COMBINATORICS OF REDUCED ORDERED BINARY DECISION DIAGRAMS: APPLICATION TO UNIFORM RANDOM SAMPLING

JULIEN CLÉMENT AND ANTOINE GENITRINI

Abstract. Since three decades binary decision diagrams, representing efficiently Boolean functions, are widely used, in many distinct contexts like model verification, machine learning, cryptography or 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. This structure has a useful property: 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 size of the underlying directed acyclic graph (dag) structure. Recall the number of Boolean functions is equal to \( 2^{2^k} \), which is of double exponential growth with respect to the number \( k \) of variables; hence a combinatorial explosion is to be expected. The maximal size of a robdd with \( k \) variables is \( M_k \sim 2^k/k \) and thus, the support of the robdd size distribution is also of length \( M_k \), making \( M_k \) a natural complexity unit for our problem. In this paper, we develop the first polynomial algorithm to derive the distribution of the Boolean functions with respect to their robdd sizes. The algorithm is essentially quartic in \( M_k \) for the time complexity and quadratic for the space complexity. The main obstacle is to take into account dependencies inside the dag structure, and we propose a new combinatorial counting procedure reminiscent of the inclusion-exclusion principle. As a by-product, we present an efficient polynomial unranking algorithm for robdds, which in turn yields a uniform random sampler over the set of robdds of a given size or of a given profile. This is a great improvement to the classical random sampler which is uniform over the set of all Boolean functions in \( k \) variables. Indeed, due to the Shannon effect, the uniform distribution over Boolean functions is heavily biased to extremely complex functions, with near maximal robdd size, thus preventing in practice to sample robdds with smaller size.

Keywords: Boolean Function, Reduced Ordered Binary Decision Diagram (robdd), Enumerative Combinatorics, Uniform Random sampling, Directed Acyclic Graph

1. Introduction

Three decades ago a data structure emerged under the name of Binary Decision Diagrams (or BDDs) \([1]\). They deserve to represent Boolean functions, thus are central in computer science. Their algorithm 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 variants BDDS have been developed in recent years. In his monograph \([20]\), Wegener presents several ones like ROBDDs \([2]\), OKFBDDS \([6]\), QOBDds \([19]\), ZBDDS \([15]\), and others. While most of these data structures are used in the context of verification \([20]\), they also appear, for example, in the context of cryptography \([12]\) or knowledge compilation \([5]\). Also, the size of the structure, depending on the compaction of a decision tree, allows to improve classification in the context of machine learning \([10]\). Some specific BDDs are also relevant to strategies for the resolution of combinatorial problems, cf \([11]\ 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 structures that are ROBDDs, for Reduced Ordered Binary Decision Diagrams. One of their fundamental properties relies on the single representative for each Boolean function (of a given number of

This work was supported by the French ANR Projects C_SyDySi [19-CE48-0007-01], and PING-ACK [18-CE40-0011].
Boolean variables). In his book [11] 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 BDDS, what appears somewhat unbelievable is that the results about the distribution of the Boolean functions according to their ROBDD size are still staggering. Another difficult question is about the distribution of the functions according to their DAG profile. The main problem to get improvements in these directions is that no recursive characterization was expected to describe the structure of ROBDDs, as opposed, for instance, to the recursive decomposition of binary trees which is the core approach in their combinatorial studies (profile, width, depth). Here we have no local-constraint for the decomposition of ROBDDs.

Related Work and disruption induced by our approach. An important step in the comprehension of the distribution of the Boolean functions according to their ROBDD size has been achieved by Wegener [19]. He proved the weak Shannon effect: 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. Later Gröpl et al. [9] improved the result by exhibiting when the strong Shannon effect takes place or not, according to the values of $k$. The strong Shannon effect states that almost all functions have the same ROBDD size as the largest ROBDDs up to a factor of $1 + o(1)$ as $k$ tends to infinity. We observe these facts in Figure 1 by observing the blue curve representing the (probability) distribution of Boolean functions with 12 variables according to their ROBDD size. In fact, as it is generally the case in the context of Boolean functions, asymptotical results can be observed for quite small values of $k$. 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 usually not of exponential size.

In [17], the authors study, experimentally, numerically, and theoretically, ROBDD sizes 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 in $k$ variables, equal to $2^{2^k}$, gives only access to the first values for $k = 1, \ldots, 4$. Then the authors extrapolate the distributions by sampling. Later Clément and Genitrini [4] obtain similar combinatorial results. Thanks to a new approach based on a partial recursive decomposition, they manage to go farther. In fact they partition the ROBDDs according to their profile yielding a much more efficient counting algorithm although using a lot of space due to memoization techniques (storing intermediate results). They obtain exact distributions of the size of ROBDDs up to $k = 9$, thus partitioning the set of $2^{2^{12}}$ Boolean functions into ROBDDs of sizes ranging from 3 to 143.

In this paper, we improve drastically the latter approach so that all the extrapolated results presented in [17] for Boolean functions up to 12 variables are now fully and exactly described. Using a personal computer, in a couple of hours we obtain an exhaustive enumeration of the ROBDDs representing functions up to 12 variables. Indeed we partition the $2^{4096}$ Boolean functions according to their ROBDD size. 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 distribution of functions of 12 variables.
Observing such curves from $k = 1$ to $k = 12$, we notice the exponential growth of the largest ROBDDs when the number $k$ of variables increases. Indeed in Theorem 13 we define $M_k$ to be the size of the largest ROBDDs with $k$ variables. The sequence starts as $(M_k)_{k=1,\ldots,12} = (3, 5, 7, 11, 19, 31, 47, 79, 143, 271, 511, 767)$.

Since each of these values corresponds to the length of the associated distribution support, it is natural to analyze the complexity of the algorithms computing the distribution of the size of ROBDDs with at most $k$ variables according to $M_k$. We note that $M_k$ behaves like $2^k/k$ as $k$ tends to infinity. In this context, we obtain that essentially Newton and Verna’s approach [17] is of order $\Omega(M_k^{4/3})$, i.e. that their algorithm is of near factorial complexity. Clément and Genitrini’s algorithm from [4] is essentially of complexity $\Omega(M_k^{1/2}\log M_k)$. Although they manage to obtain results for bigger $k$, their algorithm is not of polynomial complexity in $M_k$. In this paper, we describe an algorithm that calculates the exact distribution in time complexity $O(M_k^{1/2}\log M_k)$. To our knowledge, our paper gives the first polynomial complexity algorithm computing the distribution of the ROBDD size of Boolean functions.

Our combinatorial approach, as by-products, gives an exhaustive generation algorithm and thus a uniform random sampler for ROBDDs of a given size. Several practical applications derive from these generation algorithms, in particular in the context of random testing of structural and algorithmic program properties. In [7] the authors execute tests for an algorithm taking as entry a ROBDD. It is based on the famous and widely used QuickCheck software [3]. This later aims at designing tests for program properties by using as an entry a random generator and building test cases for test suites. Using a uniform sampler, the goal is to derive statistical testing: since the underlying distribution of the samples is uniform, it allows to extract statistics thanks to the tests like in [10]. Another application is exhaustive testing for small structures, like the studies in [13, 14], that can be driven with QuickCheck or with other tools like Korat for Java. Finally another direction for testing is to bias the uniform complete distribution in order to focus on special corner cases. We are able to uniformly sample in polynomial time and space ROBDDs of a given size, or a given profile. Especially for small sizes of structures, while it is unlikely to sample them using the global uniform sampler, we aim, e.g. at constructing particular ROBDDs for instance sharing the same profile as the one of symmetric functions (known to have small ROBDDs, cf. [11]). Since we do not need to compute the full distribution, we can sample such structures for much larger values of $k$.

The paper is organized as follows: in Section 2 we present the formal notions for the description of our counting and sampling approaches. Section 3 is devoted to the formal description of the counting method to partition Boolean functions according to their ROBDDs size. Finally Section 4 presents the algorithmic context and then describes algorithms to sample ROBDDs.

