for short) composed of internal* decision nodes*, labelled by variables and two* terminal nodes *labelled by $\{\bot, \top\}$ representing respectively the constants* 0 *and* 1*. Each decision node labelled by x_i has two children, the low child (resp. high child) such that traversing the edge to the low child (resp. high child) corresponds to assign x_i to* 0 *(resp. to* 1*). The* size *of a DAG is its number of decision nodes.*

▶ **Definition 2** (OBDD). *Let $f \in \mathcal{B}_k$ and pick one of its DAG representation. The DAG is called an Ordered Binary Decision Diagram for f (OBDD for short) when all paths from the root to a terminal node traverse decision nodes with index in increasing order.*

By taking an OBDD for a function with a pointed decision node ν labelled by x_i, and extracting the sub-DAG rooted in ν by taking all its descendants, we obtain an OBDD representing a Boolean function in the variables $x_i, x_{i+1}, \ldots, x_k$.

▶ **Definition 3** (ROBDD). *Let $f \in \mathcal{B}_k$ and let B be one of its OBDD. If all sub-DAGs of B are representing distinct Boolean functions, then B is called Reduced Ordered Binary Decision Diagram (ROBDD).*

Figure 3 shows the forbidden configurations in ROBDD and the operations used to compress the structure.

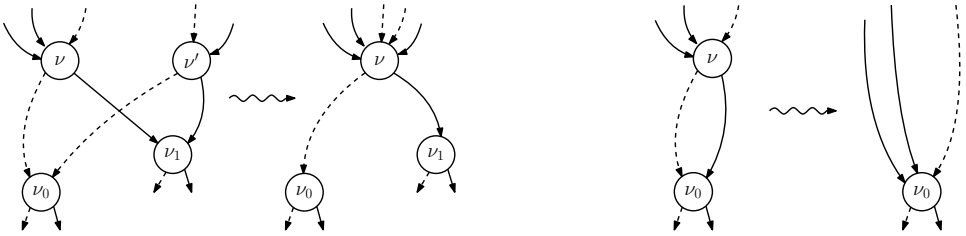

Figure 3 The two forbidden configurations in ROBDDs and the resulting operations when compressing: **(left)** ν and ν' are merged; **(right)** ν is deleted (from [7]).

▶ **Fact.** *Let $f \in \mathcal{B}_k$, there exists a single* ROBDD *representing f.*

The data structure of ROBDDs is especially famous due to this property of canonicity[1]. The reader may read Knuth [8] for the proof of uniqueness and several other properties satisfied by ROBDDs.

For the rest of the paper we are only dealing with ROBDDs. Furthermore, we take the point of view of layered ROBDD by studying and representing ROBDDs layer by layer: each layer contains all decision nodes labelled by the same variable. This layer-by-layer decomposition is natural since the variables are ordered. Thus, a ROBDD for a function in k variables is composed of k layers (plus a layer for the two terminal nodes). Note that a layer could be empty (which means the Boolean function does not depend on the particular variable associated to this layer).

▶ **Definition 4** (ROBDD's profile). *Let $f \in \mathcal{B}_k$ and let B be its* ROBDD*. Using the layer-by-layer point of view, we define the profile B to be the non-negative integer sequence $[p_1, p_2, \ldots, p_k]$ such that p_i is the number of nodes labelled by x_i in B, i.e. the size of the layer corresponding to x_i, for all $i \in \{1, \ldots, k\}$.*

2.2 Combinatorial description

In our previous paper [3] we proposed a kind of recursive decomposition of a ROBDD based on the low and high children of the root. Here we propose a new and simpler point of view, based on a layer-by-layer description, which is much more efficient for the counting problem (also for the generating problem). We need to introduce a generalization of a ROBDD, called multientry ROBDDs, which corresponds exactly to the structure obtained by removing some upper layers in a standard ROBDD (see Figure 4 for an example).

Informally a multientry ROBDD is a structure obtained by cutting off a certain number of the top layers of a ROBDD. The resulting structure is still a DAG, but with several sources. We also keep track of where the half-edges from the top layers were pointing. In Figure 4 a ROBDD and multientry ROBDD obtained by removing the 3 top layers are depicted.

▶ **Definition 5** (Multientry ROBDD). *A multientry* ROBDD *M with at most $k \geq 0$ variables and size $n \geq 0$ is a couple $(M, \mathfrak{E})$. The structure M is a layered* DAG *structure with n decision nodes $\mathcal{Q}$ distributed on k layers and two constant nodes $\{\top, \bot\}$ such that any two subgraphs are non-identical (as in* ROBDDs*). The multiset $\mathfrak{E}$ has its elements in $\mathcal{Q} \cup \{\top, \bot\}$ and is such that any source node in M (i.e., having in-degree 0), must appear at least once. The nodes in $\mathfrak{E}$ are called* distinguished *(and correspond with destination nodes of red half-edges in Figure 4).*

As a special case, a ROBDD is a multientry ROBDD having one source (the root) and a multiset $\mathfrak{E}$ reduced to the root (with multiplicity 1).

In the multientry ROBDD in Figure 4, by numbering the nodes from top to bottom and from left to right, i.e. the leftmost x_4 is number 1, the second x_4 is 2, the rightmost x_4 is 3, the rightmost x_5 is 4, …, x_6 is 7 and $\bot$ and $\top$ are respectively 8 and 9, we obtain the multiset $\mathfrak{E} = \{1, 2, 2, 3, 4, 4, 5\}$. We note that our definition of multientry ROBDD is similar (but not exactly identical) to the one of *shared*-BDDs presented by Knuth [8] to represent several Boolean functions in the same decision diagram.

