A Theoretical and Numerical Analysis of the Worst-Case Size of Reduced Ordered Binary Decision Diagrams

JIM NEWTON and DIDIER VERNA, EPITA, France

Binary Decision Diagrams (BDDs) and in particular ROBDDs (Reduced Ordered BDDs) are a common data structure for manipulating Boolean expressions, integrated circuit design, type inferencers, model checkers, and many other applications. Although the ROBDD is a lightweight data structure to implement, the behavior, in terms of memory allocation, may not be obvious to the program architect. We explore experimentally, numerically, and theoretically the typical and worst-case ROBDD sizes in terms of number of nodes and residual compression ratios, as compared to unreduced BDDs. While our theoretical results are not surprising, as they are in keeping with previously known results, we believe our method contributes to the current body of research by our experimental and statistical treatment of ROBDD sizes. In addition, we provide an algorithm to calculate the worst-case size. Finally, we present an algorithm for constructing a worst-case ROBDD of a given number of variables. Our approach may be useful to projects deciding whether the ROBDD is the appropriate data structure to use, and in building worst-case examples to test their code.

CCS Concepts: • **Mathematics of computing** → *Combinatoric problems*; • **Computing methodologies** → *Boolean algebra algorithms*;

ACM Reference Format:
Jim Newton and Didier Verna. 2018. A Theoretical and Numerical Analysis of the Worst-Case Size of Reduced Ordered Binary Decision Diagrams. 1, 1 (November 2018), 45 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Binary Decision Diagrams (BDDs) are a data structure useful for representing Boolean expressions. The data structure has countless applications to problems involving Boolean algebra. In the Art of Computer Programming [Knu09, Page iv], Donald Knuth writes, "[BDDs] have become the data structure of choice for Boolean functions and for families of sets, and the more I play with them the more I love them. For eighteen months I've been like a child with a new toy, being able now to solve problems that I never imagined would be tractable."

The decision diagram has been defined in several different flavors in currently available literature. Colange [Col13, Section 2.3] provides a succinct historical perspective, including the BDD [Bry86], the Multi-Valued Decision Diagram (MDD) [Sri02], Interval Decision Diagram (IDD) [ST98], the Multi-Terminal Binary Decision Diagram (MTBDD) [CMZ+97], the Edge-Valued Decision Diagram (EVDD) [LS92], and the Zero-Suppressed Binary Decision Diagram (ZBDD) [Min93].

The particular decision diagram variant which we investigate in this article is the Reduced Ordered Binary Decision Diagram (ROBDD). When we use the term ROBDD we mean, as the name implies, that the BDD has its variables **O**rdered as described in Section 2.1 and has been fully **R**educed by the rules presented in Section 2.2. It is worth noting that there is variation in the

Authors' address: Jim Newton, jnewton@lrde.epita.fr; Didier Verna, didier@lrde.epita.fr, EPITA, Research and Development Laboratory (LRDE), Le Kremlin-Bicêtre, France, 94270.

XXXX-XXXX/2018/11-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

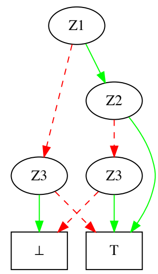


Fig. 1. BDD for $(Z_1 \wedge Z_2) \vee (Z_1 \wedge \neg Z_2 \wedge Z_3) \vee (\neg Z_1 \wedge \neg Z_3)$

terminology used by different authors. For example, Knuth [Knu09] and Bryant [Bry18] both use the unadorned term BDD for what we are calling an ROBDD.

Even though the ROBDD is a lightweight data structure to implement, and can be easily implemented, some of its behavior regarding the amount of necessary memory allocation may not be obvious in practice. In this paper we convey an intuition of expected sizes and shapes of ROBDDs from several perspectives.

Section 2 provides illustrations of ROBDD constructing from the point of view of reduction operations. Section 3.2 examines worst cases sizes of ROBDDs: first, we look exhaustively at cases involving a small number of variables; then, we examine experimentally the average and worst-cases sizes for several cases involving more variables. Section 3 examines the shapes of the graphs of the worst-cases sizes. In Section 3.6 we use an intuitive understanding to derive an explicit formula to calculate the worst-case size for a given number of variables. Finally in Section 4, we provide an algorithm for generating a worst-case sized ROBDD for a given number of Boolean variables.

2 BDD CONSTRUCTION AND REDUCTION

An equation of Boolean variables can be represented by a data structure called a Binary Decision Diagram (BDD). The literature on BDDs is abundant [Bry86, Bry92, Ake78, FTV16] [Knu09, Section 7.1.4] [Col13, Section 2.3]. Andersen summarizes many of the algorithms for efficiently manipulating BDDs [And99]. We do not provide a formal definition of BDD here. Instead the interested reader is invited to consult [And99] or [Knu09, Section 7.1.4].

BDDs can be implemented easily in a variety of programming languages with only a few lines of code. The data structure provides a mechanism to manipulate Boolean expressions elegantly. Operations such as intersection, union and complement can be performed resulting in structures representing Boolean expressions in canonical form [Bry86]. The existence of this canonical form makes it possible to implement the equality predicate for Boolean expressions, either by straightforward structural comparison, or by pointer comparison depending on the specific BDD implementation. Some programming languages model types as sets [HVP05, CL17, Ans94]. In such programming languages, the BDD is a potentially useful tool for representing types and for performing certain type manipulations [Cas16, NVC17, NV18].

Figure 1 shows an example of a BDD which represents a particular function of three Boolean variables: Z_1, Z_2, and Z_3. The BDD in the figure is actually an ROBDD; we will define more precisely what that means later. When the Boolean function is expressed in Disjunctive Normal Form (DNF), it contains three terms, each of which being represented by a respective path in the BDD from the

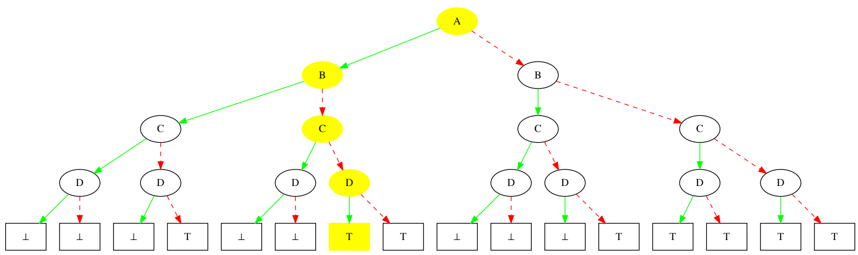


Fig. 2. UOBDD for $F = \neg((A \wedge C) \vee (B \wedge C) \vee (B \wedge D))$. The highlighted path of nodes corresponds to the highlighted row of the truth table in Figure 3.

root node, Z_1, to the leaf node, $\perp$. The variables in each term are logically negated (*i.e.* $\neg Z_i$) if the path leaves node Z_i via its dashed red exit arrow, and are not negated (*i.e.* Z_i) if the path follows the solid green exit arrow.

In order to avoid confusion, when this article is printed in black and white, we hereafter refer to the red dashed arrow as the negative arrow and the green solid arrow as the positive arrow. Respectively, we refer to the nodes which the arrows point to as the positive and negative child nodes.

There are several conventions used in literature for graphically representing a Boolean expression as a BDD. Some conventions indicate the false (logically negated $\neg$) case as an arrow exiting the node on the bottom left and the true case as an arrow exiting the node on the left. We found that such a convention forces BDDs to be drawn with excessively many crossing lines. In order to allow the right/left arrows within the BDDs to be permuted, thus reducing the number of line crossings, we avoid attaching semantic information to left and right arrows, and instead use red dashed arrows for the false (negative) case and solid green arrows for the true (positive) case.

Casually generating a set of sample BDDs for randomly chosen Boolean expressions quickly reveals that a BDD may have redundant subtrees. It seems desirable to reduce the memory footprint of such a tree by reusing common subtrees (sharing pointers) or eliminating redundant nodes. Here, we introduce one such approach: first, we start with a complete binary tree (Section 2.1), and then, we transform it into a reduced graph by applying certain reduction rules to its nodes (Section 2.2). The process is intended to be intuitive, conveying an understanding of the topology of the resulting structure. On the contrary, this construction process is not to be construed as an algorithm for efficiently manipulating BDDs programmatically.

In addition to significantly reducing the memory footprint of a BDD, the optimization strategy described here also enables certain significant algorithmic optimizations, which we won't discuss in depth in this article. In particular, the equality of two Boolean expressions often boils down to a mere pointer comparison [NVC17].

2.1 Initial construction step

One way to understand the (RO)BDD representation of a Boolean expression is by first representing the truth table of the Boolean expression as decision diagram. Consider this Boolean expression of four variables: $\neg((A \wedge C) \vee (B \wedge C) \vee (B \wedge D))$. Its truth table is given in Figure 3, and its BDD representation in Figure 2. When a BDD is a tree corresponding exactly to the truth table of its Boolean function (which is not necessarily true), the BDD is referred to as an UOBDD (unreduced ordered BDD). Each non-leaf node of the UOBDD represents the appearance of a Boolean variable

A	B	C	D	F
$\top$	$\top$	$\top$	$\top$	$\bot$
$\top$	$\top$	$\top$	$\bot$	$\bot$
$\top$	$\top$	$\bot$	$\top$	$\bot$
$\top$	$\top$	$\bot$	$\bot$	$\top$
$\top$	$\bot$	$\top$	$\top$	$\bot$
$\top$	$\bot$	$\top$	$\bot$	$\bot$
$\top$	$\bot$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\bot$	$\top$
$\bot$	$\top$	$\top$	$\top$	$\bot$
$\bot$	$\top$	$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$	$\top$	$\bot$
$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\bot$	$\bot$	$\top$	$\top$	$\top$
$\bot$	$\bot$	$\top$	$\bot$	$\top$
$\bot$	$\bot$	$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$

Fig. 3. Truth table for $F = \neg\,((A \wedge C) \vee (B \wedge C) \vee (B \wedge D))$. The highlighted row of the truth table corresponds to the path of highlighted nodes in Figure 2.

in the expression. Each path from the root node A to a leaf node represents one row of the truth table. For example, the highlighted path in Figure 2 corresponds to the highlighted row in Figure 3.

A BDD is said to be ordered if there is some ordering of the variables $\{v_1, v_2, ..., v_n\}$, such that whenever there is an arrow from v_i to v_j then $i < j$. Some authors such as Gröple *et al.* [GPS98, GPS01] and Langberg *et al.* [LPR03], when exploring a BDD variant called qOBDD, consider the further restriction that arrows only connect v_i with v_{i+1}. We do not make such a restriction in our treatment of ROBDDs. For UOBDDs having not yet undergone any reduction, being ordered implies that every branch from the root to a leaf contains exactly the same variables in the same order. In Figure 2 for example, every path from the root to a leaf contains exactly A, B, C, D in that order. On the contrary, and as in Figure 1, some of these paths contain fewer nodes than others. Nevertheless, the nodes visited by each path remain ordered. For the extent of this paper we will use the natural lexicographical orderings: $A < B < C < D$, and $Z_1 < Z_2 < ...Z_n$.

2.2 Reduction rules

The three reduction rules described in this section give us the ability to convert an ordered BDD into an ROBDD.

Given its UOBDD, it is straightforward to evaluate an arbitrarily complex Boolean expression of n variables, simply descend the tree in n steps according to the values of the n variables. This is equivalent to tracing across the corresponding row of the truth table (see the highlighting in Figure 3). However, the size of the tree grows exponentially with the number of variables. The UOBDD representing a Boolean expression of n variables has

$$|UOBDD_n| = 2^{n+1} - 1 \tag{1}$$

nodes. Fortunately, it is possible to reduce the allocated size of the UOBDD by taking advantage of certain redundancies. There are three rules which can be used to guide the reduction. Andersen and Gröple [And99, GPS98] also explain the merging and deletion rules, so we will dispense with many of the details. However, we consider an extra rule, the terminal rule, which is really a special case of the merging rule.

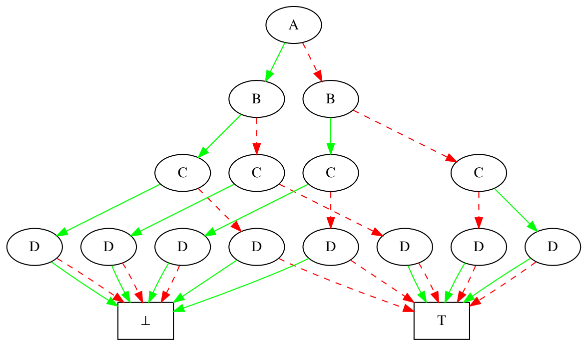


Fig. 4. BDD after applying Terminal rule

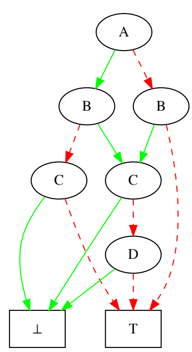


Fig. 5. BDD after applying Terminal, Deletion, and Merging rules. This diagram is an ROBDD logically equivalent to the UOBDD shown in Figure 2.

Terminal rule: The only possible leaf nodes are $\top$ and $\bot$, so these nodes can be represented by two singleton objects, allowing pointers to them to be shared.

Deletion rule: If any node X is such that its positive and negative arrows both point to the same node Y, then X is said to be symmetric. Such a node can be deleted and arrows previously pointing to it may be promoted to point to Y directly.

Merging rule: If any two nodes U and V corresponding to the same Boolean variable are such that their positive arrows both point to node X and negative arrows both point to node Y, then U and V are said to be congruent, and they may be merged. Any arrow pointing to U may be updated to point to the V, and U may be removed (or the other way around).

Applying the Terminal rule reduction cuts the number of nodes roughly by half, as shown in Figure 4.

Further applications of the deletion rule and merging rules, results in the ROBDD shown in Figure 5. In this case the graph shrinks from 31 nodes in Figure 2 to 8 nodes in Figure 5.

3 WORST-CASE ROBDD SIZE AND SHAPE

The BDD shown in Figure 2 is a 31 nodes UOBDD, reduced in Figure 5 to an 8 nodes ROBDD, thanks to the three reduction rules presented in Section 2.2. We may naturally ask whether this reduction process is typical.

The size and shape of a reduced BDD depends on the chosen variables ordering [Bry86]. Finding the best ordering is coNP-Complete [Bry86]. In this article we do not address the questions of choosing or improving the variables ordering. Given a particular variables ordering however, the size and shape of the ROBDD depends only on the truth table of the Boolean expression. In particular, it does not depend on the chosen representation for the expression. For example, $(A \vee B) \wedge C$ has the same truth table as $(A \wedge C) \vee (B \wedge C)$, so these two expressions are equivalent and will be reduced to the exact same ROBDD. In a practical sense, the ROBDD serves as a canonical form for Boolean expressions.

The best-case size (in terms of node count) for a constant expression is obviously one, *i.e.*, a Boolean expression which is identically $\top$ or $\bot$. But what is the worst-case size of an ROBDD of n variables? We examine this question both experimentally and theoretically in the following sections.

3.1 Process summary

We start by showing all possible ROBDDs for the 1- and 2-variable cases. Then, we address the question of the worst-case size by looking at the exhaustive list of ROBDDs up to the 4-variable case, extrapolating from random samples thereafter. The way we do random sampling is also explained.

Given the above data, we observe that the difference between the worst-case size and the average size becomes negligible as the number of Boolean variables increases. At this stage however, this observation is only a conjecture, and we would like to prove it formally. We define a quantity called residual compression ratio which measures how effective a representation the ROBDD is as compared to the size of the truth table. We note from experimental data that this ratio decreases, but to which value is unclear.

We continue by deriving a formula for the worst-case ROBDD size, based on the number of nodes in each row, and holding for any number of variables. This derivation is motivated by images of sample worst-case ROBDDs as the number of variables increases. The derivation is obtained as follows.

First, we introduce a threshold function which represents the competition between an increasing exponential and a decreasing double exponential. We are able to express the worst-case ROBDD size in terms of this threshold. Then we argue that natural number thresholds are non-decreasing, and that real number thresholds are strictly increasing. We then derive bounds on the threshold function, and use them to provide an Algorithm for computing threshold values, usually within one or two iterations.

Ultimately, we use those bounds to show that the residual compression ratio indeed tends to zero.