2. Preliminaries

2.1. Notations. By $\mathbb{N}$, we denote the set of positive integers; $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. For a set or a multiset $A$, the size of $A$, denoted by $|A|$, is its number of elements. The support of a multiset $A$ is the set elements in $A$ (each element appears exactly once). Let $\mathbb{N}_0^*$ be the set of finite sequences of integers. The empty sequence is denoted $[\ ]$. The concatenation product of two sequences $x = [x_0, \ldots, x_k]$ and $y = [y_0, \ldots, y_{k-1}]$ is $x \cdot y = [x_0, \ldots, x_{k-1}, y_0, \ldots, y_{k-1}]$. The empty sequence $[\ ]$ is the neutral element for this product (for any sequence $x$, $x \cdot [\ ] = [\ ] \cdot x = x$). The product order ‘$\preceq$’ on $\mathbb{N}_0^*$ is a non-strict partial order: for two sequences $x = [x_0, \ldots, x_k]$ and $y = [y_0, \ldots, y_k]$ of same length, $x \preceq y$ if and only if $x_i \leq y_i$ for all $i \in \{0, \ldots, k\}$.

Definition (Layered set decomposition). A layered set is a family of pairwise disjoint sets $(Q_i)_{i \in \mathbb{N}_0}$ indexed by $\mathbb{N}_0$. Each set $Q_i$ (possibly empty) for $i \in \mathbb{N}_0$ is called the $i$-th layer.

Let us fix a finite layered set $(Q_i)_{i \in \mathbb{N}_0}$ and denote by $Q = \cup_{i \in \mathbb{N}_0} Q_i$. We define

- The level of an element $q \in \cup_{i \in \mathbb{N}_0} Q_i$, denoted by $\text{level}(q)$, is the layer $i$ of the layer $Q_i$ to which it belongs (it is uniquely defined since the sets are pairwise disjoint).
- The maximal level is $\max\{i \in \mathbb{N}_0 \mid Q_i \neq \emptyset\} = \max_{q \in Q} \{\text{level}(q)\}$.
- The profile is the sequence $p = [p_0, \ldots, p_{k-1}]$ of length $|p| = k + 1$ where $k$ is the maximal level and $p_i = |Q_i|$ for $i \in \{0, \ldots, k\}$. The size of profile $p$, denoted $|p|$, is $\sum_{i=0}^k p_i$. 

Definition 2 (Sequence representation and non-strict partial order for profiles). With a profile \( p = [p_1, \ldots, p_0] \) we associate a sequence \( \sigma(p) \) in \( \mathbb{N}_0^* \) defined by \( \sigma(p) = [k^{|p_1|} \cdot (k-1)^{|p_2|} \cdot \ldots \cdot 1^{|p_1|} \cdot 0^{|p_0|}] \), where \([\alpha]^\beta\) is the sequence of the element \( \alpha \in \mathbb{N}_0 \) repeated \( \beta \) times \( (\beta \in \mathbb{N}_0) \) and \([\alpha]^0 = \{\}\).

The set of profiles is equipped by a non-strict partial order derived from the product order on their sequence representations. Let \( p \) and \( q \) two profiles, we note \( p \preceq q \) if and only if \( \sigma(p) \preceq \sigma(q) \).

Remark. Only profiles of same size may be comparable. For instance let \( 2.2 \) (respectively left and middle).

In Figure 2 a node \( i \) is a level by level. First we introduce multientry \( \sigma \) is similar to the one of \( \text{robdd} \) functions in the same decision diagram.

2.2. Combinatorial description. We introduce a combinatorial description of a ROBDD in layers. This point of view gives a convenient formal characterization of the decomposition of ROBDDs level by level. First we introduce multientry ROBDDs corresponding exactly to removing some upper layers in a usual ROBDD (see Figure 2 for an example).

Definition 3 (Multientry ROBDD). A multientry ROBDD \( B \) with at most \( k > 0 \) variables and size \( n \geq 0 \) is a tuple \( B = ((Q_i)_{i \in \mathbb{N}_0}, \mathfrak{M}, \delta) \) where

- \( (Q_i)_{0 \leq i \leq k} \) is a layered finite set and the set of nodes of \( B \) is \( Q = \bigcup_{i \in \mathbb{N}_0} Q_i \).
- The bottom layer \( Q_0 \) at level 0 satisfies \( Q_0 \subset \{\perp, \top\} \).
- \( \mathfrak{M} \) is a multiset of elements in \( Q \), called the multiset of entries. The layers of \( \mathfrak{M} \) are induced by the layers of \( Q \).
- \( \delta : Q \setminus Q_0 \rightarrow Q \times Q \) is the transition function. Nodes \( \perp \) and \( \top \) are sink nodes. We denote \( \delta_1 \) and \( \delta_0 \) the two coordinate projection maps of \( \delta \) (i.e., \( \delta(q) = (\delta_0(q), \delta_1(q)) \)). For a node \( q \in Q \setminus Q_0 \), \( \delta_0(q) \) and \( \delta_1(q) \) are called respectively the low child and the high child of \( q \).
- A path from \( u \) to \( v \) (\( u, v \in Q \)) is a sequence of nodes \( (u = q_0, \ldots, q_\ell = v) \) such that for all \( i \in \{0, \ldots, \ell - 1\} \) we have either \( q_{i+1} = \delta_0(q_i) \) or \( q_{i+1} = \delta_1(q_i) \). The path is of length \( \ell \) (there is always a path of length 0 from a node to itself).
- There are constraints on \( \delta \) translating that the structure represents a ROBDD:
  (i) for any node \( q \in Q \setminus Q_0 \), \( \delta_0(q) \neq \delta_1(q) \) (useful node property).
  (ii) For any distinct nodes \( q \) and \( q' \) with the same level \( i > 0 \), we have \( \delta(q) \neq \delta(q') \) (descendants unicity in layer).
  (iii) For any node \( q \in Q \setminus Q_0 \), \( \text{level}(q) > \max(\text{level}(\delta_1(q)), \text{level}(\delta_0(q))) \) (acyclicity).
  (iv) For any node \( q \) in \( Q \) there exists an entry \( e \in \mathfrak{M} \) such that there is a path from \( e \) to \( q \) (accessibility).

In the previous definition we allow for a multiset \( \mathfrak{M} \) of entries, but if \( \mathfrak{M} \) is reduced to a single node \( r \) with multiplicity 1 in the top layer, we recover the usual definition of a ROBDD with root \( r \).

In Figure 2 a ROBDD and multientry ROBDD (obtained by removing the 3 top layers) are depicted (respectively left and middle).

Although the definition is general, we could have supposed that the size of a ROBDD is greater or equal to \( n = 3 \). When \( n < 3 \), the multientry ROBDDs are very special cases. When \( n \geq 2 \), the definition implies \( Q_0 = \{\perp, \top\} \) and when \( n \geq 3 \), an important property is true: for any node \( q \) in \( Q \setminus Q_0 \) both sink nodes \( \perp \) and \( \top \) are accessible from \( q \). We note that our definition of multientry ROBDD is similar to the one of shared-BDDs presented by Knuth [11] to represent several Boolean functions in the same decision diagram.

Definition 4 (Spine of a ROBDD, tree and non-tree edges). The spine of a ROBDD \( B \) is the spanning tree obtained by a depth-first search of the ROBDD (where low child is accessed before the high one), and omitting both sinks \( \perp \) and \( \top \). For a ROBDD \( B \), the edges of the spine forms the set of tree edges. The other edges form the set of non-tree edges.