Furthermore, remark that the same multientry ROBDD can be exhibited by cutting off top layers (not even the same number) of different ROBDDs.

[1] Uniqueness of the ROBDD for f is ensured when a variable ordering is fixed.

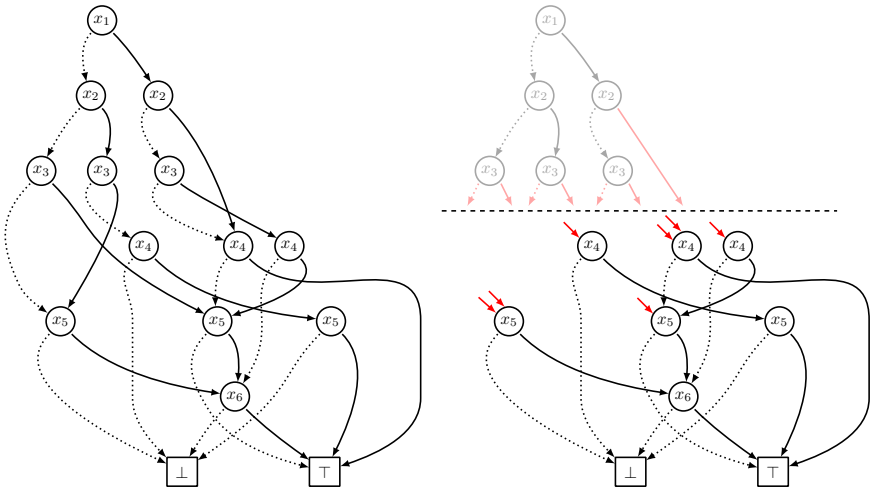

Figure 4 A ROBDD of size 13 (not counting the constant nodes $\{\bot$ and $\top\}$), with 6 variables and profile $\boldsymbol{p} = [1, 2, 3, 3, 3, 1]$. **(left)** A DAG representation of the ROBDD; **(right)** Cutting off top three layers, we obtain the multientry ROBDD keeping track with red half edges of nodes which were disconnected.

3 Full iterative counting formula

In the following we describe an approach to counting the number of ROBDDs with a given profile $\boldsymbol{p}$. We use a powerful algebraic representation to encapsulate a kind of inclusion-exclusion principle. This motivates the definition of a linear application on polynomials using substitutions. The linearity property of the applications is crucial for achieving our algorithm polynomial complexity. We consider the polynomial ring $\mathbb{Z}[X]$ and linear endomorphisms over $\mathbb{Z}[X]$ (i.e., linear maps between $\mathbb{Z}[X]$ and $\mathbb{Z}[X]$). Thus, for a linear map $g : \mathbb{Z}[X] \to \mathbb{Z}[X]$ and two polynomials P and Q in $\mathbb{Z}[X]$, $g[P + Q] = g[P] + g[Q]$, and for any scalar $\lambda \in \mathbb{Z}$ and polynomial $P \in \mathbb{Z}[X]$ we have $g[\lambda P] = \lambda g[P]$.

In the following theorem, we state the main result of the paper which gives access to the number of multientry ROBDDs for a given profile.

▶ **Theorem 6** (Multientry ROBDDs counting formula)**.** *For the family of linear maps $(\phi_r)_{r\geq 0}$, each mapping $\phi_r : \mathbb{Z}[X] \to \mathbb{Z}[X]$ is defined with respect to the canonical basis $(X^m)_{m\geq 0}$ by*

$$\phi_r[X^m] = \left(\prod_{i=0}^{r-1} (X^2 - X - i)\right) \cdot \left(\sum_{j=0}^{m-r} \binom{m}{j} \left\{ {m-j \atop r} \right\} X^j\right). \tag{1}$$

Let $M(\boldsymbol{p}, m)$ be the number of multientry ROBDDs with profile $\boldsymbol{p} = [p_1, \ldots, p_k]$ and m incoming half edges. We have, for $k \geq 0$,

$$M(\boldsymbol{p}, m) = (\phi_{\boldsymbol{p}}[X^m])_{X=2}\,, \tag{2}$$

where

- *$\phi_{\boldsymbol{p}}$ is the composition product $\phi_{p_k} \circ \phi_{p_{k-1}} \circ \cdots \circ \phi_{p_1}$;*
- *for a polynomial $P \in \mathbb{Z}[X]$, $(P)_{X=2}$ is the evaluation of P at $X = 2$.*

In the theorem, $\binom{m}{j}$ stands for the binomial coefficient and $\left\{ {n \atop k} \right\}$ is the Stirling number of the second kind counting the number ways to partition a set of n objects into k non-empty subsets.

▶ Remark 7. This is actually a stronger result that what we need for counting ROBDDs, since the number of ROBDDs corresponds to the special case where $\boldsymbol{p} = [p_1 = 1, p_2, \ldots, p_k]$ and $m = 1$ (meaning there is one source node in the top layer). However, the fact that we are able to compute $\phi_{\boldsymbol{p}}$ on the basis $(X^m)_{m\geq 0}$ is key to our approach.

The detailed the proof of Theorem 6 is deferred to after an example of such a computation.