3.2 Experimental analysis of worst-case ROBDD Size

We saw above that, given a variable ordering, every truth table of n variables corresponds to exactly one ROBDD. Otherwise stated, there is a one–to–one correspondence from the set of n-variable truth tables to the set of n-variable ROBDDs. However, for any given truth table, there are infinitely many equivalent Boolean expressions. An n-variable truth table has 2^n rows, and each row may contain a $\top$ or $\bot$ as the expression's value. Thus there are 2^{2^n} different n-variable truth tables.

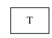
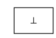
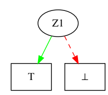
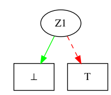
No. Nodes	ROBDD and Boolean Expression	ROBDD and Boolean Expression
1	 $\top$	 $\bot$
3	 Z_1	 $\neg Z_1$

Fig. 6. All ROBDDs of one variable

For small values of n it is reasonable to consider every possible ROBDD exhaustively to determine the maximum possible size. However, it becomes impractical to do so for large values. For example, there are $2^{2^{10}} > 1.80 \times 10^{308}$ ROBDDs of 10 variables. In our analysis, we treat the 1- through 4-variable cases exhaustively, and use extrapolated results (explained below) otherwise.

Figure 6 shows all the possible ROBDDs of a single variable. We see that only 4 ROBDDs are possible ($2^{2^n} = 2^{2^1} = 4$). Two of the ROBDDs have one node, and two have two nodes. Here we consider an n-variable expression as an expression having n or fewer variables. This is because some Boolean expressions of n variables can be reduced to equivalent expressions having fewer ones. For example, $A \vee (A \wedge \neg B)$, a 2-variable expression, is in fact equivalent to just A. Figure 7 shows an exhaustive list of the possible ROBDDs of 2 variables. Here, the worst-case node count is 5, occurring twice out of a total of $2^{2^n} = 2^{2^2} = 16$ possible expressions.

3.3 Statistics of ROBDD size distribution

Figure 9 and Figure 10 are histogram plots illustrating the possible sizes of an ROBDD *vs.* the number of possible Boolean functions which reduce to an ROBDD of that size. In Figure 9, we have exhaustively counted the possible sizes of each ROBDD for 1 to 4 variables. In Figure 10 we have extrapolated from random sampling the 5- through 10-variable cases as described below. The worst case for 4 variables is 11 nodes. We can estimate from Figure 9 that of the 65536 different Boolean functions of 4 variables, only about 12000 of them (18%) have node size 11. The average size is about 10 nodes.

We generated the data in Figure 10 for 5 through 8 Boolean variables, by randomly selecting truth tables, counting the nodes in the ROBDD, and multiplying by a factor to compensate for the sample size. In particular, we did the computation work in Common Lisp [Ans94] using the SBCL [New15] Common Lisp compiler. SBCL (version 1.4.3) uses the MT19937 `prng` algorithm [MN98] for generating random numbers. For each plot, we generated a list of random integers between 0 and $2^n - 1$, removing duplicates so as not to count the same truth table twice. From each such sampled integer, we generated an ROBDD and counted its nodes. For example, from the integer $9_{10} = 01001_2$ represents the truth table shown in Figure 8. From this truth table we generated the Boolean expression

$$((\neg Z_1 \wedge \neg Z_2 \wedge \neg Z_3 \wedge \neg Z_4 \wedge \neg Z_5) \vee (Z_1 \wedge Z_2 \wedge \neg Z_3 \wedge \neg Z_4 \wedge \neg Z_5)),$$

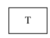
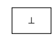
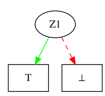
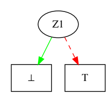
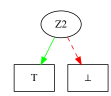
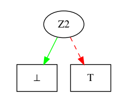
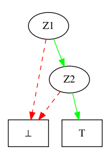
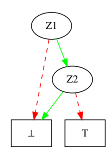
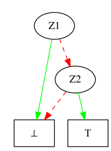
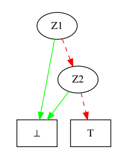
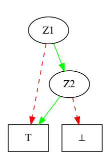
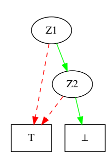
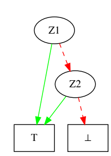
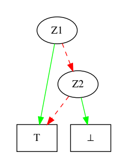
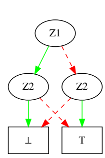
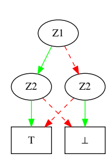
No. Nodes	ROBDD and Boolean Expression	ROBDD and Boolean Expression	ROBDD and Boolean Expression	ROBDD and Boolean Expression
1	 $\top$	 $\bot$		
3	 Z_1	 $\neg Z_1$	 Z_2	 $\neg Z_2$
4	 $(Z_1 \wedge Z_2)$	 $(Z_1 \wedge \neg Z_2)$	 $(\neg Z_1 \wedge Z_2)$	 $(\neg Z_1 \wedge \neg Z_2)$
4	 $((Z_1 \wedge Z_2) \vee \neg Z_1)$	 $((Z_1 \wedge \neg Z_2) \vee \neg Z_1)$	 $((\neg Z_1 \wedge Z_2) \vee Z_1)$	 $((\neg Z_1 \wedge \neg Z_2) \vee Z_1)$
5	 $((Z_1 \wedge \neg Z_2) \vee (\neg Z_1 \wedge Z_2))$	 $((Z_1 \wedge Z_2) \vee (\neg Z_1 \wedge \neg Z_2))$		

Fig. 7. All ROBDDs of two variables

and from that Boolean expression, we generated the corresponding ROBDD.

Construction of such large ROBDDs is compute intensive. We have shared access to a cluster of Intel Xeon^{TM} E5-2620 2.00GHz 256GB DDR3 machines. Consequently, we tried to achieve a reasonably large sample size with the limited resources available. There are potential ways of increasing the sample size, discussed in Section 7.

Figure 12 lists the number of samples and corresponding compute times we observed. See Section 3.4 for a discussion of how we determined which number of samples to use. Figure 13

Z_5	Z_4	Z_3	Z_2	Z_1	F	min-term
0	0	0	0	0	1	$(\neg Z_1 \wedge \neg Z_2 \wedge \neg Z_3 \wedge \neg Z_4 \wedge \neg Z_5)$
0	0	0	0	1	0	
0	0	0	1	0	0	
0	0	0	1	1	1	$(Z_1 \wedge Z_2 \wedge \neg Z_3 \wedge \neg Z_4 \wedge \neg Z_5)$
0	0	1	0	0	0	
0	0	1	0	1	0	
0	0	1	1	0	0	
0	0	1	1	1	0	
...						
1	1	1	0	1	0	
1	1	1	1	0	0	
1	1	1	1	1	0	

Fig. 8. 5-Variable truth table representing $9_{10} = 1001_2$. To interpret a binary integer as a truth table, enter the bits in order with least significant bit at the top and most significant bit at the bottom. Bits which are 1 correspond to min-terms as shown.

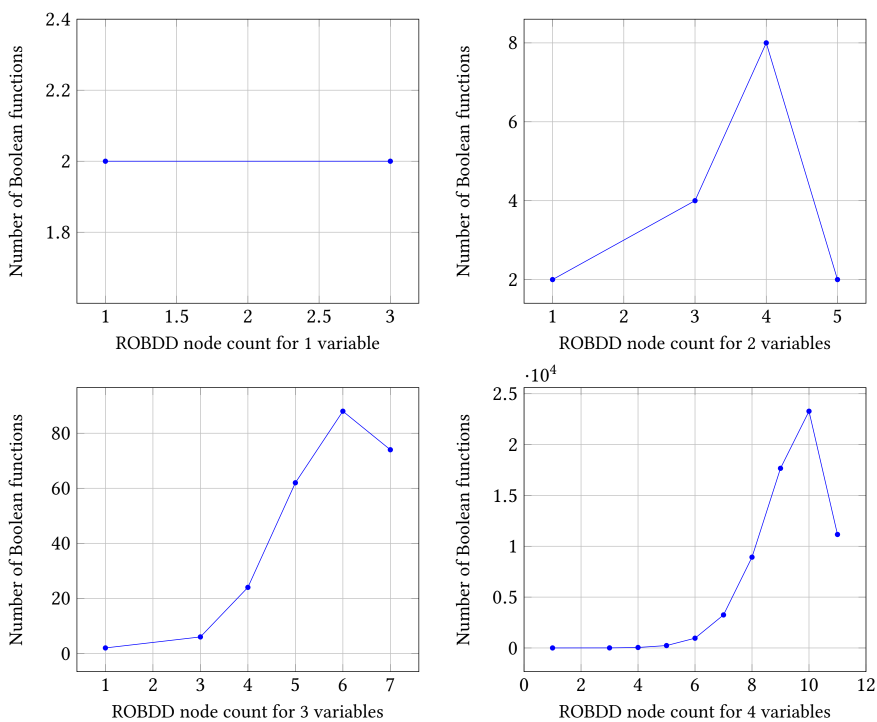


Fig. 9. Histograms illustrating size distributions of ROBBDs from 1 to 4 Boolean variables. The histograms are based on exhaustive data.

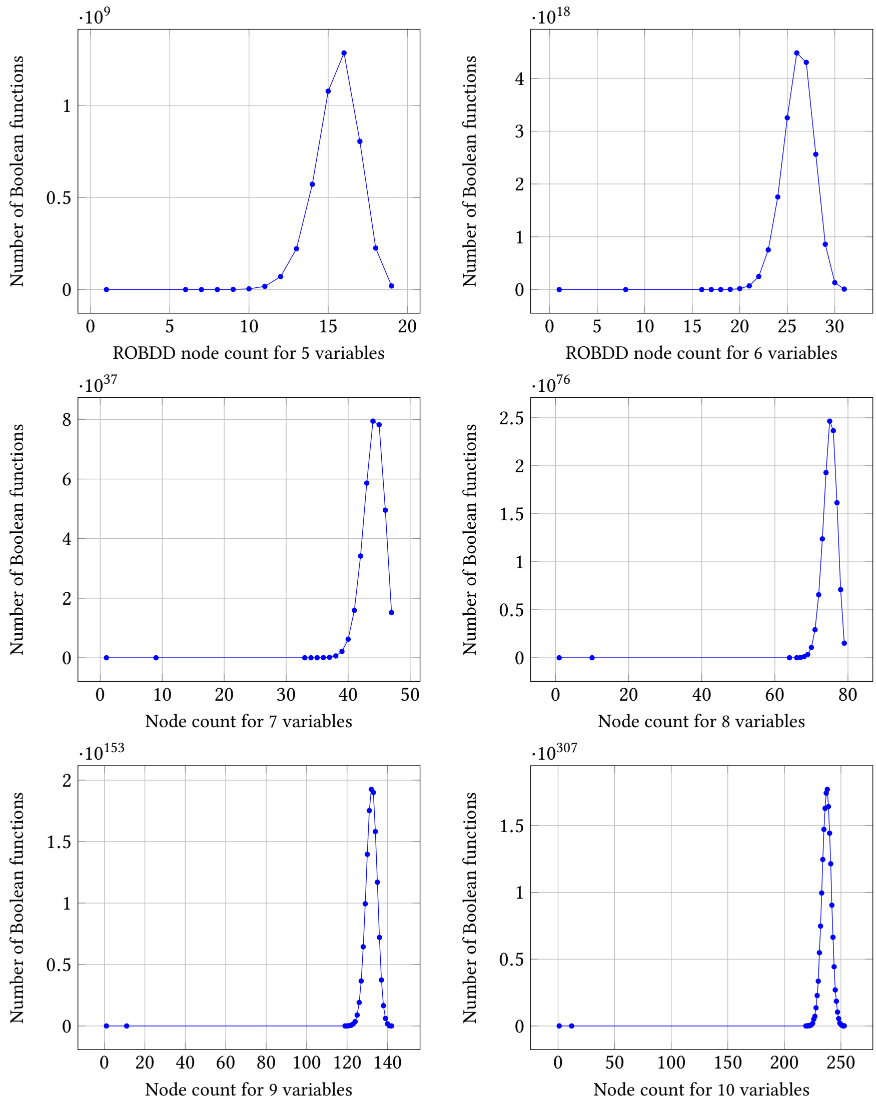


Fig. 10. Histograms illustrating size distributions of ROBBDs from 5 to 10. The histograms are based on extrapolations from sampled data.

consolidates the data from Figures 9 and 10 into a single plot but normalized so that the total number of Boolean functions in each curve is 100 percent. This normalization allows us to interpret a point (x, y) on the curve corresponding to n variables as meaning that a randomly selected Boolean expression of n variables has probability y of having an ROBDD which contains exactly x nodes.

Each point, (n, σ_n), in the plot in Figure 15 was calculated from a corresponding curve $\mathscr{C}_n$ of Figure 13 by the formula:

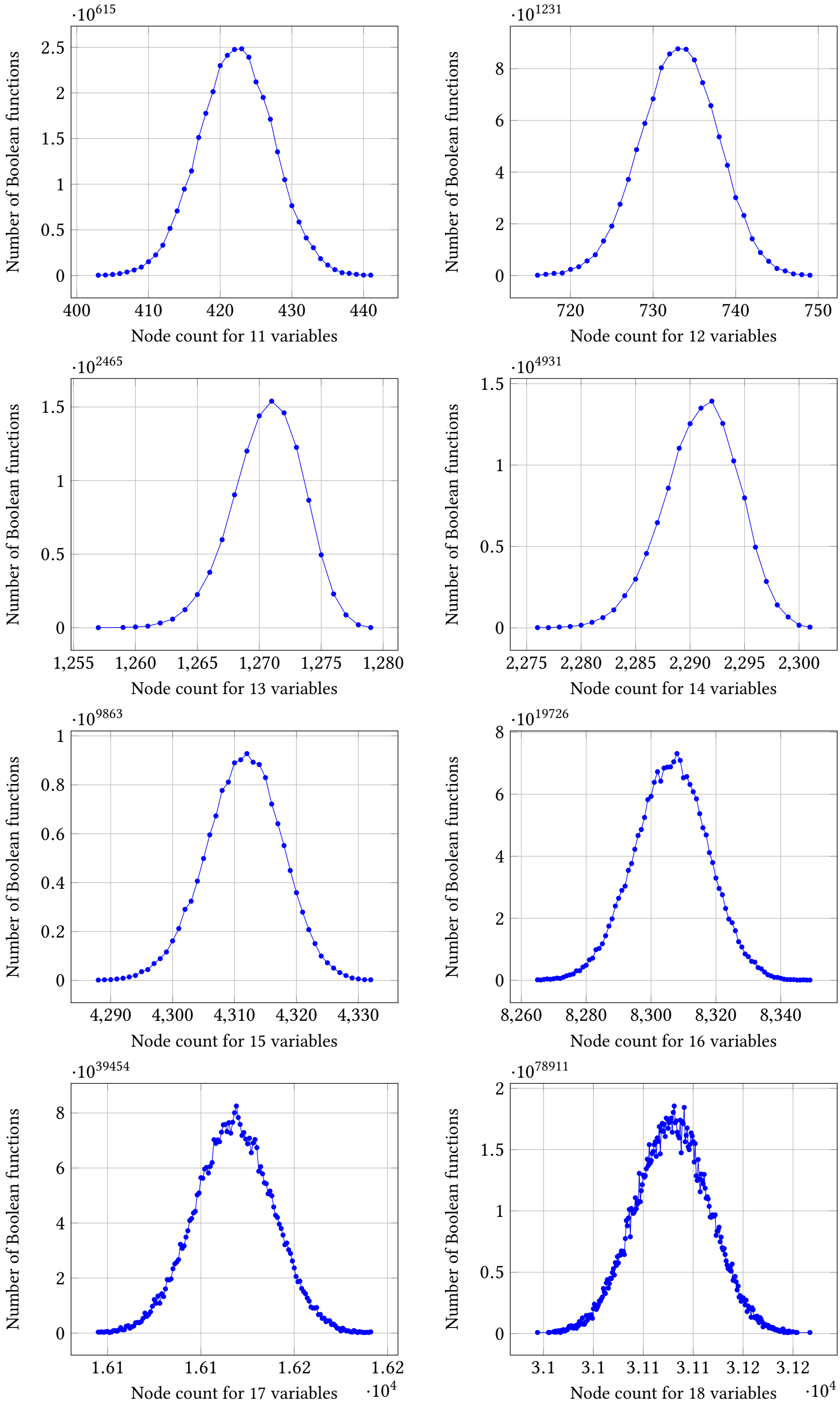


Fig. 11. Histograms illustrating size distributions of ROBBDs from 11 to 18. The histograms are based on extrapolations from sampled data.