In Figure 2 (right), the tree edges are drawn in black and non-tree edges in light gray.
Theorem 7 (Multientry ROBDDs counting formula). Let us consider the family of linear applications $(\phi_r)_{r \geq 0}$ where each application $\phi_r$ is relative to the family of polynomials $(R_{r,m}(X))_{m \geq 0}$ given by
\[
R_{r,m}(X) = \left( \prod_{i=0}^{r-1} (X^2 - X - i) \right) \cdot \left( \sum_{j=0}^{m-r} \binom{m}{j} \binom{m-j}{r} X^j \right). \tag{1}
\]
Let $M(p, m)$ be the number of multientry ROBDDs with profile $p = [p_k, \ldots, p_1, p_0 = 2]$ and a multiset of entries $\mathcal{E}$ with profile $e \leq m = [m_k, \ldots, m_0]$. We have for $k > 0$

$$M(p, m) = G_{p, m}(2),$$

(2)

where $G_{p, m}(X)$ is the polynomial given by

$$G_{p, m}(X) = X^{p_0} \phi_{p_1}(X^{p_1} \phi_{p_2}(\ldots \phi_{p_{k-1}}(X^{m_{k-1}} \phi_{p_k}(X^{m_k})) \ldots)).$$

(3)

In the theorem, $\binom{m}{k}$ stands for the binomial coefficient and $\binom{n}{k}$ 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 8.** This is actually a stronger result that what we need for counting ROBDDs, since the number of ROBDDs corresponds to the special case where $p = [p_k = 1, p_{k-1}, \ldots, p_1, p_0 = 2]$ and $m = [1] \cdot [0]^k$ (meaning there is one entry mapped to the root of the ROBDD in layer $k$).

The detailed proof of Theorem 7 is presented in Appendix A.2. Here we present an example of such a computation.

**Example.** Let $p = [1, 2, 4, 2, 2]$ and $m = [1]\cdot[0]^4$, then we compute $Q_i(X) = \phi_{p_i}(\phi_{p_{i+1}}(\ldots \phi_{p_k}(X) \ldots))$ for $1 \leq i \leq 4$:

- $X \overset{\phi_1}{\rightarrow} X^2 - X$
- $X^2 \overset{\phi_2}{\rightarrow} X^4 - 2X^3 + X$
- $X^4 \overset{\phi_3}{\rightarrow} X^8 - 4X^7 + 14X^5 - 6X^4 - 16X^3 + 5X^2 + 6X$
- $X^8 \overset{\phi_4}{\rightarrow} 28X^{10} + 28X^9 - 98X^8 - 112X^7 + 76X^6 + 92X^5 - 12X^4 - 8X^3 + 6X^2$.

Evaluating the last polynomial at $X = 2$, we get that there are 11160 ROBDDs with profile $[1, 2, 4, 2, 2]$.

### 4. Counting and sampling algorithms

In this section, we describe the algorithms that count and sample ROBDDs. The time and space complexities are measured respectively in terms of arithmetical operations on $\mathbb{Z}$ and memory space used to store integers in $\mathbb{Z}$. However recall that for $k$ variables, integers can be of order $2^{2^k}$ (the number of Boolean functions with $k$ variables when the size is not constrained).

The reader can find an implementation of the following algorithms in this Github repository.

In the following we suppose that each polynomial $R_{r, m}$ (of degree $m + r$, presented in Theorem 7) is available thanks to a precomputation step, and we do not count the memory space used to store these polynomials as it is negligible.

#### 4.1. Counting algorithms.

**4.1.1. Basic counting.** The basic block in our approach is to be able to compute $\phi_r(P)$ for $r \in \mathbb{N}_0$. This is done by Algorithm 1.

**Algorithm 1: iterate($r, P$)**

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

**Algorithm 2: count($p = [p_k, \ldots, p_0]$)**

- **Input:** profile $p = [p_k, \ldots, p_0]$
- **Output:** the number of multientry ROBDDs with profile $p$
- $P \leftarrow X$
- for $i$ from $k$ downto 1 do
  - $P \leftarrow \text{iterate}(p_i, P)$ \hspace{1cm} \triangleright computation of $\phi_{p_i}(P)$
- return $P(2)$

**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.\(^1\)

\(^1\)Recall in Proposition 6 we do not take into account the precomputation step of the family $R_{r, m}$. 

Proof. Each polynomial $R_{r,m}(X)$ 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 $\phi_0$ is the identity). \qed

More generally in the next proposition, we give the complexity of computing $M(p, m)$ of Theorem 7.

**Proposition 10** (Computing $M(\cdot, \cdot)$). The computation of $M(p, m)$ (see Theorem 7) for $|p| = |m| = k + 1$ and size $|p| = n$ can be performed with $O(k n^2)$ arithmetical operations over $\mathbb{Z}$ using $O(n)$ space.

**Proof.** Let $p = [p_k, \ldots, p_0]$ and $m = [m_k, \ldots, m_0]$. The quantity $G_{p,m}(X)$ in [3] in Theorem 7 is computed by iterating $k$ times a function of type $\phi_r$, shifting the resulting polynomial according to $m$ and starting from an initial polynomial $X^{m_k}$. By Proposition 9 the algorithm performs $O(n^2)$ operations for each iteration since the degree of last polynomial is $n + 1$. The total computation thus performs $O(kn^2)$ arithmetical operations over $\mathbb{Z}$ and use $O(n)$ memory space to store coefficients. Evaluating at $X = 2$ can be done in time complexity $O(n)$ (by Horner’s method for instance). \qed

As a corollary, by Remark 8 and applying Theorem 7 it is straightforward to compute the number of ROBDDs with profile $p = [p_k, \ldots, p_0]$ (implying implicitly $p_k = 1$, $p_0 = 2$ and $m = [1] \cdot [0]^k$ ) The pseudo-code is given in Algorithm 2.

**Corollary 11.** Algorithm [3] computes the number of ROBDDs of size $n$ with $k$ variables and given profile $p$ and performs $O(k n^2)$ arithmetical operations over $\mathbb{Z}$ and uses $O(n)$ space.

### 4.1.2. Counting up to some size.

The main goal of this section is to compute the distribution of the Boolean functions in at most $k$ variables according to the ROBDD size. In the following we use the linearity of applications $\langle \phi_r \rangle_{r \geq 0}$, for an integer $k \geq 1$ we efficiently estimate the sequence $(N_i)_{0 \leq i \leq N}$ where $N_i$ is the number of ROBDDs of sizes $i + 2$, representing a Boolean function in at most $k$ variables. We first present an algebraic point of view for counting, and then exhibit its algorithmic counterpart.

We introduce a formal variable $u$ accumulating the number of nodes in each layer except the bottom one ($u$ does not count the two constant-nodes of the bottom layer $Q_0$). Let us define as $Z$ the subset of $\mathbb{Z}[u, X]$ as

$$Z = \{ P = \sum_{i=0}^{d} u^i Q_i(X) \in \mathbb{Z}[u, X] \mid d \geq 0, \text{ and for } 0 \leq i \leq d, Q_i \in \mathbb{Z}[X] \text{ with } \deg(X)(Q_i) \leq i + 1 \}.$$  

We formally define the linear application $\varphi : Z \to Z$

$$\varphi : u^r X^m \mapsto \sum_{i=0}^{m+1} u^{r+i} \phi_i(X^m).$$

Then we are able to compute the generating function of all ROBDDs with at most $k$ variables

$$H_k(u) = \sum_{i \geq 0} N_i u^i = G_k(u, 2) \quad \text{where } G_k(u, X) = \varphi \circ \cdots \circ \varphi(X).$$

**Example.** Let us consider $H_k(u, X)$ of [4], we get

- $H_1(u, X) = u^2 \varphi(X) = \phi_0(X) + u \phi(X) = (X^2 - X)u + X$. We can verify that $H_1(u, 2) = 2 + 2$, thus there are 2 ROBDDs with one variable and size 1 + 2 = 3 (since we have 2 nodes for the bottom layer $Q_0$ containing the 2 constants and one root). The constant term in $H_1(u)$ corresponds to the fact that there are always 2 ROBDDs with no variable (the constants).