▶ **Example.** *Let us consider a profile* $\boldsymbol{p} = [1, 2, 4, 2]$ *for* 4 *variables* x_1, x_2, x_3, x_4 *and choosing* $m = 1$ *in* (2)*. Then, for* $1 \leq i \leq 4$*, we compute iteratively* $\phi_{p_i} \circ \cdots \circ \phi_{p_1}(X)$*:*

$$\begin{aligned} X &\xmapsto{\phi_1} X^2 - X \\ &\xmapsto{\phi_2} X^4 - 2X^3 + X \\ &\xmapsto{\phi_4} X^8 - 4X^7 + 14X^5 - 6X^4 - 16X^3 + 5X^2 + 6X \\ &\xmapsto{\phi_2} 28X^{10} + 28X^9 - 98X^8 - 112X^7 + 76X^6 + 92X^5 - 12X^4 - 8X^3 + 6X^2. \end{aligned}$$

Evaluating the last polynomial at $X = 2$*, we get that there are* 11 160 *ROBDDs with profile* $[1, 2, 4, 2]$*. The power of our approach is that we could have stopped at any iteration, and still get the number of ROBDDs for the considered truncated profile. On the particular example this yields*

$\boldsymbol{p}$	$\phi_{\boldsymbol{p}}$	$(\phi_{\boldsymbol{p}})_{X=2}$
$[\,]$	X	2
$[1]$	$X^2 - X$	2
$[1, 2]$	$X^4 - 2X^3 + X$	2
$[1, 2, 4]$	$X^8 - 4X^7 + 14X^5 - 6X^4 - 16X^3 + 5X^2 + 6X$	0
$[1, 2, 4, 2]$	$28X^{10} + 28X^9 - 98X^8 - 112X^7 + 76X^6 + 92X^5 - 12X^4 - 8X^3 + 6X^2$	11 160

The number 0 *when considering* $[1, 2, 4]$ *may seem counterintuitive at first, but indeed a ROBDD can only have up to* 2 *nodes on its last layer, otherwise one node has to be a duplicate of another.*

Proof of Theorem 6. (Multientry ROBDDs counting formula). The proof is obtained by induction on the number $k \geq 0$ of layers with decision nodes.

Base case. When $k = 0$ and $n \geq 0$. The number of multientry ROBDDs is $M([\,], m) = 2^m$, i.e., X^m evaluated at $X = 2$, as we must map the m half edges to either one of the two constants. Note that if $m = 0$ then $M([\,], m) = 1$ corresponding to the void function (which is a special case).

Induction step. Now suppose Theorem 6 is true for $k \geq 0$.

Let us consider a profile of length $k + 1$ as $[r] \cdot \boldsymbol{p}$ with $r \geq 0$ and $\boldsymbol{p}$ a profile of length k.

- If $r = 0$, a simple computation shows that ϕ_0 is the identity. Hence, the empty layer can in fact be omitted since

$$\phi_{\boldsymbol{p}} \circ \phi_0 [X^m] = \phi_{\boldsymbol{p}} [X^m], \quad \text{so that } M([0] \cdot \boldsymbol{p}, m) = M(\boldsymbol{p}, m). \tag{3}$$

- From now on let us suppose $0 < r \leq m$. The set of m half edges pointing at the first layer can be decomposed in two subsets for $j \in \{0, \ldots, m - r\}$: j entries will go to layers below the first one, and $m - j$ entries will be mapped to the r nodes of the first layer. A Stirling number of the second kind $\left\{ {n \atop k} \right\}$ counts the number of ways to partition a set of n objects into k non-empty subsets. So there are $\binom{m}{j} \cdot \left\{ {m-j \atop r} \right\}$ such partitions. We write

$$M([r] \cdot \boldsymbol{p}, m) = \sum_{j=0}^{m-r} \binom{m}{j} \left\{ {m-j \atop r} \right\} f_{\boldsymbol{p}}^{(r)}(j), \tag{4}$$

where $f_{\boldsymbol{p}}^{(r)}(j)$ denotes the number of multientry ROBDDs with j free half edges and r pairs of half edges (the ones resulting from the r nodes of the first layer). These $2r$ half edges must thus obey the following constraints:

- in each pair, the two half edges must be distinct, i.e., point to different nodes;
- all r pairs of half edges must be distinct with one another (as pairs).

Our goal now is to get rid of the constraints coming from these r nodes and express all quantities in terms of free half edges.

The following equation translates the previous constraints on the pair of adjacent half edges coming from the first of the r nodes

$$f_{\boldsymbol{p}}^{(r)}(j) = f_{\boldsymbol{p}}^{(r-1)}(j+2) - f_{\boldsymbol{p}}^{(r-1)}(j+1) - (r-1)f_{\boldsymbol{p}}^{(r-1)}(j). \tag{5}$$

Indeed, the first term $f_{\boldsymbol{p}}^{(r-1)}(j+2)$ corresponds in adding 2 free half edges, and results in overcounting. Then following an inclusion-exclusion principle, firstly we subtract $f_{\boldsymbol{p}}^{(r-1)}(j+1)$ which would count the number of configurations if the two half edges in this pair were merged. Finally, we subtract $(r-1)f_{\boldsymbol{p}}^{(r-1)}(j)$ in (5). This last quantity counts the number of configurations if the pair of half edges was merged with one of the $r-1$ remaining ones (hence $r-1$ choices). Note that no additional free half edge is added because of this merge. This yields (5).