No. Variables (n)	No. Samples	No. Unique Sizes
5	500003	15
6	400003	18
7	795986	17
8	108098	17
9	96007	26
10	58868	37
11	47522	45
12	51334	41
13	50932	25
14	50416	31
15	45340	52
16	66124	94
17	56968	162
18	55780	276
19	76374	443

Fig. 12. Number of samples and compute times for generating the plots in Figures 10 and 11. The table also shows the number of unique ROBDD sizes which were detected for each value of n.

$$\sigma_n = \sqrt{\sum_{(x,y)\in\mathscr{C}_n} y \cdot (x - \mu_n)^2}\,, \quad \text{with} \quad \mu_n = \sum_{(x,y)\in\mathscr{C}_n} x \cdot y\,.$$

It is not clear from Figure 13 whether the spread of ROBDD sizes grows with the number of variables. However, from the standard deviation plot in Figure 15, the spread seems to grow in absolute terms. Despite this, the average (expected size), median, and worst-case sizes summarized in Figure 14 give the impression that the distinction between average size and worst-case size becomes negligible as the number of variables increases. Otherwise stated, it appears that for large values of n, $|ROBDD_n|$ becomes a good approximation for average size, an observation which seems related to the Shannon Effect discussed by Gröpl *et al.* [GPS98].

The plot in Figure 13 gives the impression that the distribution of possibles ROBDD sizes for a given number of variables is clustered around the average such value. The standard deviation plot in the same figure gives an impression of how tight this clustering is. In this article, we don't present a formula for this standard deviation as a function of n, but from the plot, it appears to grow faster than linearly.

One might be tempted to assume that the data represented in Figure 10, and consequently in Figure 13, follows a normal distribution, as the curves have a bell-like shape. However, the distribution is not Gaussian. In particular, each of the curves extend left to the point $(1, 2)$ because there are always two constant functions of N variables, namely, $f = \top$ and $f = \bot$. On the other hand, we did not see any case in our experimentation where the curves extended to the right any considerable distance beyond the peak. Later, we show what the actual maximum size of an ROBDD of N variables is (see Figure 24), and in each case, the rightmost points in Figure 13 agree impeccably with Figure 24.

If we believed the data followed a Gaussian distribution, we could interpret the standard deviation more strictly. But for any distribution where we can calculate the mean and standard deviation, we

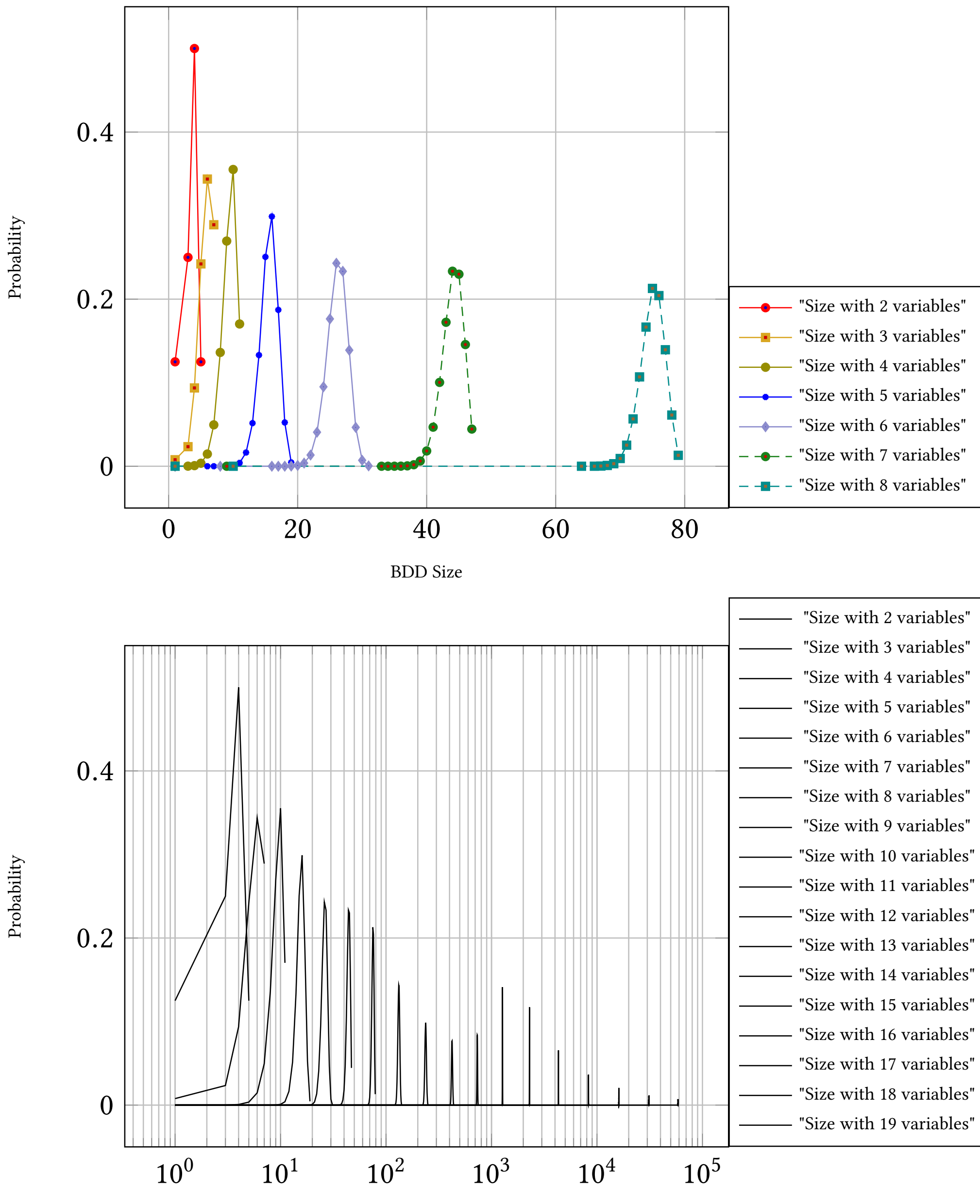


Fig. 13. Normalized histograms of size distribution probability functions for ROBBDs of 2 to 10 variable Boolean expressions, based on exhaustive data for 2, 3 and 4 variables, and on randomly sampled data for 5 and more variables.

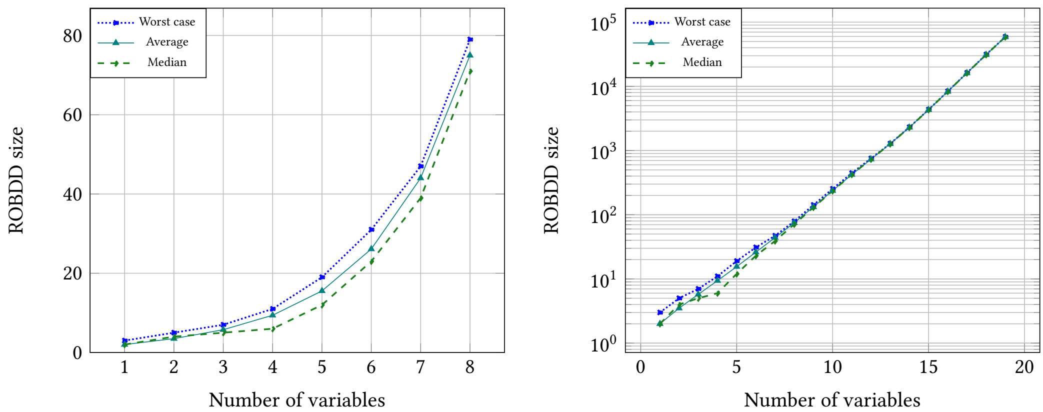


Fig. 14. Expected and worst-case ROBDD size from 1 to 10 variables, exhaustively determined for 1 through 4 variables, experimentally determined for 5 and more variables.

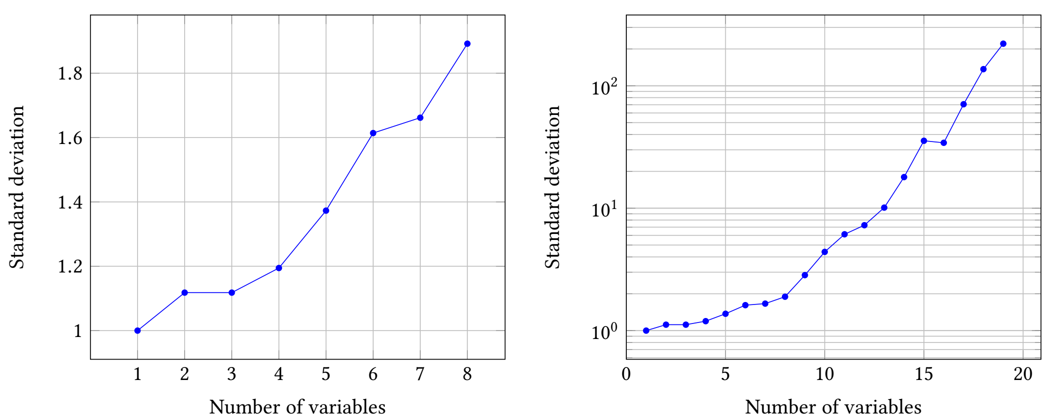


Fig. 15. Standard deviations for each of the curves shown in Figure 13 and whose averages are shown in Figure 14.

can interpret the standard deviation according to the Chebyshev inequality. The standard deviation plot (Figure 15) can be interpreted according to the Chebyshev inequality [Als11], with X being the size of a randomly selected ROBDD.

$$\Pr(|X - \mu| > k \cdot \sigma) \leq \frac{1}{k^2} \qquad \text{Chebyshev's inequality}$$

If the standard deviation of the probability function (Figure 13) for n Boolean variables is σ_n and the average ROBDD size is μ_n, then for a given real number, $k \geq 1$ ($k > 1$ in practice), the probability of a randomly selected ROBDD of n variables having more than $\mu_n + k \cdot \sigma_n$ nodes or less than $\mu_n - k \cdot \sigma_n$ nodes, is less than $\frac{1}{k^2}$. As an example of using the plots in Figures 14 with the

Chebyshev inequality, taking $k = 2$:

$$\begin{aligned}
\mu_8 &= 75.0 && \text{from Figure 14} \\
\sigma_8 &= 1.89 && \text{from Figure 15} \\
k &= 2 \\
\frac{1}{k^2} &= \frac{1}{2^2} = 25\% \\
\mu_8 - k \cdot \sigma_8 &= 71.22 \\
\mu_8 + k \cdot \sigma_8 &= 78.78\,.
\end{aligned}$$

This means that given a randomly selected 8-variable ROBDD, there is a $100\% - 25\% = 75\%$ chance that it has between 71 and 79 nodes.

3.4 Sufficiency of sample size

In Section 3.3 we discussed the approximation of ROBDD size histograms by a method of extrapolation based on a random sample. Any such sample size must necessarily be miniscule compared to the total space. For example, there are $2^{2^{10}} = 1.8 \times 10^{308}$ Boolean functions of 10 variables. In order to sample just 1% of that space, even at a rate of 1000 per second, we'd need 1.8×10^{303} seconds. However, assuming the universe is 13.8 billion years, that is only 4.4×10^{17} seconds. Knowing that we cannot sample a significant number of Boolean functions, how can we know whether we have sampled a sufficient amount to have confidence that the histograms in Figures 10 and 11 are good approximations? To answer this question, we once again take a look at the Chebyshev inequality. To trust its results, we must have confidence that the approximated average, μ_n, and standard deviation, σ_n, are close to the actual result. *I.e.*, if we increased the sample size, would μ_n and σ_n change significantly.

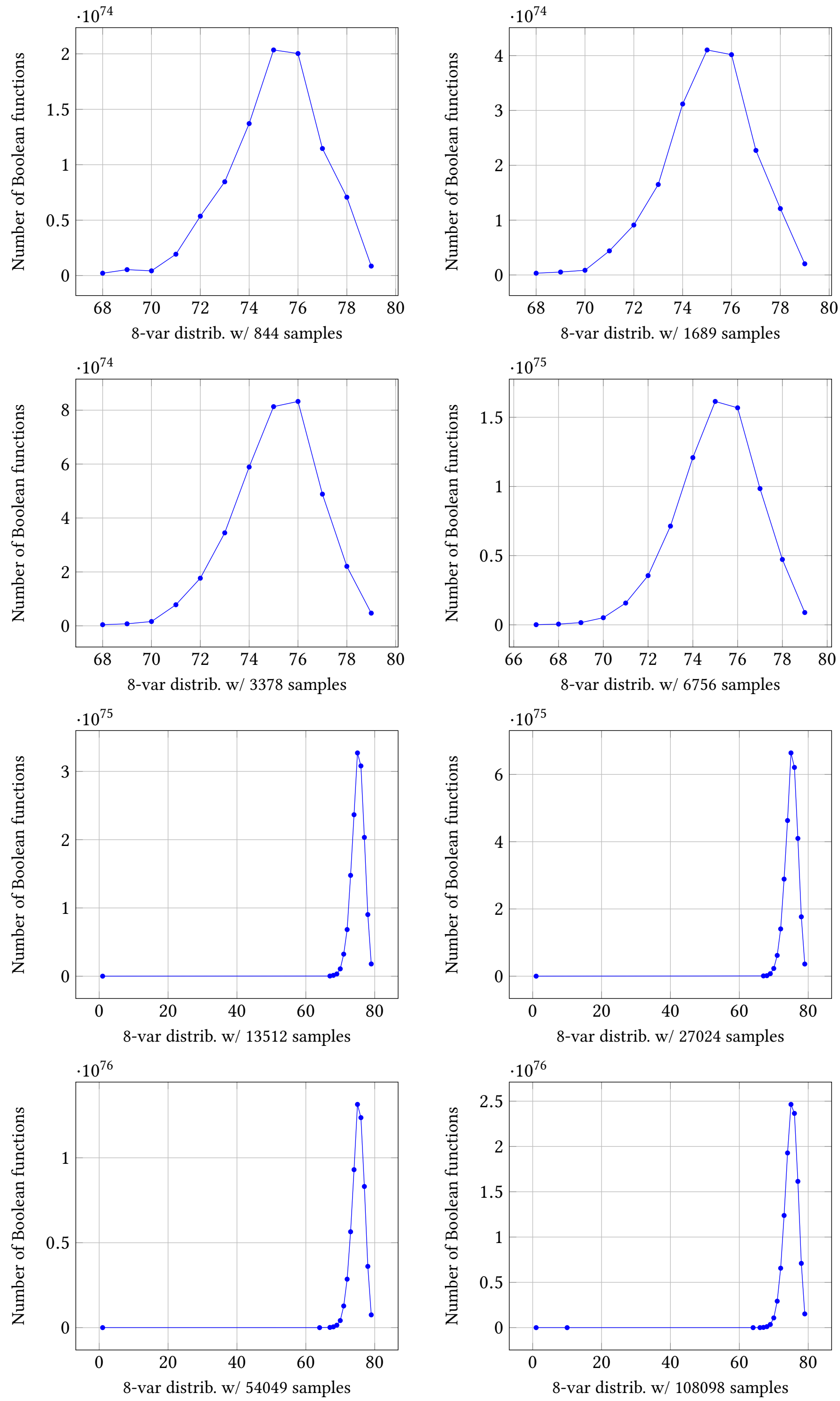


Fig. 16. Histograms illustrating size distributions of successively larger samples for 8 variables.