- Adding a second layer, we get $H_2(u, X) = \varphi(H_1(u, X))$, i.e.,

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

We get $H_2(u, 2) = 2 u^3 + 8 u^2 + 4 u + 2$, hence there are 2, 8, 4 ROBDDs of respective sizes 5, 4, 3 with at most 2 variables.
• Iterating with a third layer, we get $H_3(u, X) = \overline{\varphi}(H_2(u, X))$ which has 34 terms and gives
  
  \[H_3(u, 2) = 74u^5 + 88u^4 + 62u^3 + 24u^2 + 6u + 2.\]

  Hence there are respectively 74, 88, 62, 24 and 6 ROBDDs of size 7, 6, 5, 4 and 3.

• Adding a fourth layer yields for $H_4(u, X) = \overline{\varphi}(H_3(u, X))$ a polynomial with 134 terms and
  
  \[H_4(u, 2) = 11160u^9 + 23280u^8 + 17666u^7 + 8928u^6 + 3248u^5 + 960u^4 + 236u^3 + 48u^2 + 8u + 2.\]

  There are 11160 ROBDDs with 4 variables of size 11, 23280 ROBDDs of size 10, etc.

This is a rather algebraic point of view, and Algorithm 3 that follows, uses sequences of polynomials $[P_0, \ldots, P_n]$ to represent a polynomial in $\mathbb{Z}$ of the form $G(u, X) = \sum_{i=0}^{n} u^i P_i(X)$ (where $\deg_X(P_i) \leq i + 1$).

**Theorem 12** (Algorithm for bounded size distribution). Algorithm 3 returns a sequence $(N_i)_{i=0}^{n-2}$ where $N_i$ is the number of ROBDDs of size $i + 2$ with at most $k$ variables. It performs $O(k n^4)$ arithmetical operations in $\mathbb{Z}$ and uses $O(n^2)$ space.

**Proof.** First we note that we can truncate polynomials and keep only monomials of degree less or equal to $n$ for $u$ and less or equal to $n + 1$ for $X$ when we restrict the counting only to a bounded size $n$. This is crucial for complexity matters as for $k$ variables, the maximal size of a ROBDD is exponential $O(2^k/k)$ (as we will see in the next paragraph). Secondly, the algorithm uses two lists of polynomials of length $O(n)$ where polynomials are degree $O(n)$ yielding $O(n^2)$ coefficients in $\mathbb{Z}$. For each layer we have $O(n^2)$ calls to iterate which itself has complexity $O(n^2)$. Overall the algorithm performs $O(k n^4)$ arithmetical operations over $\mathbb{Z}$ and uses $O(n^2)$ space. \(\square\)

---

**Algorithm 3**: computeSizeDistribution($n, k$)

**Input**: Maximal values for the number of variables, $k$, and size, $n$

**Output**: Sequence $N = [N_0, \ldots, N_{n-2}]$ where $N_i$ is the number of ROBDDs of size $i + 2$

with at most $k$ variables

\[
T \leftarrow 0
\]

previous $\leftarrow [X]$ \quad ▶ starting from $X^1u^0$

for $t$ from $k$ downto 1 do

  current $\leftarrow$ array(0..min($2(T + 1), n - 2$))

  for $s$ from 0 to $T$ do

    for $r$ from 0 to $\min(s + 1, n - s - \lceil \sqrt{n - t} \rceil)$ do

      current[$s + r$] $\leftarrow$ current[$s + r$] + iterate($r$, previous[$s$])

    previous $\leftarrow$ current

  $T \leftarrow$ min($2T + 1, n - 2$)

$N \leftarrow []$

for $P$ in previous do

  $N \leftarrow N \cdot [P(2)]$

return $N$

**Remark.** The algorithm is easily tailored to count essential ROBDDs, i.e. where all layers must be non-empty: it suffices that $r$ starts from 1 instead of 0 within the inner loop.

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

**Theorem 13** (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

\[M_k = 2^{k - \theta} - 1 + 2^{\log_2 k}, \quad \text{with } \theta = \lceil \log_2 (k - \lfloor \log_2 (k) \rfloor) \rceil.\]

Note that this formula is equivalent to the one given without proof by Pontus von Brömssen [18]. The existence of $\theta$ is proved in [17] and from there we can derive the explicit expression of $\theta$.

Then we just have to substitute $n = M_k$ in Algorithm 3 to compute the complete size distribution. The same substitution in [12] yields the following corollary.
Corollary 14 (Algorithm for complete size distribution). Algorithm 3 returns the complete distribution of Boolean functions with at most $k$ variables according to the ROBDD size. The algorithm performs $O(2^{4k}/k^3) = O(M_k^4 \cdot \log M_k)$ arithmetical operations in $\mathbb{Z}$ and uses $O(M_k^2)$ space.

4.2. Sampling algorithms. The ranking/unranking techniques for objects of a combinatorial class $C$ of size $N$ consists in building a bijection between any $c \in C$ and an integer (its rank) in the interval $[0..N−1]$. This leads trivially to a uniform sampling algorithm by drawing uniformly first a rank and then building the corresponding object.

We first present the overall pseudocode for unranking a ROBDD with a fixed size and number of variables. Internally the ROBDDs and spines are represented by sequences of nodes with a layer attribute, and a dictionary representing the transition function. In the following we request the classical traversals for trees. The formal definitions are presented in Appendix A.3.

Algorithm 4: unrank($r, n, k$)

Input: Rank $r$, size $n$, (max) number of variables $k$

Output: the $r$-th ROBDD: a list of nodes in reverse postorder (so that root is the first element) and transition function $\delta$

1. $p, r^* \leftarrow \text{unrankProfile}(r, n, k)$ ▷ Step 1
2. $L, \delta, D \leftarrow \text{unrankWithProfile}(r^*, n, k, p)$ ▷ Step 2
3. $L, \delta \leftarrow \text{adjustNonTreeEdges}(L, \delta, D)$ ▷ Step 3

return $L, \delta$ ▷ list of nodes $L$ and the transition function $\delta$

Our unranking algorithm consists in three phases:

Step 1. Given a rank $R$, the unranking algorithm will first select the profile. This is done by iteratively computing the number of nodes in each layer, starting from layer $k$ down to layer 1 (see Algorithm 5).

Step 2. Once the profile is fixed, the edge structure (tree edges and non tree edges) of the ROBDD is built in a reverse postorder process. The goal is to build the spine (i.e., the tree edges of the ROBDD) and, along the way, to gather information (a relative rank) for non tree edges (see Algorithm 6).

Step 3. Finally an inorder traversal of the spine is needed, such that, for every non tree edge, we identify its absolute destination, that is a precise node. We use the fact that with an inorder traversal, we are able to identify the set of nodes, called pool, to which a non tree edge can point knowing only a rank within this set (see Algorithm 7).

Theorem 15 (Unranking algorithm for ROBDDS). The unranking algorithm 4 for a ROBDD with profile $p$ of size $n$ with at most $k$ variables has

- $O(k^2 n^5)$ time complexity and uses $O(n^2)$ extra space for identifying the profile;
- $O(k^2 n^3)$ time complexity to generate the ROBDD with fixed profile $p$.

The latter theorem is a consequence of the results presented in Propositions 16, 17 and 18.

Remark. By using memoization techniques, we could generate more efficiently (in time) several ROBDDS with the same characteristics (same size, or same profile). Like when we were counting, the unranking method can be transposed to focus on essential ROBDDS.

In the following we present some more details about each step. The complete descriptions and algorithms are presented in Appendix A.3.

Step 1: identifying the profile. To determine the profile we need an auxiliary function computing the number $M'(n, k, p)$, corresponding to the number of ROBDDS of size $n$ and with $k$ variables when the profile prefix is $p$. This is an adaptation of computeSizeDistribution algorithm and not detailed in this paper.