Solving the simple recurrence (5) with respect to r yields

$$f_{\boldsymbol{p}}^{(r)}(j) = \sum_{i=0}^{2r} a_i f_{\boldsymbol{p}}^{(0)}(i+j), \tag{6}$$

where coefficients (a_i) are obtained by identifying $P(X) = \prod_{i=0}^{r-1}(X^2 - X - i) = \sum_{i=0}^{2r} a_i X^i$.

At this point, we remark the equality true for $m \geq 0$

$$M(\boldsymbol{p}, m) = f_{\boldsymbol{p}}^{(0)}(m), \tag{7}$$

which reflects the fact that in $M(\boldsymbol{p}, m)$, all m half edges are unconstrained. Then (4) rewrites

$$M([r]\cdot\boldsymbol{p}, m) = \sum_{j=0}^{m-r} \binom{m}{j} \left\{ {m-j \atop r} \right\} \sum_{i=0}^{2r} a_i M(\boldsymbol{p}, i+j) = \sum_{i=0}^{m+r} c_i M(\boldsymbol{p}, i),$$

with coefficients c_i obtained by identifying $\sum_{i=0}^{m+r} c_i X^i = \phi_r(X^m)$ from (1).

By induction hypothesis on the length k of $\boldsymbol{p}$ we have for $0 \leq i \leq m+r$

$$M(\boldsymbol{p}, i) = \left(\phi_{\boldsymbol{p}}\left[X^i\right]\right)_{X=2}. \tag{8}$$

By linearity

$$\sum_{i=0}^{m+r} c_i \phi_{\boldsymbol{p}}\left[X^i\right] = \phi_{\boldsymbol{p}}\left[\sum_{i=0}^{m+r} c_i X^i\right] = \phi_{\boldsymbol{p}}\left[\phi_r[X^m]\right] = \phi_{\boldsymbol{p}} \circ \phi_r[X^m],$$

and finally

$$M([r]\cdot\boldsymbol{p}, m) = \sum_{i=0}^{m+r} c_i M(\boldsymbol{p}, i) = \sum_{i=0}^{m+r} c_i \left(\phi_{\boldsymbol{p}}\left[X^i\right]\right)_{X=2} = \left(\phi_{\boldsymbol{p}} \circ \phi_r[X^m]\right)_{X=2}$$

This ends the proof. ◀

4 Counting algorithms

In this section, we present an algorithm for counting ROBDDs of size n.

The time and space complexities are measured respectively in terms of arithmetical operations on $\mathbb{Z}$ and memory space used to store integers in $\mathbb{Z}$. When considering ROBDDs of size upper bounded by n, all integers in $\mathbb{Z}$ involved can be checked to be of bit length $O(n \log n) = O(\log n!)$.

The reader can find an implementation of the following algorithms at `https://github.com/agenitrini/BDDgen`.

4.1 Linear maps: precomputation step

A first step is to pre-compute a representation of linear maps $(\phi_r)_{r\geq 0}$. For a ROBDD of size n we know that the maximal number of half edges is $n+1$, and the maximal number of nodes on a layer is also loosely upper bound by n. Hence, it is sufficient to compute $\phi_r[X^m]$ in $\mathbb{Z}[X]$ for $0 \leq r \leq n$ and $0 \leq m \leq n+1$. In the form of (1), $\phi_r[X^m]$ is equal to $P_r(X)Q_{r,m}(X)$ with

$$P_r(X) = \prod_{i=0}^{r-1}(X^2 - X - i), \quad \text{and} \quad Q_{r,m}(X) = \sum_{j=0}^{m-r} \binom{m}{j} \left\{ {m-j \atop r} \right\} X^j.$$

So the first step is to compute coefficients of $P_r(X)$ and $Q_{r,m}(X)$. Concerning binomial coefficient and Stirling numbers of the second kind, both tables can be computed by a naive algorithm (for binomials, it is the famous Pascal's triangle) in space $O(n^2)$ with $O(n^2)$ arithmetic operations on integers.

Once these coefficients are available, we compute the products $\phi_r[X^m] = P_r(X)Q_{r,m}(X)$. Computing P_r from P_{r-1} necessitates $O(n)$ arithmetical operations on integers, yielding a total $O(n^2)$ number of arithmetical operations for the whole family $(P_r)_{r\leq n}$. Each polynomial $(Q_{r,m})$ necessitates $O(n)$ arithmetical operations per polynomial (supposing binomial coefficients and Stirling number of the second kind are precomputed). Finally, $\phi_r[X^m]$ is computed with $O(n^2)$ arithmetical operations (by a naive product of two polynomials). There are $O(n^2)$ such polynomials to compute. Hence, we get a total $O(n^4)$ of arithmetic operations using $O(n^3)$ storage memory space for coefficients. We thus get the next lemma.

▶ **Lemma 8** (Precomputing step: linear maps). *The precomputation step for the representation of linear maps by computing $\phi_r[X^m]$ for $0 \leq r \leq n$ and $0 \leq m \leq n+1$ necessitates $O(n^4)$ arithmetical operations on integers and uses memory space for $O(n^3)$ coefficients in $\mathbb{Z}$.*

4.2 Basic counting

The basic block in our approach is to be able to compute $\phi_r[P]$ for $r \geq 0$. This is done by Algorithm 1 which is a direct translation of Theorem 6 using the linearity of the linear maps $(\phi_r)_r$.

▶ **Proposition 9** (Complexity of basic step). *Let P be a polynomial of degree d. Algorithm 1 computes $\phi_r[P]$ and performs $O(rd + d^2)$ arithmetical operations over $\mathbb{Z}$ to compute $\phi_r[P]$, using $O(d^2)$ memory space*[2]*.*

[2] Recall in Proposition 9 we do not take into account the precomputation step of the family $(\phi_r[X^m])_r$.