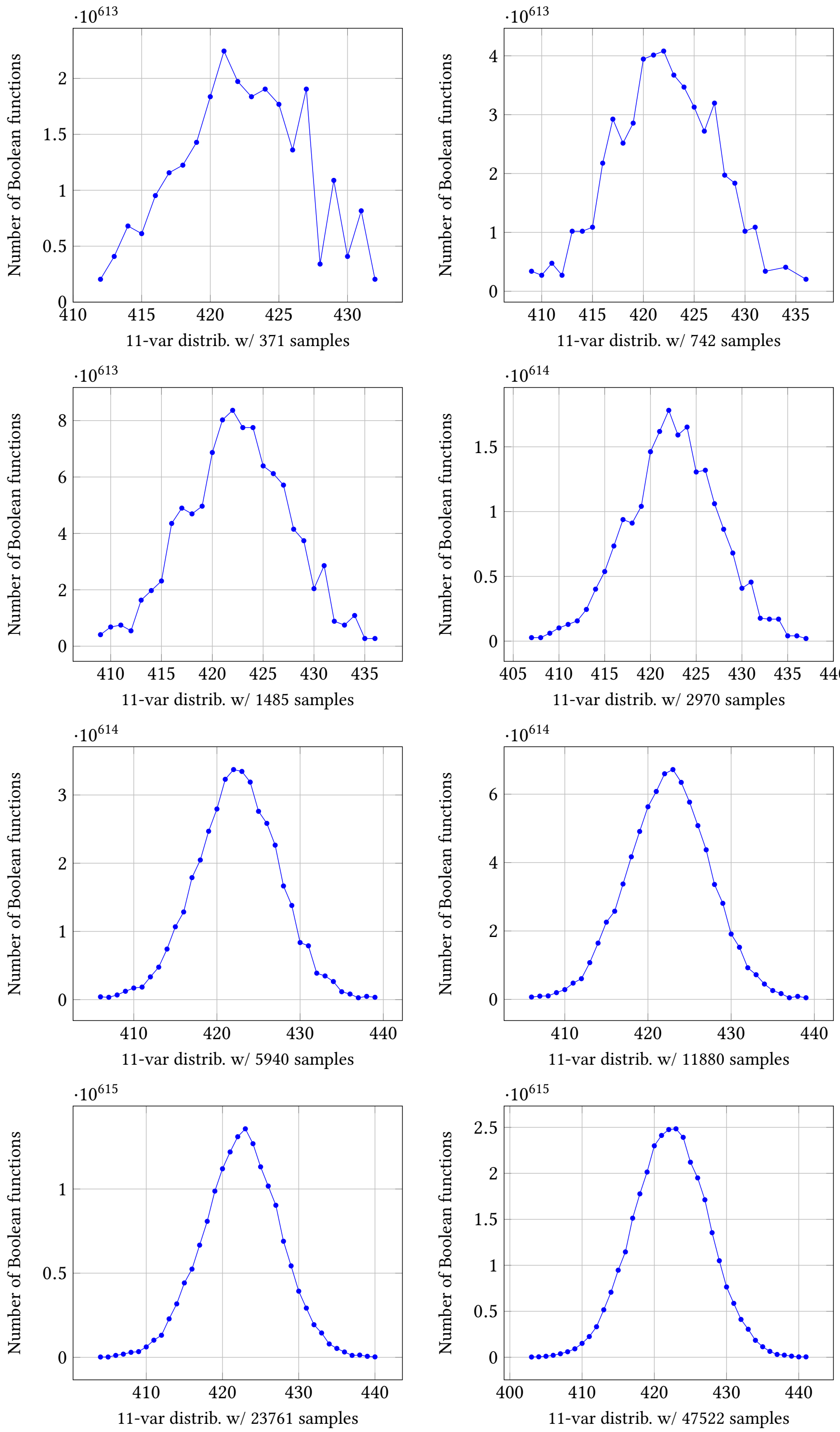


Fig. 17. Histograms illustrating size distributions of successively larger samples for 11 variables.

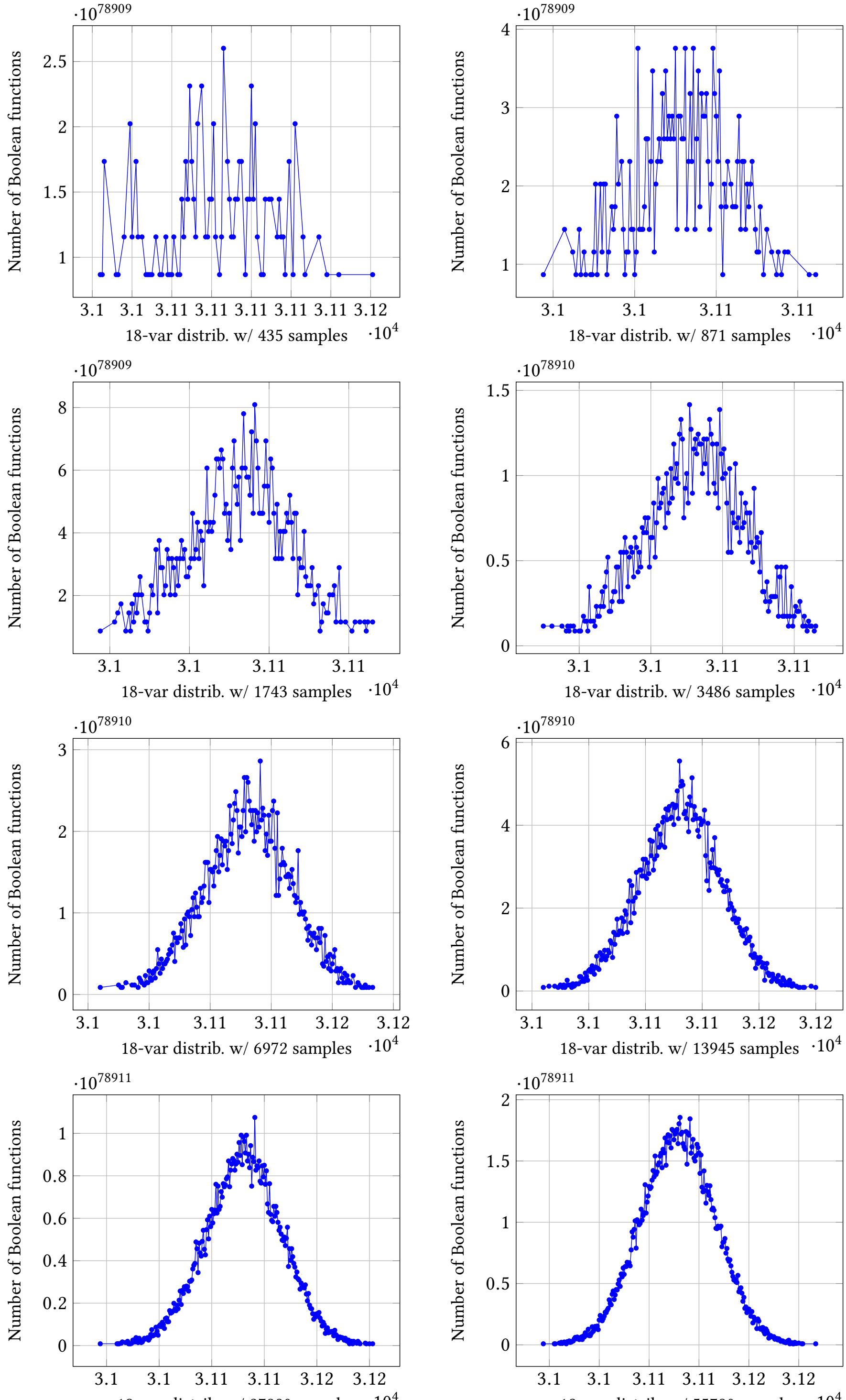


Fig. 18. Histograms illustrating size distributions of successively larger samples for 18 variables.

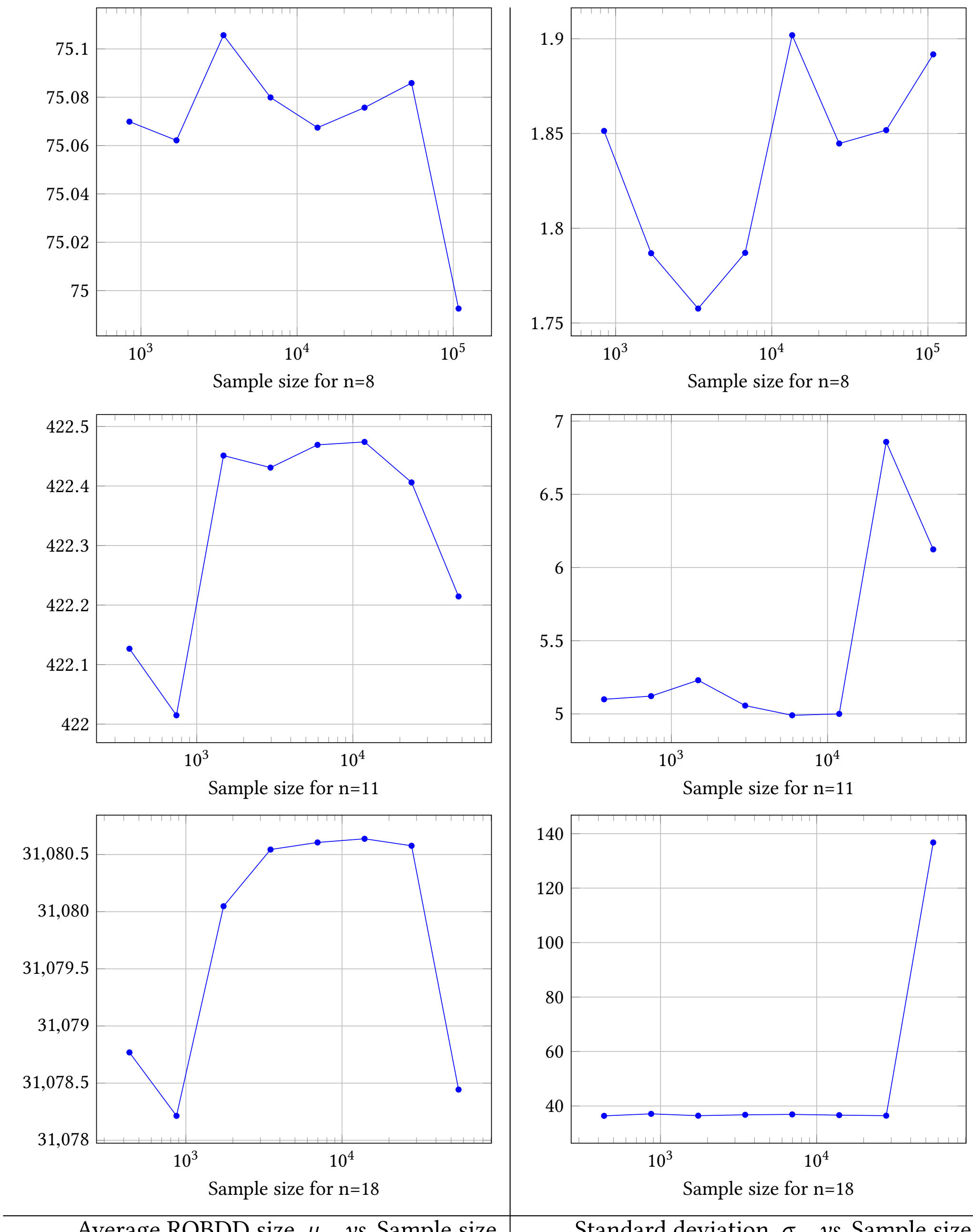


Fig. 19. Averages and Standard deviations of the 8, 11, and 18 variable distirbutions for successively greater number of samples as seen in Figures 16, 17, and 18.

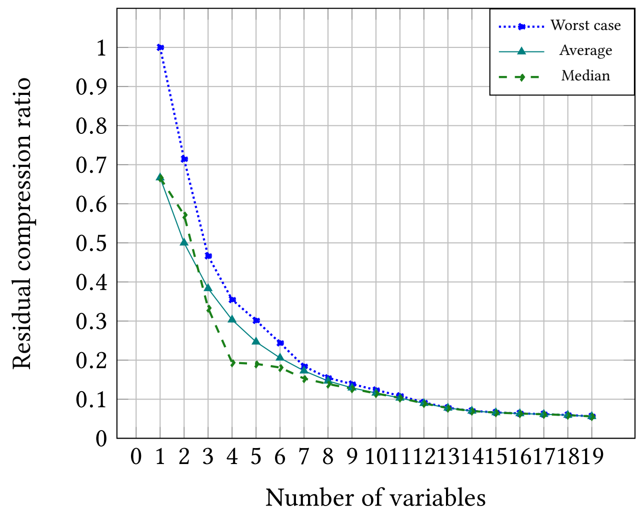


Fig. 20. Residual compression ratio of ROBDD as compared to UOBDD

3.5 Measuring ROBDD residual compression

In Section 2.2, a 31 node UOBDD of 4 Boolean variables was reduced to an equivalent ROBDD with 8 nodes, meaning a residual compression ratio of $8/31 \approx 25.8\%$. The question was posed as to how typical this reduction is. Figure 20 shows a plot of the worst-case, average, and median sizes divided by the size of the UOBDD. The figure shows the residual compression ratio,

$$\rho_n = \frac{|ROBDD_n|}{|UOBDD_n|}, \tag{2}$$

for sizes $n = 1$ through $n = 10$ Boolean variables. The residual compression ratio quantifies which portion of the original size remains after converting a UOBDD into an ROBDD. The closer to zero, the better the compression.

The points in the plot are calculated by starting with the numbers from Figure 14 and dividing each by the size of the UOBDD. A UOBDD of n Boolean variables (as well as a full binary tree of n levels and 2^n leaves) has $|UOBDD_n| = 2^{n+1} - 1$ nodes. It appears from the plot that the residual compression ratio improves (the percentage decreases) as the number of variables increases. It is not clear from the plot what the asymptotic residual compression ratio is, but it appears from experimental data to be less than 15%. It would also appear that whether the residual compression ratio is measured using the average size or worst-case size, the difference is negligible as the number of variables increases.

In Section 3.6, we derive a formula for the worst-case ROBDD size as a function of the number of Boolean variables. In order to do that, we need to understand the topology of such ROBDDs. What are the connectivity invariants which control the shape? In this section, we examine some example worst-case ROBDDs. Section 4 discusses an algorithm for constructing such ROBDDs.

Figure 21 shows examples of worst-case ROBDD for 1 through 7 variables. Those ROBDDs have 3, 5, 7, 11, 19, 31, and 47 nodes respectively.