Proposition 16 (Unranking algorithm for profile). Algorithm 5 for identifying the profile given the rank $r$ among ROBDDS of size $n$ and with at most $k$ variables has a time complexity $O(k^2 n^5)$ and uses $O(n^2)$ memory space.
An simple adaptation of computeSizeDistribution algorithm associated with a binary search for the calculation of \( m^\ast \) in Algorithm 5 gives this result. The proof is given in Appendix A.3.

Step 2: spine building process in reverse postorder. The fundamental idea is to extend the leftmost branch of the current spine by adding a node to a free half edge. This is done by considering for each possible extension the number of ROBDDs which can be grafted on this new partial spine (using Proposition 27). The rank is updated for each configuration until the correct one is reached. A detailed pseudocode is given in appendix (see Algorithm 6).

**Proposition 17** (First pass of unranking with fixed profile). Algorithm 4 has \( O(k^2 \ n^3) \) time complexity. The extra space used is \( O(n^2 + kn) \).

Step 3: inorder traversal for identifying non tree edges. This last step consists in doing an inorder traversal of the spine to set up the non tree edges thanks to the rank information gathered in Step 2. Indeed when encountered along the inorder traversal of the spine, a node may have one or two (if external) non tree outgoing edges. In each case we know the rank of either the destination node or the pair of destination nodes within the pool, the set of non nodes we can point to. In the case of an external node which has two outgoing half edges to the pool, we must ensure that the pair of destinations is distinct from the ones of its siblings in the same layer. A detailed pseudocode is given in appendix (see Algorithm 7).

**Proposition 18** (second pass of unranking with fixed profile: adjusting non tree edges). Algorithm 7 has \( O(n^2 + kn) \) time complexity and uses \( O(n) \) extra space.

5. Conclusion

Our new counting approach, computing the distribution of Boolean functions according to their ROBDDs sizes, does not need a complete enumeration of either the spines or the profiles, and thus allows us to get an algorithm that is polynomial time complexity in \( M_k \), compared to the previous one presented in [1]. Using this counting method we then are able to draw efficiently and uniformly at random ROBDDs either according to a given size, or 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 [17], 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 = 122 is approximately \( 1.212 \times 10^{-957} \). In practice, several classical functions have ROBDDs of small sizes. For example the symmetrical functions in \( k \) variables are associated with ROBDDs of quadratic size in \( k \).

Our approach can be extended obviously in the context of OBDDs, where each function does not have a canonical representative. There we have just to remove the constraints (i) and (ii) of Definition 3 since the compression rules are relaxed. An interesting future work consists in enumerating the BDD structures where our counting and sampling methods can be applied.

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 [4]. It
would be interesting to see if the machinery of analytic combinatorics 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] K. Claessen and J. Hughes. Quickcheck: A lightweight tool for random testing of haskell programs. SIGPLAN Not., 35(9):268–279, 2000.
[4] 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.
[5] A. Darwiche and P. Marquis. A knowledge compilation map. J. Artif. Int. Res., 17(1):229–264, sep 2002.
[6] 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.
[7] P. Dybjer, Q. Haiyan, and M. Takeyama. Verifying Haskell Programs by Combining Testing and Proving. In QSC’03, pages 272–279, 2003.
[8] P. Flajolet and R. Sedgewick. Analytic Combinatorics. Cambridge University Press, New York, NY, USA, 1 edition, 2009.
[9] C. Grüll, H. J. Prömel, and A. Srivastav. Ordered binary decision diagrams and the shannon effect. Discret. Appl. Math., 142(1-3):67–85, 2004.
[10] R. Grosu and S. A. Smolka. Monte carlo model checking. In 11th International Conference TACAS, pages 271–286, 2005.
[11] D. E. Knuth. The Art of Computer Programming, Volume 4A, Combinatorial Algorithms. Addison-Wesley Professional, 2011.
[12] 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.
[13] D. Marinov, A. Andoni, D. Daniluc, S. Khurshid, and M. Rinard. An evaluation of exhaustive testing for data structures. Technical report, MIT -LCS-TR-921, 2003.
[14] A. Milicevic, S. Misailovic, D. Marinov, and S. Khurshid. Koorat: A tool for generating structurally complex test inputs. In 29th International Conference on Software Engineering (ICSE 2007), pages 771–774. IEEE Computer Society, 2007.
[15] S.-I. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. 30th ACM/IEEE Design Automation Conference, pages 272–277, 1993.
[16] C. Mues, B. Baesens, C. M. Files, and J. Vanthielen. Decision diagrams in machine learning: an empirical study on real-life credit-risk data. Expert Systems with Applications, 27(2):257–264, 2004.
[17] 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.
[18] N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences, Sep 2019. Sequence A327461.
[19] 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.
[20] I. Wegener. Branching Programs and Binary Decision Diagrams. SIAM, 2000.
A.1. Appendix related to Section 2

Proof of Proposition 5 (Profiles of ROBDDs). By definition, we have $p_0 = 2$ for constants of the bottom layer. Then the profile $p = [p_k, \ldots, p_1]$ is a valid profile for a binary tree (meaning each node is connected to either two nodes or zero) if and only for all $1 \leq \ell \leq k$

$$p_\ell \leq P_{>\ell} + 1.$$  

(5)

This follows from a classic property of binary trees, which states that a binary tree with $n$ nodes has $n + 1$ out-going available half-edges. Hence we can add up to $P_{>\ell} + 1$ nodes on level $\ell$ and $p_\ell \leq P_{>\ell} + 1$. So by induction we have $\Box$.

Secondly, for any level $1 \leq \ell \leq k$ we get a connected structure as soon as

$$0 \leq p_\ell \leq P_{<\ell}(P_{<\ell} - 1).$$

(6)

This inequality entails that there are sufficiently many pairs of nodes available to be valid children for all nodes at level $\ell$. We denote $B = P_{<\ell}$ and $q = p_\ell$. Thus Equation (6) translates to $0 \leq q \leq (B - q)(B - q - 1)$. Solving the quadratic inequation $(B - q)(B - q - 1) - q \geq 0$ and, noting $q$ is an integer, that is equivalent to $0 \leq q \leq B - \lceil \sqrt{B} \rceil$. Thus we conclude Equation (6) is equivalent to $0 \leq p_\ell \leq P_{<\ell} - \lceil \sqrt{P_{<\ell}} \rceil$.

\[ \square \]

A.2. Appendix related to Section 3

Proof of Theorem 7 (Multientry ROBDDs counting formula). We will exploit the following properties related to the family $(R_{r,m})_{r,m \geq 0}$:

- $\phi_0$ is the identity (since $P_{0,m}(X) = X^m$ for $m \geq 0$), implying by linearity that $\phi_0[P] = P$ for any polynomial $P$.
- when $r \geq 0$, for any polynomial $P$ and any constant $c \in \mathbb{Z}$ we have $\phi_r[P + c] = \phi_r[P]$.

Indeed, if $r \geq 0$, $R_{r,m}(1) = R_{r,m}(0) = 0$ for any $m \geq 0$ since by definition $X^2 - X = X(X - 1)$ divides $R_{r,m}(X)$. So for any constant $c \in \mathbb{Z}$, $\phi_r[c] = 0$ whenever $r \geq 0$ and by linearity, we get the result.

The proof is given by induction on $(k + 1)$, the number of layers for $k \geq 0$.

Base case. When $k = 0$ and $n \geq 2$, if the entry multiset is non-empty, i.e., $m > 0$. The number of ROBDDs is $M([2],[m]) = 2^m$ as we must map the $m$ entries to either one of the two constants: we get $G_{[2],[m]}(X) = X^m$. If the entry multiset is empty then $M([2],[m]) = 1$ corresponding to the void function).

Induction step. Now suppose Theorem 7 is true for $k \geq 0$.

Let us decompose $p = \{p_k, p_{k-1}, \ldots, p_0\}$ and $m = \{m_k, m_{k-1}, \ldots, m_0\}$ (with $k \geq 0$) as $p = [r] \cdot p'$ and $m = [m, z] \cdot m'$ with $r = p_k + 1$, $m = m_k + 1$ and $z = m_k$.