Algorithm 1 Computing $\phi_r[P]$.

Input: an integer $r \geq 0$, a polynomial $P(X) = \sum_{m=0}^{d} p_m X^m \in \mathbb{Z}[X]$
Output: $\phi_r[P] \in \mathbb{Z}[X]$
$Q \leftarrow 0$
for m **from** r **to** d **do**
  $Q \leftarrow Q + p_m \phi_r[X^m]$
return Q

Algorithm 2 Computing $(\phi_{\boldsymbol{p}}[X])_{X=2}$.

Input: a profile $\boldsymbol{p} = [p_1, \ldots, p_k]$
Output: the number of ROBDDs with profile $\boldsymbol{p}$
$P \leftarrow X$
for i **from** 1 **to** k **do**
  $P \leftarrow \phi_{p_i}[P]$
return $P(2)$

Proof. Each polynomial $\phi_r[X^m]$ is of degree $r + m = O(r + d)$. Thus, the number of operations needed on coefficients is $O(rd + d^2)$ if $r > 0$ (or $O(1)$ if $r = 0$ since ϕ_0 is the identity). ◀

By Remark 7 and applying Theorem 6, it is straightforward to compute the number of ROBDDs with profile $\boldsymbol{p} = [p_1, \ldots, p_k]$ (using $m = 1$). The pseudocode is given in Algorithm 2. We have the following proposition.

▶ **Proposition 10.** *Algorithm 2 computes the number of ROBDDs of size n with k variables and given profile $\boldsymbol{p}$. It performs $O(k\, n^2)$ arithmetical operations over $\mathbb{Z}$ and uses $O(n)$ extra memory space to store integers.*

Proof. The number of ROBDDs is $(\phi_{\boldsymbol{p}}[X])_{X=2}$. The polynomial $\phi_{\boldsymbol{p}}[X]$ is computed by iterating k times a linear map of type ϕ_r starting from an initial polynomial X. By Proposition 9, the algorithm performs $O(n^2)$ operations for each iteration since polynomials have $O(n)$ coefficients. The total computation thus performs $O(k\, n^2)$ arithmetical operations over $\mathbb{Z}$ and use $O(n)$ memory space to store coefficients. Evaluating $(\phi_{\boldsymbol{p}}[X^m])_{X=2}$ at $X = 2$ can be done in time complexity $O(n)$ (by Horner's method for instance). ◀

4.3 Generating function for ROBDD size

The main goal of this section is compute the distribution of Boolean functions in k variables according to the ROBDD size. For $f \in \mathcal{B}_k$ a Boolean function, we let $\lambda(f)$ be the size of its ROBDD, i.e., its number of decision nodes. A convenient way to represent the distribution of the size λ on $\mathcal{B}_k$ consists in computing the generating function [6]

$$F_k(u) = \sum_{f \in \mathcal{B}_k} u^{\lambda(f)} = \sum_{i \geq 0} f_i u^i,$$

where u is a formal variable marking the size. Then for $i \geq 0$, the coefficient $f_i = [u^i]F_k(u)$ is the number of ROBDDs of size i with k variables, i.e. the notation $[u^i]F_k(u)$ corresponds to the coefficient-extraction of the monomial u^i. We also introduce the truncation $F_k^{\leq n}(u) = \sum_{0 \leq i \leq n} f_i u^i$ as the generating functions of ROBDDs of size less than or equal to n.

We first extend the formalism introduced in Section 3 and define a linear map $\varphi : \mathbb{Z}[u, X] \to \mathbb{Z}[u, X]$.

▶ **Definition 11.** *The linear map $\varphi : \mathbb{Z}[u, X] \to \mathbb{Z}[u, X]$ is defined via its action on the basis $(u^r X^m)_{r,m \geq 0}$ as*

$$\varphi : u^r X^m \mapsto \varphi[u^r X^m] = \sum_{i=0}^{m+1} u^{r+i} \phi_i[X^m]. \tag{9}$$

With this notation we have the following proposition.

▶ **Proposition 12** (Generating function for ROBDD size). *The generating function enumerating Boolean functions by considering the ROBDD size is given by*

$$F_k(u) = \left(\varphi^k[X]\right)_{X=2}, \quad \text{where } \varphi^k \text{ denotes the composition product } \underbrace{\varphi \circ \cdots \circ \varphi}_{k \text{ times}}.$$

Proof. Each application of φ corresponds with adding a layer. The formal variable u marks the number of decision nodes added on the current layer. ◀

We remark that in practice we can truncate polynomials, keeping only the terms useful along the computation. The key point here is that if we consider ROBDDs with size bounded by n, we should make all computation modulo u^{n+1}. Indeed, the formal variable u marks the number of nodes (which is bounded by n). Sections A.1 and A.2 int the appendix illustrate this point.

Algorithm 3 computes the bivariate polynomial $\varphi[X^m]$ for $m \geq 0$. Algorithm 4, computes recursively the iterated (univariate) version $\left(\varphi^\ell[X^m]\right)_{X=2}$ (that is the evaluation at $X = 2$).

Algorithm 3 Computing $\varphi[X^m]$.