The 2-variable ROBDD represents the Boolean expression $((Z_1 \wedge \neg Z_2) \vee (\neg Z_1 \wedge Z_2))$, which is the `xor` function. We did not recognize any obvious pattern in the Boolean expressions for the cases of 3 variables or more. As will become clear in Section 4 and in Algorithm 2, the worst-case ROBDD is not unique. There is considerable flexibility in constructing it. One may naturally wonder whether

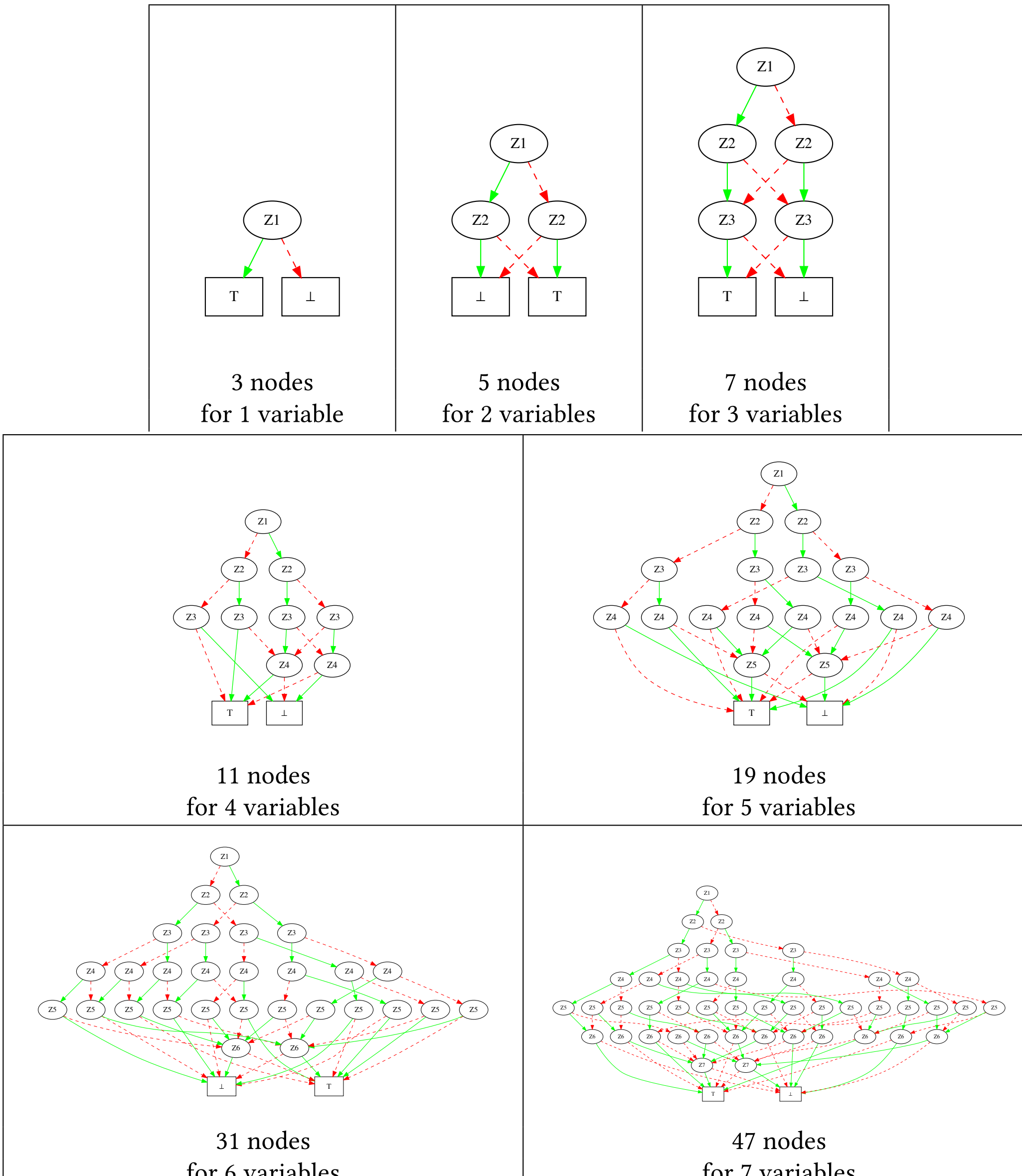


Fig. 21. Shapes of worst-case ROBDDs for 1 to 7 variables

there is some underlying pattern within the Boolean expressions corresponding to these worst-case ROBDDs. We have not investigated this question yet, and leave it open for further investigation.

Even if there is no obvious pattern among the closed form Boolean expressions, we do notice a general pattern in the overall shapes of the worst-case ROBDDs, as we increase the number of variables. We will make this pattern explicit in Section 3.6, but intuitively, it seems that the shapes expand from the top (root node) to somewhere close to mid-way down and thereafter contract toward the bottom, always ending with two rows having exactly two nodes each.

This shape makes sense because the maximum possible expansion (top toward bottom) occurs when each row contains twice as many nodes as the row directly above it. Each node in the i^{th} row corresponding to variable Z_i has two arrows (one positive and one negative) pointing to nodes of variable Z_{i+1}. If the i^{th} row is to have the maximum number of nodes possible, then no node may be symmetric, otherwise the node could be eliminated by the Deletion rule. Furthermore, no two nodes may be congruent, otherwise one of the nodes could be eliminated by the Merging rule.

However, this exponential expansion is limited by the fact that the bottommost row must contain exactly two leaf nodes in worst case, corresponding to $\top$, and $\bot$. We know this because if the bottom row had only one of $\top$ or $\bot$, then any node in the second to last row would be symmetric, having its positive and negative arrows pointing to this same leaf node. Such a node would be eliminated by the Deletion rule. Thus, the second to last row would be empty. From this we can conclude that if an ROBDD has exactly one leaf, it also has exactly one node. Such an ROBDD is obviously not a worst-case ROBDD.

We know from the previous argument that the bottommost row has exactly two leaves. That being the case, if the second to last row had any symmetric node, such a node would be removed by the Deletion rule. Furthermore, if any two nodes in the row were congruent, one of the nodes would be eliminated by the Merging rule. Therefore, as worst case, there may be only as many nodes in the second to last row as there are ordered parings of the leaf nodes. There are only two such ordered pairs: $(\top, \bot)$ and $(\bot, \top)$. The second to last row has exactly two nodes.

A similar argument limits the third to last row, and the rows above it. In each such case, the number of nodes in the row is limited by the number of ordered pairs which can be formed by all the nodes below it, having no symmetric node and no congruent nodes. This implies a combinatorial expansion from bottom toward the top.

As argued above, there is an exponential growth from the topmost node downward, and there is a combinatorial growth from the bottommost node upward. At some point, the widest part of the graph, these two growth rates meet.

3.6 Worst-case ROBDD Size

In Section 3.2, we saw the worst-case sizes of ROBDDs for different numbers of Boolean variables. We observed an exponential top-down growth rate, a bottom-up combinatorial one, and a point somewhere in between where these two growths meet. In this section, we derive explicit formulas for these observations, and from them, derive the worst-case size, $|ROBDD_n|$.

As can be seen in Figure 21, the number of nodes per row (per variable), looking from the top down, is limited by 2^i where i is the index of the variable. The number of nodes per row follows the sequence $2^0 = 1, 2^1 = 2, 2^2 = 4, \ldots 2^k$.

The row corresponding to the last variable has two nodes, one with children $positive = \bot$, $negative = \top$ and one with $positive = \top$, $negative = \bot$. In the worst case, each row above the bottom has the number of nodes necessary for each node to uniquely connect its positive and negative arrows to some unique pair of nodes below it. The number of ordered pairs of m items is $m^{\underline{2}}$ (read m raised to the second power descending). Recall that $m^{\underline{a}} = \frac{m!}{(m-a)!}$ which, for the special case of $a = 2$, becomes $m^{\underline{2}} = \frac{m!}{(m-2)!} = m \cdot (m-1)$.

We denote the size of the k^{th} row of the worst-case ROBDD of n variables as ${}^nR_k$, and the total number of nodes of rows $k+1$ through n as ${}^nS_k$. In other words, ${}^nS_k$ is the number of nodes in the rows strictly below row k. Viewed from the bottom up, the sequence of rows have sizes ${}^nR_{n-1}$, ${}^nR_{n-2}$, ${}^nR_{n-3}$, *etc.* The number of nodes in row i is a function of the sum of the number of nodes in the rows below it, namely ${}^nR_i = {}^nS_i^{\underline{2}} = {}^nS_i \cdot ({}^nS_i - 1)$.

$$ {}^nS_n = {}^nR_{n+1} = 2 \qquad {}^nR_{n+1} = 2 $$

$$ {}^nS_{n-1} = {}^nR_{n+1} + {}^nR_n = 4 \qquad {}^nR_n = {}^nS_n{}^{\underline{2}} = 2 \cdot 1 = 2 $$

$$ {}^nS_{n-2} = {}^nR_{n+1} + \ldots + {}^nR_{n-1} = 16 \qquad {}^nR_{n-1} = {}^nS_{n-1}{}^{\underline{2}} = 4 \cdot 3 = 12 $$

$$ {}^nS_{n-3} = {}^nR_{n+1} + \ldots + {}^nR_{n-2} = 256 \qquad {}^nR_{n-2} = {}^nS_{n-2}{}^{\underline{2}} = 16 \cdot 15 = 240 $$

$$ {}^nR_{n-3} = {}^nS_{n-3}{}^{\underline{2}} = 256 \cdot 255 = 65280 $$

$$ {}^nS_{n-k} = \sum_{i=n-(k-1)}^{n+1} {}^nR_i \qquad {}^nR_{n-k} = {}^nS_{n-k}{}^{\underline{2}} \tag{3} $$

Fig. 22. The two interrelated sequences ${}^nS_i$ and ${}^nR_i$.

Notice that the bottom row of a non-trivial worst-case ROBDD has exactly 2 nodes, the $\top$ and $\bot$ nodes, thus ${}^nR_{n+1} = 2$. For each i, ${}^nS_i$ can be calculated as the sum of the previous ${}^nR_j$ for $j = n - i, \ldots, n + 1$. This is illustrated by the equations in Figure 22.

An interesting pattern emerges: ${}^nS_n = 2^{2^0}$, ${}^nS_{n-1} = 2^{2^1}$, ${}^nS_2 = 2^{2^2}$, ${}^nS_3 = 2^{2^3}$, suggesting Lemma 3.1.

Lemma 3.1. *Let*

$$ {}^nS_{n-k} = \sum_{i=n-(k-1)}^{n+1} {}^nR_i \,, $$

where

$$ {}^nR_{n+1} = 2 $$

and for $k > 1$,

$$ {}^nR_{n-k} = {}^nS_{n-k} \cdot ({}^nS_{n-k} - 1)\,. $$

Then for every positive integer k,

$$ {}^nS_{n-k} = 2^{2^k}\,. $$

Proof. By Induction: The initial case, $k = 0$ is that

$$ \begin{aligned} {}^nS_{n-k} &= {}^nS_{n-0} \\ &= {}^nR_{n+1} = 2 = 2^1 = 2^{2^0} = 2^{2^k}\,. \end{aligned} $$

It remains only to be shown that for $k \geq 0$, ${}^nS_{n-k} = 2^{2^k}$ implies ${}^nS_{n-(k+1)} = 2^{2^{k+1}}$. Assume

$$ {}^nS_{n-k} = 2^{2^k}\,. $$

It follows that

$$\begin{aligned}
{}^nS_{n-(k+1)} &= \sum_{i=n-k}^{n+1} {}^nR_i \\
&= {}^nR_{n-k} + \sum_{i=n-(k+1)}^{n+1} {}^nR_i \\
&= {}^nR_{n-k} + {}^nS_{n-k} \\
&= ({}^nS_{n-k}) \cdot ({}^nS_{n-k} - 1) + ({}^nS_{n-k}) \\
&= ({}^nS_{n-k}) \cdot ({}^nS_{n-k} - 1 + 1) \\
&= {}^nS_{n-k} \cdot {}^nS_{n-k} \\
&= ({}^nS_{n-k})^2 \\
&= (2^{2^k})^2 \quad = \quad 2^{2 \cdot 2^k} \quad = \quad 2^{2^{k+1}} .
\end{aligned}$$

□

Next, we show more concise forms for ${}^nR_{n-k}$ and ${}^nR_i$. As a convention, we will use the variable i to index rows and summations when counting from the top (root) node down. By contrast we will use the variable k to index rows and summations when counting from the bottom up.

Lemma 3.2. *If $k \geq 0$, then*

$$ {}^nR_{n-k} = 2^{2^{k+1}} - 2^{2^k} ,$$

and if $i \leq n$,

$$ {}^nR_i = 2^{2^{n-i+1}} - 2^{2^{n-i}} .$$

Proof.

$$\begin{aligned}
{}^nR_{n-k} &= {}^nS_{n-k}{}^{\underline{2}} && \text{by 3} \\
&= (2^{2^k}) \cdot (2^{2^k} - 1) && \text{by Lemma 3.1} \\
&= (2^{2^k} \cdot 2^{2^k}) - 2^{2^k} \\
&= (2^{2^k})^2 - 2^{2^k} \\
&= 2^{2 \cdot 2^k} - 2^{2^k}
\end{aligned}$$

$$ {}^nR_{n-k} = \begin{cases} 2^{2^{k+1}} - 2^{2^k} & \text{if } k \geq 0 \\ 2 & \text{if } k = -1 \end{cases}$$

$$ {}^nR_i = \begin{cases} 2^{2^{n-i+1}} - 2^{2^{n-i}} & \text{if } i \leq n \\ 2 & \text{if } i = n+1 \end{cases}$$

□

As explained already, ${}^nR_i$ is the number of elements which would fit into row i, only taking into consideration the combinatorial growth from the bottommost row up to row i. However, when looking from the topmost row down, taking into account the exponential growth only, the number of nodes in row i is given by

$$^{n}r_i = 2^{i-1} \tag{4}$$

$$^{n}r_{n-k} = 2^{n-k-1}\,. \tag{5}$$

Each row within the worst-case ROBDD is limited in size by the two terms, $^{n}R_i$ and $^{n}r_i$. The precise number of nodes in each row is the minimum of these two terms. The total number of nodes in a worst-case ROBDD of n variables is the sum of the number of nodes in each of its rows, given by Equation 7 which holds when $n > 1$.

$$|ROBDD_n| = 2 + \sum_{i=1}^{n} \min\{^{n}r_i, {}^{n}R_i\} \tag{6}$$

$$= 2 + \sum_{i=1}^{n} \min\{2^{i-1}, 2^{2^{n-i+1}} - 2^{2^{n-i}}\} \tag{7}$$

Theorem 3.3 is stated and proven now. This theorem is useful in the later discussion of Algorithm 2.

Theorem 3.3. *Every row of a worst-case ROBDD, except the first row, has an even number of nodes.*

Proof. The i'th row of an n-variable ROBDD either has $^{n}r_i$ nodes or $^{n}R_i$ nodes. If $i > 1$, then $^{n}r_i = 2^{i-1}$ (by Equation 4) is even. If $1 < i \leq n$, then $^{n}R_i = 2^{2^{n-i+1}} - 2^{2^{n-i}}$ (by Lemma 3.2) is even. The final case is when $i = n + 1$, the row of terminal nodes, $^{n}R_{n+1} = 2$ which is even. □

In Section 3.6 we derived the sizes of the rows of the worst-case ROBDD. We also derived Equation 7 which is easy to state but difficult to evaluate, which gives the value of $|ROBDD_n|$ in terms of the sum of the row sizes. In this section we will build up to and prove Theorem 3.9 which makes a step forward in making this value easier to calculate.

Corollary 3.4. *A worst-case ROBDD as an odd number of nodes.*

We could prove Corollary 3.4 by straightforward examination of Equation 7, and we would reach the same conclusion as the following simpler argument.

Proof. The first row of any ROBDD (worst-case or not) has a single node. By Theorem 3.3 every row thereafter of a worst-case ROBDD has an even number of nodes. Therefore the the total number of nodes is necessarily odd. □

There is a shortcut for calculating the sum in Equation 7. To make the shortcut more evident, first consider the example where $n = 10$, and calculate the size of row $i = 1$. To calculate $\min\{2^{1-1}, 2^{2^{10-1+1}} - 2^{2^{10-1}}\} = \min\{2^0, 2^{2^{10}} - 2^{2^9}\} = 1$, there is no need to calculate the 1024 digits of $2^{2^{10}} - 2^{2^9}$, because $2^0 = 1$ is obviously smaller. The trick is to realize that the summation (Equation 7) is the sum of leading terms of the form 2^i, plus the sum of trailing terms of the form $2^{2^{n-i+1}} - 2^{2^{n-i}}$. How many leading and trailing terms may not be obvious however. Theorem 3.9 shows the existence of the so-called threshold function, θ, which makes these two sums explicit.

3.7 The threshold function θ

In this section, we prove the existence of the so-called threshold function, θ, and express $|ROBDD_n|$ in terms of θ (Theorem 3.9). Before proving that lemma, we establish a few intermediate results to simplify later calculations.

Lemma 3.5. *If $f : \mathbb{R} \to \mathbb{R}$ is differentiable, then*

$$\frac{d}{dx}2^{f(x)} = 2^{f(x)} \cdot \ln 2 \cdot \frac{d}{dx}f(x)$$

Proof.

$$\begin{aligned}\frac{d}{dx}2^{f(x)} &= \frac{d}{dx}e^{\ln 2^{f(x)}} = \frac{d}{dx}e^{f(x)\cdot\ln 2}\\ &= e^{f(x)\cdot \ln 2} \cdot \ln 2 \cdot \frac{d}{dx}f(x)\\ &= e^{\ln 2^{f(x)}} \cdot \ln 2 \cdot \frac{d}{dx}f(x)\\ &= 2^{f(x)} \cdot \ln 2 \cdot \frac{d}{dx}f(x)\end{aligned}$$

□

Lemma 3.6. *If $f : \mathbb{R} \to \mathbb{R}$ is differentiable, then*

$$\frac{d}{dx}2^{2^{f(x)}} = 2^{f(x)+2^{f(x)}} \cdot \ln 4 \cdot \frac{d}{dx}f(x)$$

Proof. Change of variables, let $g(x) = 2^{f(x)}$, and use Lemma 3.5 twice.

$$\begin{aligned}\frac{d}{dx}2^{2^{f(x)}} &= \frac{d}{dx}2^{g(x)} = 2^{g(x)} \cdot \ln 2 \cdot \frac{d}{dx}g(x)\\ &= \frac{d}{dx}2^{f(x)} \cdot \ln 2 \cdot 2^{2^{f(x)}}\\ &= \left(2^{f(x)} \cdot \ln 2 \cdot \frac{d}{dx}f(x)\right) \cdot \left(\ln 2 \cdot 2^{2^{f(x)}}\right)\\ &= 2^{f(x)+2^{f(x)}} \cdot \ln 4 \cdot \frac{d}{dx}f(x)\end{aligned}$$

□

Even though Lemma 3.7 is trivial to prove, we provide it because it removes redundant steps in proving Lemmas 3.8 and 3.10.