If $r = 0$, the empty layer can be omitted and we have

$$M([r] \cdot p', [m, z] \cdot m') = M(p', [m + z] \cdot m').$$

(7)

Since $\phi_0$ is the identity, the theorem is true by induction hypothesis in this case (we have $P_{0,m}(X) = X^m$). Note also that we must have $m \geq r$ as all the $r$ nodes of $Q_{k+1}$ must be covered by at least one entry (otherwise $M(p, m) = 0$).

From now on let us suppose $0 < r \leq m$. The multiset of $m$ possible entries at layer $k + 1$ can be decomposed in two subsets for $j \in \{0, \ldots, m - r\}$: $j$ entries will go to some layer below (of level strictly less than $k$), and $m - j$ entries will be mapped to the $r$ nodes of the layer $Q_k$. A Stirling number of the second kind $
 choose k$ counts the number of ways to partition a set of $n$ objects into $k$ non-empty subsets. So there are $\binom{m}{r}$ such partitions. We write

$$M([r] \cdot p', [m, z] \cdot m') = \sum_{j=0}^{m-r} \binom{m}{j} \binom{m-j}{r} M([r] \cdot p', [r, z+j] \cdot m').$$

(8)
On the other hand, since \( r > 0 \) for \( m = r \) and \( z > 0 \) we can write

\[
M([r] \cdot p', [r, z] \cdot m') = M([r - 1] \cdot p', [r - 1, z + 2] \cdot m') \\
- M([r - 1] \cdot p', [r - 1, z + 1] \cdot m') \\
- (r - 1)M([r - 1] \cdot p', [r - 1, z] \cdot m').
\]

(9)

Indeed Equation (9) translates that removing \( M \) entries, we must however respect additional constraints entries (corresponding to the children of the removed node) strictly below layer \( k \). Adding these entries, we must however respect additional constraints

1. We must prevent the two entries to be equal (property (i) of usefulness of Definition 3);
2. We must also prevent the two entries to duplicate a pair of children corresponding to the \((r - 1)\) other nodes at layer \( k \) (property (ii) of distinctness of Definition 3).

So to take care of the first constraint in (9), we follow the inclusion-exclusion principle and subtract \( M([r - 1] \cdot p', [r - 1, z + 1] \cdot m') \) which would count the number of ROBDDs if two entries were “merged” at layer of level less than \((k - 1)\). Formally it is equivalent to remove a node at layer \( k \) and add only one entry of lesser level. As for the second constraint we subtract \((r - 1)M([r - 1] \cdot p', [z, r - 1] \cdot m')\) in (9). This quantity counts the number of ROBDDs when merging two adjacent nodes among the \( r \) nodes at layer \( k \) (hence \( r - 1 \) choices) and adding no other entry below. Finally (9) is proved.

By induction, using (7) and (9) we prove

\[
M([r] \cdot p', [r, z] \cdot m') = \sum_{i=0}^{2r} a_i M(p', [z + i] \cdot m'),
\]

(10)

where coefficients \((a_i)\) are obtained by identifying \( P(X) = \prod_{i=0}^{r-1} (X^2 - X - i) = \sum_{i=0}^{2r} a_i X^i \). So (9) rewrites

\[
M([r] \cdot p', [z, r] \cdot m') = \sum_{i=0}^{2} b_i M([r - 1] \cdot p', [r - 1, z + i] \cdot m'),
\]

(11)

where the coefficients \(b_i\) are defined by identifying \( \sum_{i=0}^{2} b_i X^i = X^2 - X - (r - 1) \). By induction hypothesis, for any \( z \geq 0 \),

\[
M([r - 1] \cdot p', [r - 1, z] \cdot m') = \sum_{i=0}^{2r - 2} c_i M(p', [z + i] \cdot m'),
\]

(12)

where the coefficients \(c_i\) are defined by identifying \( \sum_{i=0}^{2r - 2} c_i X^i = \prod_{i=0}^{-r} (X^2 - X - i) \). Plugging (11) in (11), after elementary but cumbersome computation we get (10).

Finally we can plug (10) into (8) to get

\[
M([r] \cdot p', [m, z] \cdot m') = \sum_{j=0}^{m-r} \binom{m}{j} \binom{m-j}{r} \sum_{i=0}^{2r} a_i M(p', [z + j + i] \cdot m') \\
= \sum_{i=0}^{m+r} c_i M(p', [z + i] \cdot m'),
\]

(13)

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

By induction hypothesis, for any \( q, e \) of length \( k \), we have \( M(q, e) = G_{q,e}(2) \), where \( G_{q,e}(X) \) is defined in (4). So rewriting (13) we get

\[
M([r] \cdot p', [m, z] \cdot m') = \sum_{i=0}^{m+r} c_i G_{p', [z+i] \cdot m'}(2).
\]
By linearity of the applications \((\phi_r)_{r \in \mathbb{N}_0}\)
\[
\sum_{i=0}^{m+r} c_i G_{p', [x+i]} m'(X) = \sum_{i=0}^{m+r} c_i X^{m_0} \phi_{p_1} (X^{m_1} \phi_{p_2} (\ldots \phi_{p_{k-1}} (X^{z+i}) \ldots))
= X^{m_0} \phi_{p_1} (X^{m_1} \phi_{p_2} (\ldots \phi_{p_{k-1}} (X^{z+i}) \ldots))
= X^{m_0} \phi_{p_1} (X^{m_1} \phi_{p_2} (\ldots \phi_{p_{k-1}} (X^z R_{r,m}(X)) \ldots))
= X^{m_0} \phi_{p_1} (X^{m_1} \phi_{p_2} (\ldots \phi_{p_{k-1}} (X^z \phi_r(X^m)) \ldots)).
\]
Hence we get \(2\) for \(p = [r] \cdot p'\) and \(m = [m, z] \cdot m'\). This ends the proof.

A.3. Appendix related to Section 4. To traverse a tree, there are different recursive strategies that we recall here:

Definition 19 (Binary tree traversals: inorder, preorder and postorder). For a binary tree, we recall the three main methods for traversing trees resulting into three orders on nodes of a tree.

Let us consider a binary tree \(T\) with root \(r\) and left subtree \(T_0\) and right subtree \(T_1\).

- Preorder traversal: (i) visit the root; (ii) traverse the left subtree \(T_0\); (iii) visit the right subtree \(T_1\).
- Inorder traversal: (i) traverse the left subtree \(T_0\); (ii) visit the root; (iii) visit the right subtree \(T_1\).
- Postorder traversal: (i) traverse the left subtree \(T_0\); (ii) visit the right subtree \(T_1\); (iii) visit the root.

We also define the reverse postorder traversal: (i) visit the root; (ii) visit the right subtree \(T_1\); (iii) traverse the left subtree \(T_0\).

Given a tree \(T\), once a traversal is fixed, the (total) order on the nodes of \(T\) is induced by the traversal: for two nodes \(q\) and \(q'\) in \(T\), \(q < q'\) if and only if \(q\) is visited before \(q'\).

On Figure 2 the spine of the ROBDD on the left is drawn on the right. Nodes are labeled according to the reverse postorder (of the spine).

A.3.1. Step 1: identifying the profile. The proof of the complexity of Step 1 is given in the following proof.

Proof of Proposition 16 (Unranking algorithm for profile). The algorithm needs to compute the number \(M'(n, k, p)\) of ROBDDs of size \(n\) and with \(k\) variables when the profile starts with \(p\). This is a slight adaptation of algorithm \texttt{computeSizeDistribution} which performs in \(O(k n^4)\) arithmetical operations over \(\mathbb{Z}\) and uses \(O(n^2)\) memory space. Identifying the number of nodes \(m\) in the current layer can be done with the help of a binary search procedure. So unranking the profile of a ROBDD is done in time complexity \(O(k^2 n^5)\) performing a binary search for each of the \(k\) layers.