Input: An integer $m \geq 0$
Output: Returns $\varphi[X^m] \in \mathbb{Z}[u, X]$
$Q \leftarrow 0$ ▷ $Q \in \mathbb{Z}[u]$
for r **from** 0 **to** m **do**
  $Q \leftarrow Q + u^r \phi_r[X^m]$
return Q

Algorithm 4 Computing $\left(\varphi^\ell[X^m]\right)_{X=2}$.

Input: Two integers ℓ, m
Output: Returns $\left(\varphi^\ell[X^m]\right)_{X=2} \in \mathbb{Z}[u]$
▷ `N.B.: Computations done modulo` u^{n+1}
  `where` n `is the maximal size for ROBDDs`
if $\ell = 0$ **then return** 2^m ▷ `base case`
$Q \leftarrow 0$ ▷ $Q \in \mathbb{Z}[u]$
$R \leftarrow \varphi(X^m)$ ▷ `Call Alg. 3`
for j **from** 0 **to** $\deg_X(R)$ **do**
  $M \leftarrow \left(\varphi^{\ell-1}[X^j]\right)_{X=2}$ ▷ `Call Alg. 4`
  $N \leftarrow [X^j]\, R(u, X)$ ▷ $N \in \mathbb{Z}[u]$
  $Q \leftarrow Q + M \cdot N$
return Q

▶ **Lemma 13.** *Algorithm 3 computes $\varphi[X^m] \in \mathbb{Z}[X, u]$, which has $O(n^2)$ integer coefficients. It performs $O(n^2)$ arithmetical operations over $\mathbb{Z}$.*

Algorithm 4 computes $\left(\varphi^\ell[X^m]\right)_{X=2} \in \mathbb{Z}[u]$, which has $O(n)$ coefficients. If we omit recursive calls, it performs $O(n^3)$ arithmetical operations over $\mathbb{Z}$, using memoization techniques with $O((n+k)n^2)$ extra memory storage.

Proof. For Algorithm 3, we perform $m = O(n)$ additions of polynomials in $\mathbb{Z}[X, u]$ with $O(n)$ terms, yielding $O(n^2)$ arithmetical operations over $\mathbb{Z}$. The result is a bivariate polynomial of bounded degree (n for the variable u, $2n$ for the variable X) yielding $O(n^2)$ coefficients. We suppose Algorithm 3 has access to polynomials $\phi_r[X^m] \in \mathbb{Z}[X]$ from a precomputation step.

For Algorithm 4, two ingredients are essential. Firstly we have to truncate polynomials modulo u^{n+1} so that operations on univariate polynomials have complexity $O(n)$ for addition and $O(n^2)$ for multiplication (using the naive multiplication on polynomials). Secondly we also use memoization techniques (meaning we compute in lazy manner intermediate results only once and keep it for further reference, at the expense of memory storage). That means that we consider that at the time we compute $\left(\varphi^\ell[X^m]\right)_{X=2}$, the polynomials $\left(\varphi^{\ell-1}[X^j]\right)_{X=2}$ are available (i.e., their complexity is taken into account independently). By an amortizing argument, the complexity of computing the complete family of polynomials $\left(\varphi^\ell[X^m]\right)_{X=2}$

$(0 \leq \ell \leq k$ and $0 \leq m \leq n+1)$ is still $O(n^3)$ arithmetical coefficients per polynomial. We need to store $O(kn^2)$ integer coefficients for memoization of all intermediate polynomials. We also suppose Algorithm 4 has access to bivariate polynomials $(\varphi[X^m] \in \mathbb{Z}[u, X])_{0 \leq m \leq n+1}$ from a precomputation step which requires $O(n^3)$ memory space for integer coefficients. A subtle point is to understand that we can truncate polynomials at each step in Algorithm 4 and still get the correct result: this can be proved by recurrence on ℓ (see also Section A.2 of the appendix for an example). ◀

Algorithm 4 also computes the generating function of ROBDDs for size up to n since posing $\ell = k$ and $m = 1$ we have

$$F_k^{\leq n}(u) = \left(\varphi^\ell[X]\right)_{X=2} \mod u^{n+1}.$$

▶ **Theorem 14** (Algorithm for computing the exact distribution)**.** *We compute the generating function $F_k^{\leq n}(u)$ of Boolean functions in $\mathcal{B}_k$ for ROBDDs of size less than or equal to n using $O(k\ n^4)$ arithmetical operations in $\mathbb{Z}$ and $O((k+n)\ n^2)$ for memory space storing integers.*

Proof. From Lemmas 13 and 8, the overall complexity is dominated by the computation of the family $(\phi_r[X^m])_{r,m}$ and the calls to Algorithm 4. By Lemma 13, each polynomial $\left(\varphi^\ell[X^j]\right)_{X=2}$ is computed with $O(n^3)$ arithmetic operations on integer coefficients and there are $O(k\ n)$ such polynomials. Hence, the total number operations over $\mathbb{Z}$ is $O(k\ n^4)$. Furthermore, we store $O(k\ n^2)$ coefficients in $\mathbb{Z}$ for memoization. We also store $O(n^3)$ coefficients for the family $(\phi_r[X^m])_{m,r} \in \mathbb{Z}[X]$. This yields the claimed complexity. ◀

To evaluate the complete size distribution we need to consider the size of largest ROBDDs with k variables.