Lemma 3.7. *If $h : \mathbb{R} \to \mathbb{R}$, then $2^{h(x)+1+2^{h(x)+1}} > 2^{h(x)+2^{h(x)}}$.*

Proof.

$$\begin{aligned}h(x) + 1 > h(x) &\implies 2^{h(x)+1} > 2^{h(x)}\\ &\implies h(x) + 1 + 2^{h(x)+1} > h(x) + 2^{h(x)}\\ &\implies 2^{h(x)+1+2^{h(x)+1}} > 2^{h(x)+2^{h(x)}}\end{aligned}$$

□

Lemma 3.8. *The function, $f(x) = 2^{2^{n-x+1}} - 2^{2^{n-x}}$ is decreasing.*

Proof. To show that f is decreasing, we show that $\frac{d}{dx}f(x) < 0$.

$$
\begin{aligned}
\frac{d}{dx} f(x) &= \frac{d}{dx}\left(2^{2^{n-x+1}} - 2^{2^{n-x}}\right) \\
&= 2^{n-x+1+2^{n-x+1}} \cdot \ln 4 \cdot (-1) - 2^{n-x+2^{n-x}} \cdot \ln 4 \cdot (-1) && \text{by Lemma 3.6} \\
&= \left(2^{n-x+1+2^{n-x+1}} - 2^{n-x+2^{n-x}}\right) \cdot \ln 4 \cdot (-1) \\
&= \left(2^{n-x+2^{n-x}} - 2^{n-x+1+2^{n-x+1}}\right) \cdot \ln 4
\end{aligned}
$$

Letting $h(x) = n - x$, and applying Lemma 3.7, we have $2^{n-x+2^{n-x}} < 2^{n-x+1+2^{n-x+1}}$. So

$$\left(2^{n-x+2^{n-x}} - 2^{n-x+1+2^{n-x+1}}\right) \cdot \ln 4 < 0\,.$$

□

The following theorem proves the existence of the threshold function θ, without giving insight into how to calculate it. See Section 3.10 for a discussion on how to calculate it.

Theorem 3.9. *For each $n > 0$, there exists an integer θ, such that*

$$|ROBDD_n| = (2^{n-\theta} - 1) + 2^{2^\theta}\,.$$

Proof. As i increases, so does ${}^n r_i = 2^{i-1}$. By Lemma 3.8, ${}^n R_i = 2^{2^{n-i+1}} - 2^{2^{n-i}}$ is decreasing (as a function of i). At $i = 0$, $2^{i-1} < 2^{2^{n-i+1}} - 2^{2^{n-i}}$. So there necessarily exists a χ_n such that when $i < \chi_n$ we have ${}^n r_i < {}^n R_i$, and when $i \geq \chi_n$ we have ${}^n r_i \geq {}^n R_i$.

$$
\begin{aligned}
|ROBDD_n| &= 2 + \sum_{i=1}^{n} \min\{{}^n r_i, {}^n R_i\} && \text{by 6} \\
&= 2 + \sum_{i=1}^{\chi_n - 1} {}^n r_i + \sum_{i=\chi_n}^{n} {}^n R_i
\end{aligned}
$$

Now, we define $\theta_n = n - \chi_n + 1$, *i.e.*, the number of terms in the second sum. We also adjust the iteration variable of the second summation to commence at 0. Finally, we apply Lemma 3.2. Simply as a matter of notation, and to facility ease of reading, we will write θ rather than θ_n.

$$
\begin{aligned}
|ROBDD_n| &= 2 + \sum_{i=1}^{n-\theta} {}^n r_i + \sum_{i=n-\theta+1}^{n} {}^n R_i \\
&= 2 + \sum_{i=1}^{n-\theta} {}^n r_i + \sum_{k=0}^{\theta-1} {}^n R_{n-k} \\
&= 2 + \sum_{i=1}^{n-\theta} 2^{i-1} + \sum_{k=0}^{\theta-1} (2^{2^{k+1}} - 2^{2^k})
\end{aligned}
$$

Notice that $\sum_{i=1}^{n-\theta} 2^{i-1}$ is a truncated geometric series whose sum is $2^{n-\theta}-1$. Furthermore, $\sum_{k=0}^{\theta_n-1}(2^{2^{k+1}} - 2^{2^k})$ is a telescoping series for which all adjacent terms cancel, leaving the difference $2^{2^\theta} - 2^{2^0} = 2^{2^\theta} - 2$. This leads to the desired equality.

$$
\begin{aligned}
|ROBDD_n| &= 2 + (2^{n-\theta} - 1) + (2^{2^\theta} - 2) \\
&= (2^{n-\theta} - 1) + 2^{2^\theta}
\end{aligned}
$$

□

This result makes sense intuitively. The $(2^{n-\theta} - 1)$ term represents the exponential growth of the ROBDD seen in the top rows, from row 1 to row $n - \theta$, as can be seen in the illustrations such as Figure 21. The $2^{2^{\theta}}$ term represents the double-exponential decay in the bottom rows as can be seen in the same illustration.

Another way to think of θ is as follows. We define the integer sequence θ_n as the corresponding values of the real valued function

$$\theta_n = \lfloor \psi(n) \rfloor \,, \tag{8}$$

where $\psi : \mathbb{R}^+ \mapsto \mathbb{R}$ such that

$$2^{2^{\psi(n)+1}} - 2^{2^{\psi(n)}} = 2^{n-\psi(n)-1} \,. \tag{9}$$

Equation 9 is the real number extension of the integer equation ${}^{n}r_{\theta_n} = {}^{n}R_{\theta_n}$. Clearly, θ and ψ are functions of n, hence we denote them as such. We will, as before, dispense with the notation when it is clear, and simply refer to θ_n as θ, and $\psi(n)$ as ψ.

Although we do not attempt to express θ in closed form as a function of n, we do know several things about that function. For example we see in Theorem 3.11 that θ is non-decreasing. We also see in Theorems 3.13 and 3.14 that θ is bounded above and below by functions which themselves go to infinity. Thus, θ becomes arbitrarily large (Equation 22).

That $\theta = \lfloor \psi \rfloor$, means that θ is the integer such that $n - \theta$ is the maximum integer for which

$${}^{n}r_{n-\theta} \leq {}^{n}R_{n-\theta} \,. \tag{10}$$

If $n - \theta$ is the maximum such integer, then

$${}^{n}r_{n-\theta+1} > {}^{n}R_{n-\theta+1} \,. \tag{11}$$

As an example, consider the case of $n = 3$.

$$\begin{aligned} {}^{3}r_2 &= 2 < {}^{3}R_2 = 12 \\ {}^{3}r_3 &= 4 > {}^{3}R_3 = 2 \end{aligned}$$

We see that ${}^{3}r_2$ is the largest value of ${}^{3}r_i$ which is less than ${}^{3}R_i$. So we have $n - \theta = 3 - \theta = 2$, or $\theta = 1$. If we look at the case of $n = 2$ we see why Inequality 10 is not a strict inequality.

$$\begin{aligned} {}^{2}r_1 &= 1 < {}^{2}R_1 = 12 \\ {}^{2}r_2 &= 2 \leq {}^{2}R_2 = 2 \\ {}^{2}r_3 &= 4 > {}^{2}R_3 = 2 \end{aligned}$$

This can be seen in Figure 21, in which the worst-case ROBDD for $n = 2$ has two nodes for Z_2. There are two nodes for two reasons: because $2^{2-1} = 2$ and also because $2^{\underline{2}} = 2$.

3.8 The threshold function is non-decreasing

This section establishes that θ (defined by Equation 8) is a non-decreasing sequence. In Section 3.9, we will show by Theorems 3.13 and 3.14 that θ is bounded above and below by increasing functions. However, this alone is not sufficient to show that θ itself is non-decreasing.

To show θ is non-decreasing (Theorem 3.11), we first show that ψ, as defined by Equation 9, is strictly increasing (Lemma 3.10). To prove Lemma 3.10, we need two identities, proven earlier in Lemmas 3.5 and 3.6.

LEMMA 3.10. *$\psi : \mathbb{R}^+ \mapsto \mathbb{R}$ is strictly increasing.*

Proof. To show that ψ is increasing, we show that its derivative, $\frac{d}{dx}\psi(x)$, is strictly positive. We use x as the variable of integration rather than n to emphasize that the domain of ψ is $\mathbb{R}^+$ not $\mathbb{N}$. Note that it is not actually necessary to calculate the derivative of ψ in a form independent of ψ. Rather, it suffices to show that the derivative is positive. We find an expression for $\frac{d}{dx}\psi(x)$ in terms of $\psi(x)$ using implicit differentiation.

$$2^{2^{\psi(x)+1}} - 2^{2^{\psi(x)}} = 2^{x-\psi(x)-1}$$
$$\frac{d}{dx}2^{2^{\psi(x)+1}} - \frac{d}{dx}2^{2^{\psi(x)}} = \frac{d}{dx}2^{x-\psi(x)-1} \tag{12}$$

For clarity, we calculate these three derivatives separately. Applications of Lemma 3.5 and Lemma 3.6 lead to:

$$\frac{d}{dx}2^{2^{\psi+1}} = 2^{\psi+1+2^{\psi+1}} \cdot \ln 4 \cdot \frac{d\psi}{dx} \tag{13}$$
$$\frac{d}{dx}2^{2^{\psi}} = 2^{\psi+2^{\psi}} \cdot \ln 4 \cdot \frac{d\psi}{dx} \tag{14}$$
$$\frac{d}{dx}2^{x-\psi-1} = 2^{x-\psi-1} \cdot \ln 2 \cdot (1 - \frac{d\psi}{dx}) \tag{15}$$

Substituting 13, 14, and 15 into 12, and solving for $\frac{d\psi}{dx}$ results in

$$\frac{d\psi}{dx} = \frac{2^{x-\psi-1}}{\ln 2 \cdot (2^{\psi+1+2^{\psi+1}} - 2^{\psi+2^{\psi}}) + 2^{x-\psi-1}} . \tag{16}$$

Since the right hand side of Equation 16 is a fraction whose numerator, $2^{x-\psi-1}$, is positive, and whose denominator is the sum of two terms, the second of which, $2^{x-\psi-1}$, is positive, then it suffices to argue that the first term in the denominator, $\ln 2 \cdot (2^{\psi+1+2^{\psi+1}} - 2^{\psi+2^{\psi}})$, is positive. If we let $h(x) = \psi(x) + 1$, then Lemma 3.7 implies $2^{\psi+1+2^{\psi+1}} > 2^{\psi+2^{\psi}}$. So since $\ln 2 > 0$, we conclude that $\ln 2 \cdot (2^{\psi+1+2^{\psi+1}} - 2^{\psi+2^{\psi}}) > 0$.

$$\frac{d\psi}{dx} = \frac{\overbrace{2^{x-\psi-1}}^{>0}}{\underbrace{\ln 2}_{>0} \cdot \underbrace{(2^{\psi+1+2^{\psi+1}} - 2^{\psi+2^{\psi}})}_{\psi+1+2^{\psi+1} > \psi+2^{\psi}} + \underbrace{2^{x-\psi-1}}_{>0}} > 0 .$$

□

Theorem 3.11. *$\theta : \mathbb{N} \mapsto \mathbb{N}$ by $\theta_n = \lfloor \psi(n) \rfloor$ is non-decreasing.*

Proof. $\psi : \mathbb{R}^+ \mapsto \mathbb{R}$ is increasing (Lemma 3.10), implies that if $m \in \mathbb{N}$, then $\psi(m+1) > \psi(m)$. Thus $\lfloor \psi(m+1) \rfloor \geq \lfloor \psi(m) \rfloor$; *i.e.*, $\theta_{m+1} \geq \theta_m$ holds for all $m \in \mathbb{N}$. □

3.9 Bounds for the threshold function

We showed in Theorem 3.9 that the function θ is well defined, but we didn't say how to calculate it. We now show that θ is bounded by two logarithmic functions. Using those bounds, we will then develop an efficient algorithm for calculating it iteratively (Section 3.10). To do this, we first establish an inequality (Lemma 3.12) to be used later.

Lemma 3.12. *For any real number $\alpha > 0$, we have*

$$2^{2^\alpha} < 2^{2^{\alpha+1}} - 2^{2^\alpha}.$$

Proof.

$$\begin{aligned}
1 = 2^0 &< 2^\alpha \\
2 = 2^1 &< 2^{2^\alpha} \\
1 &< 2^{2^\alpha} - 1 \\
&= 2^{(2-1)\cdot 2^\alpha} - 1 \\
&= 2^{2\cdot 2^\alpha - 2^\alpha} - 1 \\
&= 2^{2^{\alpha+1} - 2^\alpha} - 1 \\
\frac{2^{2^\alpha}}{2^{2^\alpha}} &< \frac{2^{2^{\alpha+1}}}{2^{2^\alpha}} - \frac{2^{2^\alpha}}{2^{2^\alpha}} \\
2^{2^\alpha} &< 2^{2^{\alpha+1}} - 2^{2^\alpha}
\end{aligned}$$

□

We now establish an upper bound for θ.

Theorem 3.13. *For any $n \in \mathbb{N}$, we have*

$$\theta_n < \log_2 n.$$

Proof.

$$\begin{aligned}
{}^nR_{n-\theta+1} &< {}^nr_{n-\theta+1} && \text{by 11} \\
2^{2^{\theta+1}} - 2^{2^\theta} &< 2^{n-\theta} \\
2^{2^\theta} < 2^{2^{\theta+1}} - 2^{2^\theta} &< 2^{n-\theta} && \text{by Lemma 3.12} \\
2^\theta &< n - \theta < n \\
\theta &< \log_2 n
\end{aligned}$$

□

We now establish a lower bound for θ.

Theorem 3.14. *For any $n \in \mathbb{N}$, we have*

$$\log_2(n - 2 - \log_2 n) - 2 \le \theta.$$

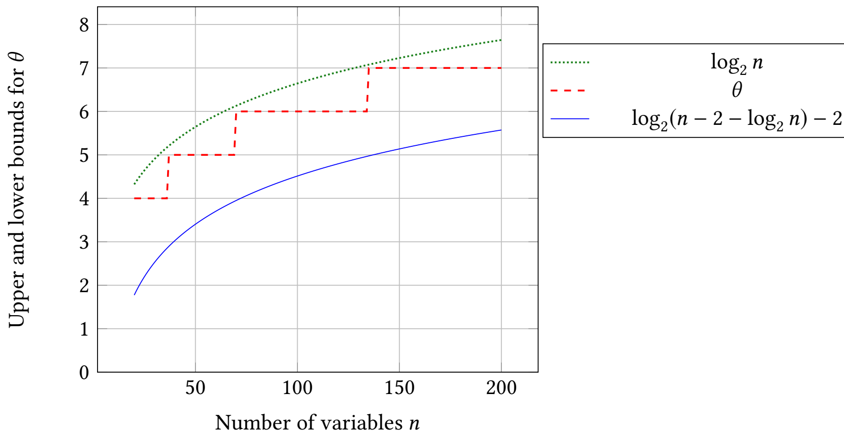


Fig. 23. Upper and lower bounds for θ

Proof.

$$
\begin{aligned}
\psi - 1 \leq \lfloor \psi \rfloor = \theta &< \log_2 n \\
\psi &< 1 + \log_2 n && (17) \\
\psi + 1 &< \theta + 2 && (18) \\
2^{n-2-\log_2 n} &= 2^{n-(1+\log_2 n)-1} && \text{by 5 and Lemma 3.2} \quad (19) \\
&< 2^{n-\psi-1} && \text{by 17} \\
&= 2^{2^{\psi+1}} - 2^{2^{\psi}} && \text{by 9} \\
&< 2^{2^{\psi+1}} \\
&< 2^{2^{\theta+2}} && \text{by 18} \\
2^{\theta+2} &> n - 2 - \log_2 n \\
\theta &> \log_2(n-2-\log_2 n) - 2 && (20)
\end{aligned}
$$

□

As a consequence of Theorems 3.13 and 3.14, Corollary 3.15 defines upper and lower bounds for θ. The continuous, real valued bounds are illustrated in Figure 23.