A.3.2. Step 2: spine building process in reverse postorder. First to identify the possible destination of a non tree half edge, we define the pool of a node.

Definition 20 (Pool of a node). The pool \(\text{pool}(q)\) of a node \(q\) with layer \(i\) is the union of \(\{\bot, \top\}\) and the set of nodes with layer \(0 \leq j < i\) which are visited before \(q\) in the inorder traversal of the spine.

When iteratively adding nodes we obtain a sequence of partial spines of increasing sizes.

Definition 21 (Partial spine). Let us consider a spine \(S\) with node set \(\{q_0, \ldots, q_{n-3}\}\) of a ROBDD of size \(n\) (so that the spine has \(n - 2\) nodes) and where \(q_i\) is the \(i\)-th node \((0 \leq i \leq n - 3)\) in reverse postorder on \(S\). Let us denote \(S_i\) the spine induced by the \(i + 1\) first nodes \(\{q_0, \ldots, q_i\}\) for \(0 \leq i \leq n - 3\) so that we have the sequence of partial spines \(S_{-1} = \emptyset \subset S_0 \subset \cdots \subset S_{n-3} = S\).

We also define the leftmost branch of a partial spine.

Definition 22 (Leftmost branch). The leftmost branch of a tree \(T\) is the path formed by the nodes visited starting from the root and going to the first external node in the inorder traversal.
Finally, we define free half edges which are half edges to which we can attach new nodes in order to make the partial spine grow.

**Definition 23** (Free and frozen half edges). A half edge of a partial spine $S$ is called free if and only if it is a non tree edge of $S$ and it departs from a node of the leftmost branch of $S$. Otherwise we say the half edge is frozen.

Intuitively, free half edges give rise to the entry multiset on which we can graft a multientry ROBDD.

We have the following structural property on the set of free half edges at a given step of the reverse postorder traversal of the spine. The precise knowledge of the structure of free half edges is important for the unranking algorithm we present next.

**Lemma 24.** Let $S$ be a spine of size $n$, and $[q_0, \ldots, q_{n-1}]$ the sequence of nodes in reverse postorder. For an induced spine $S_i$ with support $\{q_0, \ldots, q_i\}$, all free half edges of nodes in set $\{q_0, \ldots, q_{i-1}\}$ are children of nodes. The last node $q_i$ of the leftmost branch has two outgoing free half edges.

Next, given a target profile, we need to be able to count how many multientry ROBDDs can be grafted on a partial spine to achieve that target profile. This is needed to decide if a node has to be created or not in Algorithm 6. We recall the quantity $M(p, m)$ is defined in Theorem 7 that counts the number of multientry ROBDDs with profile $p$ and a multiset of entries $E$ with profile $e < m$ (i.e., is a multiset of entries compatible with $m$). The following lemma is an application of Theorem 7.

**Lemma 25** (Graftings to a spine). Let $S$ be a spine of an ROBDD of size $n$ with profile $p$ and $S'$ be a partial spine with profile $p'$. The partial spine $S'$ has leftmost branch $L$ with an lowest extremity $q$ of level $m$. Let $h$ the number of free half edges in $L \setminus \{q\}$: we denote $\ell = [\ell_0 - 1, \ldots, \ell_h - 1]$ where $\ell_j$ is the level of the source of the $j$-th, sorted by decreasing level, free half edge in $L \setminus \{q\}$. Let also $p' = $profile$(S \setminus S') = p - p'$ and $s = p'_m$. The number of ROBDDs which share the partial spine $S'$ is

$$M_s(p', \ell \cdot [m - 1, m - 1]) - M_s(p', \ell \cdot [m - 1]) - s \cdot M_s(p', \ell),$$

where $M_s(p, \ell) = M(p, \sigma^{-1}(\ell))$ is a notation using a sequence representation of a profile $\ell$ instead of the profile itself.

We give the main ideas for what constitutes the first pass for building the spine structure and compute rank information for non tree half edges in the following. Each time we attach a new node to a free half edge of the partial spine $S$, the leftmost branch is modified and some free half edges may go to a frozen state. Freezing a half edge means either that the half edge becomes a free half edge or that we fix the rank of the node it points to (in the pool). There are several possibilities when adding a new node $q'$:

- If the new node is the child of the last node $q$ of the leftmost branch, $q$ is external (by lemma 24). We have two cases: (i) $q'$ is added as a high kid of $q$: $q'$ is the new extremity of the leftmost branch; (ii) $q'$ is a low kid of $q$ and the high half edge becomes frozen: there are $w - 1$ possibilities for high kid (where $w$ is the size of pool($q$)) since high kid must be different from $q'$.
- Otherwise a new node $q'$ is added as the the left kid of a node $q$ of the current leftmost branch. Then all free half edges departing nodes below $q$ in the leftmost branch freeze. Then the number of possibilities for freezing a half edge depends on which node $u$ is attached to. Let us first remark that since $u$ is on the current leftmost branch, $w = |\text{pool}(u)|$ can be readily deduced from the target profile $p$ and the profile of the current partial spine.
  - If $u$ is the extremity of the leftmost branch which must choose a pair of distinct destinations within pool($u$). Moreover the chosen pair must be distinct from the pairs attached to the siblings of $u$ in the same layer. Hence there are $w(w - 1) - s$ possibilities where $s = p_u - |S|_e \ell$ is the number of nodes at the same level, not yet in the partial spine $S$ ($|S|_e$ is the number of nodes in layer $\ell$ of $S$ and recall $p = [p_0, \ldots, p_0]$ is the target profile).
– Otherwise, the free half edge is a left half edge of some node \( u \) and the number of possibilities to freeze it is \( w = |\text{pool}(u)| \). Again \( w \) is known since \( u \) is on the current leftmost branch.

A detailed pseudocode is given in Algorithm 6.

\textbf{Algorithm 6:} \textup{unrankIncomplete}(r, n, k, p)

\begin{itemize}
\item [\textbf{Input:}] Rank \( r \), size \( n \), (max) number of variables \( k \), target profile \( p \)
\item [\textbf{Output:}] List of nodes in reverse postorder (so that root is the first element) and transition function \( \delta \)
\item \( L \leftarrow \text{empty list } [\] \) \hspace{1em} \triangleright \text{nodes of the spine/mosso in reverse postorder traversal}
\item \( S \leftarrow \text{empty stack} \) \hspace{1em} \triangleright \text{half edges: } \text{pairs } (q, \alpha) \text{ with } q \text{ a node and } \alpha \in \{0, 1\}
\item \( D \leftarrow \text{empty stack} \) \hspace{1em} \triangleright \text{ } D \text{ contains rank information for the second pass}
\item \( \mathcal{E} \leftarrow \{k\} \) \hspace{1em} \triangleright \text{initial multiset of entry levels}
\end{itemize}