▶ **Theorem 15** (Maximal size of ROBDDs)**.** *Let $k \geq 1$ be an integer, the maximal number of nodes in a ROBDD with at most k variables is*

$$M_k = 2^{k-\theta} - 3 + 2^{2^\theta}, \quad \textit{with } \theta = \lfloor \log_2(k - \lfloor \log_2(k) \rfloor) \rfloor.$$

The generating function $F_k(u)$ of Boolean functions in $\mathcal{B}_k$ according to the ROBDD size can be computed with $O(2^{4k}/k^3)$ arithmetical operations in $\mathbb{Z}$ and uses $O(2^{2k}/k)$ space.

Proof. Note that this formula is equivalent to the one given without proof by Pontus von Brömssen [13]. The existence of θ is proved in [12] and from there we can derive the explicit expression of θ (details omitted here). Then substituting $n = M_k$ in Theorem 14, and noting that $M_k \approx 2^k/k$ yields the compitational complexity result. ◀

Note this is the polynomial with respect to the maximal size of a ROBDD for k variables, hence a huge improvement compared to exponential brute force algorithms enumerating all 2^{2^k} Boolean functions and computing their ROBDD size obtained after a compaction process. With a careful implementation, we can achieve the computation of $F_k(u)$ for k up to 11 variables on a personal computer, and, on a high-performance computer with 512 GB RAM memory, for $k = 12$ in less than 1 hour 30 minutes, and even for $k = 13$ in less than 30 hours using the `PyPy` implementation of `Python`.

5 Conclusion

As an application of the counting approach of this paper we are able sample at random ROBDDs. More precisely, we can efficiently and uniformly pick ROBDDs either according to a given size, or even a given profile or a given spine. This is a great improvement when comparing to the classical uniform random generation over the set of Boolean functions, like in [12], that is drastically biased to the largest ROBDDs due to the Shannon effect. For instance with 12 variables, the probability of drawing uniformly a Boolean function giving a ROBDD of (quadratic in k) size $144 = 12^2$ is approximately $1.212 \cdot 10^{-957}$.

In practice, several classical functions have ROBDDs of small size. For example the symmetrical functions in k variables are associated with ROBDDs of quadratic size in k (see [8]). Hence, the approach described in this paper leads the way to provide (polynomial) uniform random generator for ROBDDs of small size (i.e., of size less than exponential).

An interesting future work consists in enumerating the BDD structures where our counting and sampling methods can be applied such as (non-reduced) OBDD, or ZDD which generally used to represent sets.

Finally, another research direction consists in noting that the generating function of ROBDDs with both size and number of variables can be specified thanks to an iterative process as in Theorem 12. It would be interesting to see if the machinery of analytic combinatorics [6] is amenable to this kind of specification.

References

1 R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

2 R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.

3 J. Clément and A. Genitrini. Binary decision diagrams: From tree compaction to sampling. In *LATIN: Theoretical Informatics – 14th Latin American Symposium, Proceedings*, volume 12118 of *LNCS*, pages 571–583. Springer, 2020.

4 A. Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Int. Res.*, 17(1):229–264, September 2002.

5 R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski. Efficient Representation and Manipulation of Switching Functions Based on Ordered Kronecker Functional Decision Diagrams. In *DAC'94*, pages 415–419, 1994.

6 P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, New York, NY, USA, 1 edition, 2009.

7 C. Gröpl, H. J. Prömel, and A. Srivastav. Ordered binary decision diagrams and the shannon effect. *Discret. Appl. Math.*, 142(1-3):67–85, 2004. doi:10.1016/j.dam.2003.02.003.

8 D. E. Knuth. *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms*. Addison-Wesley Professional, 2011.

9 L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure Function Evaluation with Ordered Binary Decision Diagrams. In *CCS'06*, pages 410–420. ACM, 2006.

10 S.-I. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. *30th ACM/IEEE Design Automation Conference*, pages 272–277, 1993.

11 C. Mues, B. Baesens, C. M. Files, and J. Vanthienen. Decision diagrams in machine learning: an empirical study on real-life credit-risk data. *Expert Systems with Applications*, 27(2):257–264, 2004.

12 J. Newton and D. Verna. A theoretical and numerical analysis of the worst-case size of reduced ordered binary decision diagrams. *ACM TCL*, 20(1):6:1–6:36, 2019.

13 N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences, September 2019. Sequence A327461.

14 J. Vuillemin and F. Béal. On the BDD of a Random Boolean Function. In *ASIAN'04*, pages 483–493, 2004.

15 I. Wegener. The size of reduced OBDDs and optimal read-once branching programs for almost all Boolean functions. In *GTCCS'94*, pages 252–263, 1994.

16 I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

A Appendix

A.1 Iterating φ (an example)

In this section, we illustrate how φ can be iterated to count the distribution of ROBDDs up to 4 variables. Let us consider the set of Boolean function $\mathcal{B}_k$ for $k \in \{1,2,3\}$ and pose $H_k(u,X) = \varphi^k[X]$.