Corollary 3.15. *For any $n \in \mathbb{N}$, we have*

$$\lceil \log_2(n-2-\log_2 n) \rceil - 2 \leq \theta \leq \lfloor \log_2 n \rfloor$$

Proof. From Theorems 3.13 and 3.14 we already have

$$\log_2(n-2-\log_2 n) - 2 \leq \theta \leq \log_2 n\,,$$

but since θ is an integer, the inequality implies

$$\lceil \log_2(n-2-\log_2 n) \rceil - 2 \leq \theta \leq \lfloor \log_2 n \rfloor$$

□

As is implied by Figure 23, and as proven in Theorem 3.16, $\theta \to \infty$.

Theorem 3.16.

$$\lim_{n\to\infty} \theta_n = \infty$$

Proof. First note that for $n \gg 0$

$$\log_2 n < \frac{n}{2}\,. \tag{21}$$

Next, we have a lower bound for θ,

$$\begin{aligned}
\theta_n &\geq \log_2(n - 2 - \log_2 n) - 2 && \text{by } 20\\
\lim_{n\to\infty} \theta_n &\geq \lim_{n\to\infty} \log_2(n - 2 - \log_2 n) - 2\\
&\geq \lim_{n\to\infty} \log_2(n - 2 - \frac{n}{2}) - 2 && \text{by } 21\\
&= \lim_{n\to\infty} \log_2(\frac{n}{2} - 2) - 2\\
&= \infty
\end{aligned} \tag{22}$$

□

Corollary 3.17.

$$\lim_{n\to\infty} \psi(n) = \infty\,.$$

Proof. Since

$$\theta_n = \lfloor \psi(n) \rfloor \leq \psi(n) \leq \lceil \psi(n) \rceil \leq 1 + \theta_n\,,$$

then by application of Theorem 3.16 we have

$$\begin{aligned}
\infty = \lim_{n\to\infty} \theta_n &\leq \lim_{n\to\infty} \psi(n)\\
&\leq \lim_{n\to\infty} 1 + \theta_n = \infty\,.
\end{aligned}$$

So

$$\lim_{n\to\infty} \psi(n) = \infty\,. \tag{23}$$

□

3.10 Computing the threshold function

For a given n, the value of θ can be found iteratively, as shown in Algorithm 1. Initializing θ to the upper bound $\lfloor \log_2(n) \rfloor$ as initial guess, from Corollary 3.15, we continue to decrement θ as long as $2^{2^{\theta+1}} - 2^{2^{\theta}} < 2^{n-\theta-1}$. This iteration seems to usually terminate after 2 iterations. When running Algorithm 1 from $n = 2$ to $n = 200001$, it terminates after 3 iterations 152 times, and after 2 iterations 199848 times (99.92%). Table 24 shows the values of θ for $1 \leq n \leq 21$ as calculated by Algorithm 1.

Algorithm 1 terminates in about two iterations, which makes sense when considering Theorem 3.18. We see in that theorem that for large n the difference of the upper and lower limits expressed in Corollary 3.15 is 2. However, we see from experimentation that a small fraction of the time the algorithm terminates at 3 iterations. This is because Algorithm 1 arranges that θ is always decremented once too many (except when $n = 1$). This is why $\theta + 1$ is returned on line 1.10 of Algorithm 1.

Theorem 3.18. *For all sufficiently large n,*

$$\log_2 n - \theta_n < 2\,.$$

n	$\lfloor \log_2 n \rfloor$	θ	$2^{n-\theta} - 1 + 2^{2^\theta}$	n	$\lfloor \log_2 n \rfloor$	θ	$2^{n-\theta} - 1 + 2^{2^\theta}$
1	0	0	3	20	4	4	131,071
2	1	1	5	30	4	4	67,174,399
3	1	1	7	50	5	5	3.52×10^{31}
4	2	1	11	100	6	6	1.98×10^{28}
5	2	1	19	200	7	7	1.26×10^{58}
6	2	2	31	500	8	8	1.28×10^{148}
7	2	2	47	1000	9	9	2.09×10^{298}
8	3	2	79	2000	10	10	1.12×10^{599}
9	3	2	143	5000	12	12	3.45×10^{1501}
10	3	2	271	10,000	13	13	2.43×10^{3006}
11	3	3	511	20,000	14	14	2.42×10^{6016}

Fig. 24. Worst-case ROBDD size, $|ROBDD_n|$, in terms of number of variables, n. The table also shows θ (the threshold) and $\lfloor \log_2 n \rfloor$ demonstrating that $\lfloor \log_2 n \rfloor$ serves both as an upper bound and as an initial guess for θ. The table also shows the exponential term and the double-exponential term, whose sum is the worst-case size.

Proof. We know that for $n \gg 0$, θ_n lies between the upper and lower bounds indicated in Theorems 3.13 and 3.14. This means

$$\log_2 n - \theta_n < \log_2 n - (\log_2(n - 2 - \log_2 n) - 2)\,.$$

$$\begin{aligned}
\Delta_{bounds} &= \lim_{n\to\infty} \Bigg(\overbrace{\log_2 n}^{\text{upper bound}} - \underbrace{\big(\log_2(n-2-\log_2 n) - 2\big)}_{\text{lower bound}} \Bigg) \\
&= 2 + \lim_{n\to\infty} \log_2 \frac{n}{n - 2 - \log_2 n} \\
&= 2 + \log_2 \lim_{n\to\infty} \frac{n}{n-2-\log_2 n} = 2 + \log_2 \lim_{n\to\infty} \overbrace{\frac{\frac{d}{dn} n}{\frac{d}{dn}(n - 2 - \log_2 n)}}^{\text{L'Hôpital's rule}} \\
&= 2 + \log_2 \lim_{n\to\infty} \frac{1}{1 - \frac{1}{n}} = 2 + \log_2 1 = 2
\end{aligned}$$

□

3.11 Plots of $|ROBDD_n|$ and related quantities

Now that we can calculate θ, it is possible to plot $|ROBDD_n|$ as a function of n. The plots in Figure 25 show the relative sizes of $2^{n-\theta}$, 2^{2^θ}, and their sum $|ROBDD_n|$. The plot also shows 2^n, which is intended to convey intuition about relative sizes of the various quantities. In the plot on the right, it appears that $2^{n-\theta}$ becomes a good approximation for $|ROBDD_n|$ for large values of n. However, the plot on the left shows that this is a poor approximation for values of n below 15.

Algorithm 1: FindTheta determine θ iteratively

Input: n: positive integer $n > 0$, indicating the number of Boolean variables

Output: θ: minimum integer, θ such that ${}^{n}r_{n-\theta} \le {}^{n}R_{n-\theta}$; *i.e.*, $2^{n-\theta} \le 2^{2^{\theta}} - 2^{2^{\theta-1}}$

begin
    if $n = 1$ **then**
        return 0
    $\theta \leftarrow \lfloor \log_2 n \rfloor + 1$
    repeat
        $\theta \leftarrow \theta - 1$
        $r \leftarrow 2^{n-\theta-1}$
        $R \leftarrow 2^{2^{\theta+1}} - 2^{2^{\theta}}$
    until $R < r$
    return $\theta + 1$

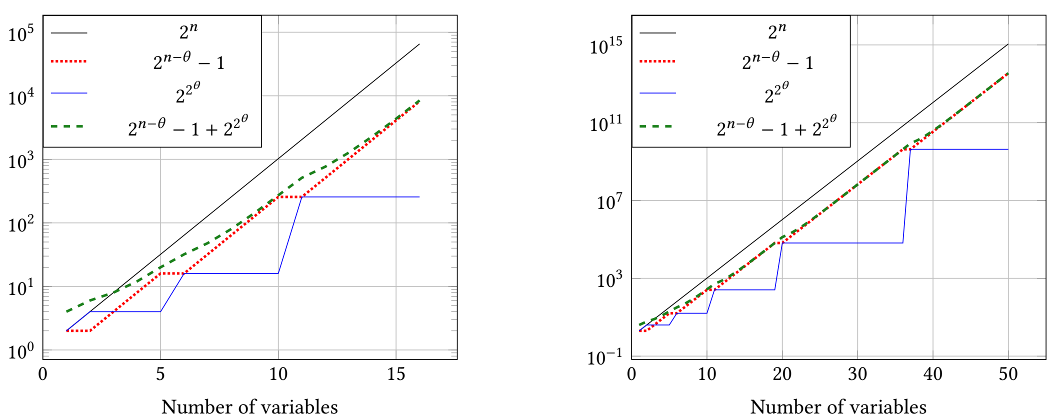


Fig. 25. The plots show the relative sizes of $2^{n-\theta}$, $2^{2^{\theta}}$, and their sum $|ROBDD_n|$.

3.12 Limit of the residual compression ratio

In Section 3.2, we introduced ρ_n, the ROBDD residual compression ratio (Equation 2). We also observed in Figure 20 that ρ_n seems to decrease as n increases. Moreover, Figure 26 shows ρ_n calculated by Equation 2 for values of $1 \le n \le 21$. The plot in Figure 26 shows the residual compression ratio for $1 \le n \le 200$. In this plot, it appears that the residual compression ratio tends to 0. This is in fact the case, as proven in Theorem 3.19.

Theorem 3.19.

$$\lim_{n\to\infty} \rho_n = 0 \,.$$

Proof. First, we establish a few helpful inequalities.

$$\begin{aligned} |UOBDD_n| &= 2^{n+1} - 1 && \text{by 1} \\ &> 2^n && \text{for } n \gg 0 \end{aligned} \tag{24}$$

$$
\begin{aligned}
2^{2^{\theta}} = 2^{2^{\lfloor \psi(n) \rfloor}} \leq 2^{2^{\psi(n)}} & & \text{by } 8 & \\
< 2^{2^{\psi(n)+1}} - 2^{2^{\psi(n)}} & & \text{by Lemma } 3.12 & \quad (25) \\
= 2^{n-\psi(n)-1} & & \text{by } 9 & \quad (26)
\end{aligned}
$$

$$
\begin{aligned}
|ROBDD_n| &= 2^{2^{\theta_n}} - 2^{n-\theta_n} - 1 & & \text{by Theorem } 3.9 & \\
&\leq 2^{n-\psi(n)-1} - 2^{n-\theta_n} - 1 & & \text{by } 26 & \quad (27)
\end{aligned}
$$

$$
\begin{aligned}
\rho_n &= \frac{|ROBDD_n|}{|UOBDD_n|} & & \text{by } 2 & \\
&< \frac{|ROBDD_n|}{2^n} & & \text{by } 24 & \\
&\leq \frac{2^{n-\psi(n)-1} - 2^{n-\theta_n} - 1}{2^n} & & \text{by } 27 & \quad (28)
\end{aligned}
$$

Now, we can apply the limit to Inequality 28.

$$
\begin{aligned}
\lim_{n\to\infty} \rho_n &\leq \lim_{n\to\infty} \frac{2^{n-\psi(n)-1} - 2^{n-\theta_n} - 1}{2^n} \\
&= \lim_{n\to\infty} \frac{2^{n-\psi(n)-1}}{2^n} - \lim_{n\to\infty} \frac{2^{n-\theta_n}}{2^n} - \lim_{n\to\infty} \frac{1}{2^n} \\
&= \lim_{n\to\infty} \frac{1}{2^{\psi(n)+1}} - \lim_{n\to\infty} \frac{1}{2^{\theta_n}} - \lim_{n\to\infty} \frac{1}{2^n} \\
&\leq 0 - 0 - 0 \qquad \text{by } 23 \text{ and } 22 \\
\lim_{n\to\infty} \rho_n &\leq 0\,.
\end{aligned}
$$

Since for each n, ρ_n is the quotient of two positive numbers, we know that $\rho_n > 0$. We can thus conclude that

$$\lim_{n\to\infty} \rho_n = 0\,.$$

□

4 PROGRAMMATIC CONSTRUCTION OF A WORST-CASE N-VARIABLE ROBDD

In the previous sections, we looked at various examples of ROBDDs of different sizes. During our experimentation, we found it necessary to devise an algorithm for generating worst-case ROBDDs. Because worst-case ROBDDs are not unique, any such algorithm has leeway in the manner it constructs them. In this section, we discuss the algorithm we developed, *i.e.*, an algorithm for constructing a worst-case ROBDD of n Boolean variables, denoted $Z_1, Z_2, \ldots Z_n$. The constructed ROBDDs resemble those shown in Figure 21.

Recall, from Section 3.6, that the worst-case ROBDD can be thought of as having two parts, which we will call the top part and the bottom part. This is illustrated in Figure 27. The top part comprises the exponential expansion from the root node (corresponding to Z_1) called row 0, and continuing until row $n - \theta - 1$. This top part contains $n - \theta$ number of rows. The bottom part comprises the double-exponential (*i.e.*, 2^{2^i}) decay, starting at row $n - \theta$, and continuing through row n for a total of $\theta + 1$ rows. The bottommost row is row n which contains precisely the two

n	$\|ROBDD_n\|$	ρ_n
1	3	100.000%
2	5	71.429%
3	7	46.667%
4	11	35.484%
5	19	30.159%
6	31	24.409%
7	47	18.431%
8	79	15.460%
9	143	13.978%
10	271	13.239%
11	511	12.479%
12	767	9.364%
13	1279	7.807%
14	2303	7.028%
15	4351	6.639%
16	8447	6.445%
17	16639	6.347%
18	33023	6.299%
19	65791	6.274%
20	131071	6.250%
21	196607	4.687%

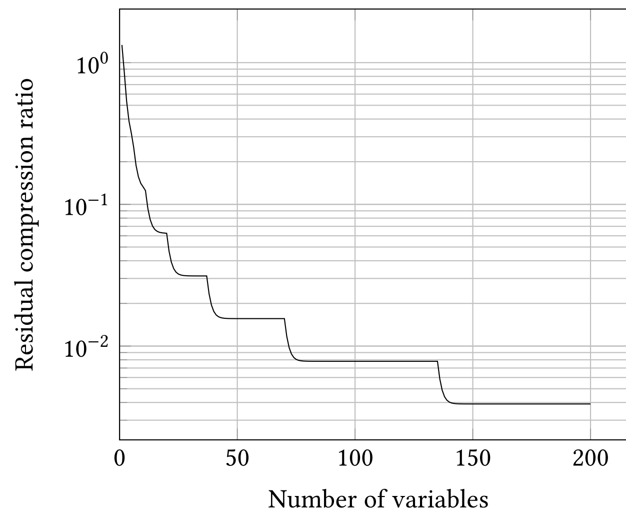


Fig. 26. Residual compression ratio of worst-case ROBDD, calculated from theoretical data as compared to UOBDD, and shown in tabular graphical form.

singleton objects $\top$ and $\bot$. From this information and the following few notational definitions, we are able to construct one of the many possible worst-case n-variable ROBDDs with Algorithm 2.

An ROBDD node is denoted as $node(\top)$ (true terminal node), $node(\bot)$ (false terminal node) or $node(i, \alpha, \beta)$ (non-terminal node on row_i with children nodes α and β.) Let row_a^b denote the set of rows a to b. If S is a set, let its cardinality be denoted $|S|$, and let $\mathscr{P}(S) = \{(\alpha, \beta) \mid \alpha, \beta \in S, \alpha \neq \beta\}$. Note that $|\mathscr{P}(S)| = |S|^{\underline{2}}$.