\begin{algorithmic}
\While {\card{\mathcal{E}} > 0}
\State \( \ell \leftarrow \mathcal{E}.\text{popMin()} \)
\While {\( p_j = 0 \)} \State \( \ell \leftarrow \ell - 1 \)
\If {\( \ell > 0 \)} \State \( p' \leftarrow p \setminus 1_\ell \)
\State \( \Delta \leftarrow M(p', \mathcal{E} \cup \{\ell - 1, \ell - 1\}) - M(p', \mathcal{E} \cup \{\ell - 1\}) - (p_j - 1) \cdot M(p', \mathcal{E}) \)
\EndIf
\If {\( r - \Delta < 0 \)} \State \( q \leftarrow \text{new node(level=}\ell) \)
\State \( \delta(q,0) \leftarrow \delta(q,1) \leftarrow \text{NIL} \)
\State \( L.\text{append}(q) \)
\EndIf
\If {\( S \text{ not empty} \)} \State \( \text{parent}, \alpha \leftarrow S.\text{pop()} \)
\State \( \delta(\text{parent}, \alpha) \leftarrow q \)
\If {\( \alpha = 1 \)} \State \( S.\text{push}((\text{parent}, 0)) \)
\EndIf
\EndIf
\State \( S.\text{push}((q, 1)) \)
\State \( p, \mathcal{E} \leftarrow p', \mathcal{E} \cup \{\ell - 1, \ell - 1\} \)
\EndWhile
\EndWhile
\State \( r \leftarrow r - \Delta \)
\State \( \mathcal{E} \leftarrow \mathcal{E} \cup \{\ell - 1\} \)
\Else
\State \( q, \alpha \leftarrow S.\text{pop()} \)
\State \( \ell \leftarrow \text{level}(q) \)
\State \( w \leftarrow \sum_{j=0}^{\ell - 1} p_j \)
\If {\( \alpha = 1 \)} \State \( \text{nbChoices} \leftarrow w - p_\ell \)
\State \( \ell' \leftarrow \mathcal{E}.\text{popMin()} \)
\State \( \Delta \leftarrow \text{nbChoices} - M(p, \mathcal{E}) \)
\If {\( r - \Delta < 0 \)} \State \( D.\text{push}(r \mod \text{nbChoices}) \)
\State \( r \leftarrow r \div \text{nbChoices} \)
\EndIf
\EndIf
\State \( r \leftarrow r - \Delta \)
\State \( \text{sizePool} \leftarrow w - 1 \)
\State \( S.\text{push}((q, 0)) \)
\EndElse
\State \( \text{sizePool} \leftarrow w \)
\State \( D.\text{push}(r \mod \text{sizePool}) \)
\State \( r \leftarrow r \div \text{sizePool} \)
\EndIf
\EndIf
\EndWhile
\EndIf
\EndFor
\State \( \text{return } L, \delta, D \) \hspace{1em} \triangleright \text{list of nodes } L, \text{ transition function } \delta, \text{ ranks for non tree edges in } D \)
\end{algorithmic}

\textbf{Proof.} Proof of Proposition 17 (First pass of unranking with fixed profile) The algorithm makes \( O(kn) \) requests to an oracle computing the number of multitenry ROBDDs with a given profile and a given entry multiset, when trying to instantiate a new node in a layer (the number of layers...
is upper bounded by $k$). Each request performs $O(kn^2)$ arithmetical operations over $\mathbb{Z}$. The complexity of computing the number $M(\cdot, \cdot)$ of multientry ROBDDs is given in Proposition \[10\] \[\square\]

A.3.3. Step 3: inorder traversal for identifying non tree edges. The final step consists in making an inorder traversal of the spine, and adjusting the destinations of half edges with the rank information which was computed in the first pass in step 2. The detailed pseudocode is given in Algorithm 7.

Algorithm 7: adjustNonTreeEdges($L, \delta, D$)

Input: A spine $(L, \delta)$, a stack $D$ containing ranks relatively to the pool (depending of the current node in the traversal)

Output: The transition function $\delta$ is updated and completed

$q \leftarrow L[0]$ \hspace{1cm} ▷ root of the spine

$k \leftarrow level(q)$ \hspace{1cm} ▷ level of the root

$P \leftarrow \{\bot, T\}$ \hspace{1cm} ▷ initial pool with constants in layer 0

$S \leftarrow$ empty stack \hspace{1cm} ▷ for inorder traversal

while $S$ not empty or not $(q \neq \text{NIL} \text{ and } level(q) > 0)$ do

if $q \neq \text{NIL}$ and $level(q) > 0$ then

$S$.push($q$)

$q \leftarrow \delta(q, 0)$ \hspace{1cm} ▷ go to low kid (inorder)

else

$q \leftarrow S$.pop()

$P \leftarrow P \cup \{q\}$

$q_0, q_1 \leftarrow \delta(q, 0), \delta(q, 1)$

if $q_0 = \text{NIL}$ or $q_1 = \text{NIL}$ then

$r \leftarrow D$.pop() \hspace{1cm} ▷ get the rank info for the current node $q$

if $q_0 = \text{NIL}$ then

if $q_1 = \text{NIL}$ then

$(q_0', q_1') \leftarrow r$-th pair of distinct nodes in pool($q$): skip the pairs of children of the siblings

$\delta(q, 0), \delta(q, 1) \leftarrow q_0', q_1'$

else

$q' \leftarrow r$-th node in pool($q$)

$\delta(q, 0) \leftarrow q'$

else

$q' \leftarrow r$-th node in pool($q$): skip node $q_0$

$\delta(q, 1) \leftarrow q'$

$q \leftarrow q_1$

\]

Proof. Proof of Proposition \[18\] (Second pass of unranking with fixed profile: adjusting non tree edges) A basic task is to be able to retrieve the $r$-th node of the pool of the current visited node $q$. The previous step returns a sequence of nodes in reverse postorder. We can maintain dynamically the pool of the current node by storing the nodes visited in inorder traversal, storing the number of nodes per layer. Hence finding the $r$-th node in the pool can be done in time $O(k + \log n)$ with binary search (i.e., find the layer, and then find the node inside the layer of the pool which is sorted) with a space $O(n)$ to represent the pool.

For external nodes, finding the $r$-th pair $(i, j)$ in a ordered set $S$ skipping an ordered forbidden set of pairs $\mathcal{F}$, can be done in $\log(n)^2$ by adapting the procedure that unrank a pair given its rank when $\mathcal{F} = \emptyset$ (see helper functions in next paragraph). However for each external node, we must build the set $\mathcal{F}$ which can be of size $O(n)$. Thus we have to compute the corresponding ranks within the pool of the pairs of children of the siblings of the current node and then sort this set $\mathcal{F}$. This yields $O(n)$ time complexity with a radix sort of $\mathcal{F}$ in $O(n)$ space complexity. Then we must retrieve the actual nodes in the pool with a time complexity $O(k + \log(n))$. The extra
space use is again $O(n)$. Finally Algorithm 7 has $O(nk + n^2)$ time complexity and uses $O(n)$ extra space as we must visit all the nodes of the spine.

Helper functions for unranking pairs. We present some helper functions used to associate a rank with a pair and conversely a pair with a rank in time complexity $O(1)$. These functions are used in Algorithm 7 when adjusting non tree edges. Let us fix $M \in \mathbb{N}_{>0}$ and consider the set of pairs $M = \{(i, j) \mid 0 \leq i, j < M \text{ and } i \neq j\}$.

We use the lexicographic order ‘$<$’ on this set, i.e., $(a, b) < (c, d)$ if and only if either $a < c$, either $a = c$ and $b < d$.

The $r$-th pair in $M$ is $(i, j)$ with

$$i = R \div M, \quad j = R \mod M \quad \text{with } R = r + (r \div M) + 1.$$ 

Conversely the rank of $(i, j) \in M$ is

$$r = R - 1 + \left((R - 1) \div (M + 1)\right) \quad \text{with } R = iM + j.$$
A.4. **Example of ROBDD.** The cardinality of the set ROBDDs of size 60 with at most 15 variables (maximal size is $M_{15} = 4351$) is approximately $1.59 \cdot 10^{114}$. The ROBDD with profile $p = [1, 2, 3, 4, 2, 4, 4, 6, 4, 7, 6, 6, 4, 3, 2, 2]$ was produced by our random uniform sampler and is depicted on Figure 3.

**Figure 3.** An example of ROBDD of size 60 with 15 variables. Tree edges (corresponding to the spine) and non tree edges are respectively drawn in black and red.

Julien Clément. Université Caen Normandie, GREYC, CNRS - UMR 6072.  
*Email address: Julien.Clement@unicaen.fr*

Antoine Genitrini. Sorbonne Université, CNRS, LIP6, UMR 7606, F-75005 Paris, France.  
*Email address: Antoine.Genitrini@lip6.fr*