- $H_1(u,X) = \varphi[X] = (\phi_0(X) + u\phi_1(X)) = \big(X^2 - X\big)u + X^2$. We can verify that substituting $X = 2$ we get $F_1(u) = H_1(u,2) = 2\,u + 2$. Indeed, $\mathcal{B}_1 = \{\bot, x_1, \overline{x_1}, \top\}$, with respective truth tables $\{\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}\}$ leading to 2 ROBDDs of size 1 and 2 ROBDDs of size 0, i.e., with no decision node.
- Adding a second layer, we get $H_2(u,X) = \varphi(H_1(u,X)) = \varphi^2[X]$, yielding

$$H_2(u,X) = \big(X^4 - 2\,X^3 + X\big)u^3 + 2\,\big(X^3 - X^2\big)u^2 + 2\,\big(X^2 - X\big)u + X.$$

  We get $F_2(u) = 2\,u^3 + 8\,u^2 + 4\,u + 2$, hence there are $2, 8, 4, 2$ ROBDDs of respective sizes $3, 2, 1$ and 0 for 16 Boolean functions on 2 variables.
- Iterating with a third layer, we get $H_3(u,X) = \varphi(H_1(u,X)) = \varphi^3[X]$ which has 34 terms and gives

$$F_3(u) = 74\,u^5 + 88\,u^4 + 62\,u^3 + 24\,u^2 + 6\,u + 2.$$

  Hence, there are respectively $74, 88, 62, 24, 6$ and 2 ROBDDs of size $5, 4, 3, 2, 1$ and 0.
- Adding a fourth layer yields for $H_4(u,X) = \varphi^k[X]$ a polynomial with 134 terms and

$$F_4(u) = 11160\,u^9 + 23280\,u^8 + 17666\,u^7 + 8928\,u^6 + 3248\,u^5 + 960\,u^4 + 236\,u^3 + 48\,u^2 + 8\,u + 2.$$

  There are 11160 ROBDDs with 4 variables of size 9, 23280 ROBDDs of size 8, etc.

Of course, we can check that $F_k(1) = 2^{2^k}$, which is the number of Boolean functions on k variables.

A.2 Truncating polynomials (an example)

In this subsection, we illustrate on an example the computational effect of truncating polynomials.

Let us consider that the generating function $F_3^{\leq 3}(u)$ of ROBDD with $k = 3$ variables and with less or equal to $n = 3$ decision nodes. The maximal size of a ROBDD for $k = 3$ variables is $M_3 = 7$ (7 decision nodes). Then $\varphi^3[X]$ is a polynomial with 34 terms. We can write (terms who would disappear modulo $u^{n+1} = u^4$ are grayed)

$$\begin{aligned}\varphi^3[X] = &\left(X^8 - 4\,X^7 + 14\,X^5 - 6\,X^4 - 16\,X^3 + 5\,X^2 + 6\,X\right)u^7\\ &+4\left(X^7 - 2\,X^6 - 3\,X^5 + 5\,X^4 + 4\,X^3 - 3\,X^2 - 2\,X\right)u^6\\ &+\left(8\,X^6 - 12\,X^5 - 11\,X^4 + 14\,X^3 + 4\,X^2 - 3\,X\right)u^5\\ &+2\left(5\,X^5 - 6\,X^4 - 5\,X^3 + 4\,X^2 + 2\,X\right)u^4\\ &+\left(9\,X^4 - 10\,X^3 - 2\,X^2 + 3\,X\right)u^3\\ &+6\left(X^3 - X^2\right)u^2\\ &+3\left(X^2 - X\right)u\\ &+X\end{aligned}$$

Truncating modulo u^4 and substituting $X = 2$ yields the polynomial

$$F_3^{\leq 4}(u) = 62\,u^3 + 24\,u^2 + 6\,u + 2.$$

The problem of computing all the terms before truncating the result is that there are $O(\frac{2^{2k}}{k^2})$ terms (since $M_k \approx 2^k/k$), hence a combinatorial explosion.

In contrast, Algorithm 4 works in a recursive manner and truncate polynomials along the way so that we have a polynomial number of terms. In the following, we describe with some details how to compute $(\varphi^3(X))_{X=2}$.

First, Algorithm 4 decomposes $(\varphi^3(X))_{X=2}$ as $(\varphi^2 \circ \varphi[X]))_{X=2}$. In general, $\varphi[X^m]$ is a polynomial of respective degree m and $2m$ in variables u and X and has $O(m^2)$ integer coefficients. We compute (collecting terms with respect to variable X)

$$\varphi[X] = (X^2 - X)u + X = X^2u + X(1-u). \tag{10}$$

Denoting $B_m^{(2)}(u) = (\varphi^2[X^m])_{X=2}$, our algorithm computes recursively for the basis $(1, X, X^2)$ (removed monomials modulo u^4 ar grayed)

$$\begin{aligned}B_0^{(2)}(u) &= 1\\ B_1^{(2)}(u) &= 2\,u^3 + 8\,u^2 + 4\,u + 2\\ B_2^{(2)}(u) &= 74\,u^4 + 90\,u^3 + 68\,u^2 + 20\,u + 4\end{aligned}$$

Then since φ is linear, we compute (still modulo u^4) using Equation (10)

$$\begin{aligned}(\varphi^3(X))_{X=2} &= B_2(u)u + B_1(u)(1-u)\\ &= u(90\,u^3 + 68\,u^2 + 20\,u + 4) + (1-u)(2\,u^3 + 8\,u^2 + 4\,u + 2)\\ &= 88\,u^4 + 62\,u^3 + 24\,u^2 + 6\,u + 2.\end{aligned}$$

In summary, computing modulo u^{n+1} along the process allows us to control the degree $O(n)$ of polynomials involved, which in turn ensures that the computation stays of polynomial complexity (with respect ti arithmetical operations on integers).

Observing such curves for k from 1 to 13, we notice the exponential growth of the largest ROBDDs when the number k of variables increases. Indeed, in Theorem 15 we define M_k to be the size of the largest ROBDDs with k variables. The sequence starts as $(M_k)_{k=1,\dots,13} = (1, 3, 5, 9, 17, 29, 45, 77, 141, 269, 509, 765, 1277)$.