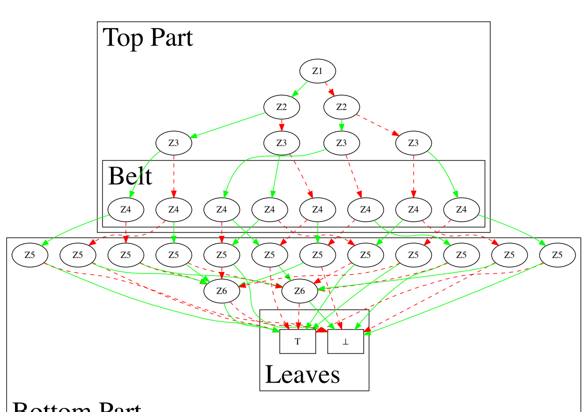


Fig. 27. 6-variable ROBDD top & bottom parts

Algorithm 2 generates an ROBDD represented as a vector of sets of nodes $[row_0, row_1, ..., row_n]$. Lines 2.4 through 2.5 generate the bottom part, and lines 2.7 through 2.10 generate the top part. Algorithm 3 generates the belt as illustrated in Figure 27.

Algorithm 2: GenWorstCaseROBDD generates a worst-case ROBDD

Input: n, positive integer indicating the number of Boolean variables
Output: a vector of sets of nodes

2.1 $\theta \leftarrow FindTheta(n)$ // Algorithm 1
2.2 $\mathcal{B} \leftarrow n - \theta - 1$ // belt row index
2.3 // Generate the bottom part
2.4 $\quad row_n \leftarrow \{node(\top), node(\bot)\}$ **for** i from $n-1$ downto $\mathcal{B}+1$ **do**
2.5 $\quad\quad row_i \leftarrow \{node(i, \alpha, \beta) \mid (\alpha, \beta) \in \mathscr{P}(row^n_{i+1})\}$ // $|row_i| = {}^nR_i$
2.6 // Generate the top part
2.7 $\quad row_{\mathcal{B}} \leftarrow GenBelt(n, \mathcal{B}, row)$ // Algorithm 3
2.8 $\quad$**for** i from $\mathcal{B}-1$ downto 0, **do**
2.9 $\quad\quad P \leftarrow$ any partition of row_{i+1} into ordered pairs // possible by Theorem 3.3, $|P| = \frac{|row_{i+1}|}{2} = 2^i$
2.10 $\quad\quad row_i \leftarrow \{node(i, \alpha, \beta) \mid (\alpha, \beta) \in P\}$ // $|row_i| = 2^i$
2.11 **return** $[row_0, row_1, ..., row_n]$ // generated $1 + (n - \mathcal{B} - 1) + 1 + \mathcal{B} = n + 1$ rows.

Algorithm 3: GenBelt generates the $\mathcal{B}$ row of the worst-case ROBDD

Input: n, positive integer indicating the number of Boolean variables
Input: $\mathcal{B}$, between 0 and n, indicating the belt's row number
Input: row, a vector which the calling function has partially populated. $row_{\mathcal{B}+1}...row_n$ are each non-empty sets of nodes.
Output: a set of $2^{\mathcal{B}}$ nodes intended to comprise $row_{\mathcal{B}}$

3.1 $p \leftarrow |row_{\mathcal{B}+1}|$ // calculate ${}^nR_{\mathcal{B}+1}$
3.2 $P_{left} \leftarrow$ any partition of $row_{\mathcal{B}+1}$ into ordered pairs // possible by Theorem 3.3
3.3 $S_{left} \leftarrow \{node(\mathcal{B}, \alpha, \beta) \mid (\alpha, \beta) \in P_{left}\}$
3.4 **if** $2^{\mathcal{B}} < p \cdot (p-1)$ **then** // if wide belt
3.5 $\quad P_{right} \leftarrow \mathscr{P}(row^n_{\beta+1})$
3.6 **else** // if narrow belt
3.7 $\quad P_{right} \leftarrow \mathscr{P}(row_{\beta+1})$
// We want $|row_{\mathcal{B}}| = 2^{\mathcal{B}}$. So limit $|S_{right}|$ to $2^{\mathcal{B}} - \frac{{}^nR_{\mathcal{B}+1}}{2}$.
3.8 $S_{right} \leftarrow$ any $\left(2^{\mathcal{B}} - |S_{left}|\right)$ sized subset of $\{node(\mathcal{B}, \alpha, \beta) \mid (\alpha, \beta) \in P_{right} \setminus P_{left}\}$
3.9 **return** $S_{left} \cup S_{right}$ // $|S_{left}| + |S_{right}| = \frac{{}^nR_{\mathcal{B}+1}}{2} + (2^{\mathcal{B}} - \frac{{}^nR_{\mathcal{B}+1}}{2}) = 2^{\mathcal{B}}$

For simplicity, we don't specify how to perform the computations on lines 2.9, 3.2, and 3.8. Solutions may vary, depending on the choice of programming language and data structures. Lines 2.9 and 3.2 call for a set with an even number of elements to be partitioned into pairs. Such a partitioning can be done in many different ways, one of those being

$$\{node_1, node_2...node_m\} \mapsto \{(node_1, node_2), (node_3, node_4)...(node_{m-1}, node_m)\}$$

Line 3.8 calls for the generation of any subset of a given size, and given a specified superset. In particular it asks for a subset,

$$S_{right} \subseteq \{node(\mathcal{B}, \alpha, \beta) \mid (\alpha, \beta) \in P_{right} \setminus P_{left}\}, \text{ such that } |S_{right}| = 2^{\mathcal{B}} - |S_{left}|.$$

One way to generate such a subset might be to first generate the superset, then truncate it to the desired size. A more clever way would be to start as if generating the superset, but stop once a sufficient number of elements is reached.

In Algorithm 3, there are two cases to consider. The question posed on line 3.4 is whether it is possible to generate $p^{\underline{2}} = p \cdot (p-1)$ unique ordered pairs of nodes from $row_{\mathcal{B}+1}$, possibly with some left over.

Wide: $2^{\mathcal{B}} > p \cdot (p-1)$ The belt is called wide because it touches not only the row directly below it, but others as well.
The Wide Belt case is illustrated in Figure 28. In this case $row_{\mathcal{B}}$ is row_3 (corresponding to Z_4) which has $2^3 = 8$ nodes. However, $row_{\mathcal{B}+1}$, *i.e.* row_4, (corresponding to Z_5) only has two nodes. There is an insufficient number of nodes in row_4 to connect the nodes in row_3. For this reason, connections are made, not only to $row_{\mathcal{B}+1}$, but also to some or all the nodes below it. Line 3.5 collects the set of all ordered pairs of nodes coming from $row_{\mathcal{B}+1}$ to row_n, and we will later (on line 3.8) want to subtract out those ordered pairs already collected in P_{left} to avoid generating congruent nodes. This set of ordered pairs might be large, so our suggestion is to generate a lazy set. Explanations of lazy data structures are numerous (Okasaki [Oka98] and Slade [Sla98, Section 14.6], to name a few).

Narrow: $2^{\mathcal{B}} \leq p \cdot (p-1)$ The belt is called narrow because unlike the wide belt, it only touches the row directly below it.
The Narrow Belt case is illustrated in Figure 28. In the figure we see that row_3, corresponding to variable Z_4, has 8 nodes, and $2^4 = 16$ arrows pointing downward. Since row_4 does not contain more than 16 nodes, it is possible to connect the belt to the bottom part simply by constructing the connecting arrows exclusively between row_3 and row_4 (between Z_4 and Z_5).

The time complexity of Algorithm 2 may vary, depending on the types of data structures used, and also according to which choices the programmers makes in implementing the set relative complement operation and truncated subset operations on line 3.8. However, in every case,. the algorithm must generate $|ROBDD_n|$ number of nodes. The complexity, therefore, cannot be made better than $\Omega(2^{n-\theta} + 2^{2^{\theta}})$, (we refer the reader to Wegener [Weg87, Section 1.5] for a discussion of Ω notation). The plots in Figure 25 convey an intuition of the relative sizes of $2^{n-\theta}$ vs $2^{2^{\theta}}$; *i.e.*

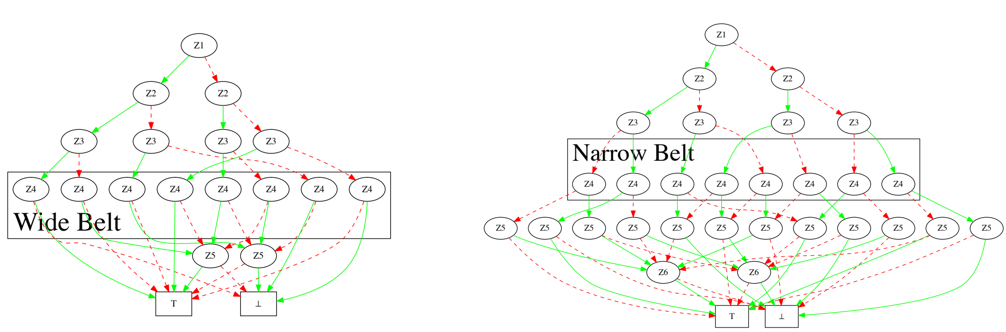


Fig. 28. Belt connections of the ROBDD for 5 and 6 variables. The narrow belt only connects to the row directly below it. The wide belt connects not only to the row below it, but also to other, lower rows.

that for large n, $2^{n-\theta} - 1$ becomes a good approximation for $|ROBDD_n|$. Thus we may approximate $\Omega(2^{n-\theta} + 2^{2^\theta}) \approx \Omega(2^{n-\theta})$.

5 RELATED WORK

Newton *et al.* [NVC17] discuss calculations related to the Common Lisp type system. These calculations are facilitated using ROBDDs, and there is an implication that Boolean expressions (describing Common Lisp types in this case) are efficient with this choice of data structure. However, no quantization is made in that work to justify the claim. In the current work we treat some of the questions relating to space efficiency using this choice of data structure.

Butler *et al.* [SIHB97] discuss average and worst-case sizes of BDDs representing multiple-valued functions. Miller *et al.* [MD03] also discusses the maximum size of discrete multiple-valued functions.

Bergman and Cire [BC16] discuss size bounds on data structures they call BDDs but which are defined slightly differently than we do. Bergman's data structures only support arcs connecting successive levels (which we call rows). That is, whereas we allow nodes in row i to connect to nodes in any row below it, Bergman only allows connections between rows i and $i+1$. Thus, in Bergman's case, BDDs such as the 4, 5, 6, and 7 variable cases we illustrate in Figure 21 are not considered. Nevertheless, we do find that our approach is similar to that of Bergman and Cire in that they both estimate the worst-case width of a row as the minimum of an exponential and a double-exponential, and then proceed to find the threshold where the exponential and double-exponential are equal.

The equation in Theorem 3.9 is similar to that provided by Knuth [Knu09, Section 7.1.4 Theorem U], and that provided by Heap *et al.* [HM94]. The derivation we show here relies heavily on intuitions gleaned from the shapes of worst-case ROBDDs, while the treatment of Knuth relies on a concept called beads which we do not address. The treatment by Heap is indeed similar to our own, albeit less grounded in geometric intuition. Heap's Theorem 1 gives a formula for $R(n)$ which differs by a constant of 2 from our Theorem 3.9. That difference is due to the fact that Heap does not include the two leaf nodes in his calculation as we do. Heap argues that the threshold (his k, our θ) is precisely $\lfloor \log_2 n \rfloor$ or $\lfloor \log_2 n \rfloor - 1$, which seems in keeping with our experimental findings from Algorithm 1 and Figure 24.

Gröpl *et al.* [GPS01] improved on the work of Heap by explaining certain oscillations shown but not explained in Heap's work. Additional work by Gröpl *et al.* [GPS98] look again into size of ROBDDs and discusses the Shannon effect, which explains the correlation between worst-case and average ROBDD sizes.

Minato [Min93] suggests using a different set of reduction rules than those discussed in Section 2. The resulting graph is referred to as a Zero-Suppressed BDD, or 0-Sup-BDDs (also referred to as ZDDs and ZBDDs in the literature). Minato claims that this data structure offers certain advantages over ROBDDs in modeling sets and expressing set-related operations, especially in sparse Boolean equations where the number of potential variables is large but the number of variables actually used in most equations is small. Additionally, Minato claims that 0-Sup-BDDs provide advantages when the number of input variables is unknown, which is the case we encounter when dealing with Common Lisp types, because we do not have a way of finding all user defined types.

Lingberg *et al.* [LPR03] consider sizes of ROBDDs representing the very special case of simple CNF formulas, in particular the representation of CNF formulas consisting of max-terms, each of which consists of exactly two variables, neither of which is negated. He does the majority of his development in terms of QOBDDs and relates back to ROBDDs with his claim that $|ROBDD_n|$ lies between the max size of a QOBDD and half that quantity.

Our work discusses the exponential worst-case ROBDD size of arbitrary Boolean functions of n variables. One might ask whether certain subsets of this space of functions have better worst-case behavior in terms of worst-case ROBDD size. Abío *et al.* [ANO+12] examine Pseudo-Boolean constraints which are integer functions of the form $\sum_{i=1}^{n} a_i x_i \leq a_0$ where a_i is an integer and x_i is a Boolean variable. The solution space of such inequalities may be represented by an ROBDD. Abío identifies certain families of Pseudo-Boolean constraints for which the ROBDDs have polynomial size and others which have exponential size.

In our research we consider ROBDD sizes for a fixed variable order. Lozhkin *et al.* [LS10] extends the work of Shannon [Sha49] in examining sizes when allowed to seek a *better* variable ordering.

In Section 3.5 we introduced the residual compression ratio. Knuth [Knu09, Section 7.1.4] discusses similar ratios of sizes of BDDs vs ZDDs. Bryant [Bry18] introduces the operation of chain reduction, and discusses size ratios of BDDs and ZDDs to their chain reduced counterparts.

Castagna [Cas16] mentions the use of a lazy union strategy for representing type expressions as BDDs. Here, we have only implemented the strategy described by Andersen [And99]. The Andersen approach involves allocating a hash table to memoize all the BDDs encountered in order to both reduce the incremental allocation burden when new Boolean expressions are encountered, and also to allow occasional pointer comparisons rather than structure comparisons. Castagna suggests that the lazy approach can greatly reduce memory allocation. Additionally, from the description given by Castagna, the lazy union approach implies that some unions involved in certain BDD-related Boolean operations can be delayed until the results are needed, at which time the result can be calculated and stored in the BDD data structure.

Brace *et al.* [BRB90] demonstrate an efficient implementation of a BDD library, complete with details about how to efficiently manage garbage collection (GC). We have not not yet seen GC as an issue as our language of choice has a good built-in GC engine which we implicitly take advantage of.

The CUDD [Som] developers put a lot of effort in optimizing their algorithms. Our BDD algorithm can certainly be made more efficient, notably by using techniques from CUDD. The CUDD user manual mentions several interesting and inspiring features. More details are given in Section 7.

The sequence $a_n = 2^{2^{n-1}} - 2^{2^{n-2}}$ with $a_1 = 1$ appears in a seemingly unrelated work of Kotsireas and Karamanos[KK04] and shares remarkable similarity to Lemma 3.2. The results of Kotsireas and Karamanos are accessible on the On-Line Encyclopedia of Integer Sequences (OEIS).[1] Using the OEIS we found that the sequence ${}^{n}R_n, {}^{n}R_{n-1}, {}^{n}R_{n-2}, \ldots$ agrees with the Kotsireas sequence from a_2 up to at least a_9, which is a 78 digit integer. This similarity inspired us to investigate whether it was in fact the same sequence, and lead us to pursue the formal development we provide in Section 3.6.

6 CONCLUSION

We have provided an analysis of the explicit space requirements of ROBDDs. This analysis includes exhaustive characterization of the sizes of ROBDDs of up to 4 Boolean variables, and an experimental random-sampling approach to provide an intuition of size requirements for ROBDDs of more variables. We have additionally provided a rigorous prediction for the worst-case size of ROBDDs of n variables. We used this size to predict the residual compression the ROBDD provides. While the size itself grows unbounded as a function of n, the residual compression ratio shrinks asymptotically to zero. That is, ROBDDs become arbitrarily more efficient for a sufficiently large number of Boolean variables.

[1] The On-Line Encyclopedia of Integer Sequences or OEIS is available at `https://oeis.org`.

In order to perform our experiments, we had to design an algorithm for generating a worst-case ROBDD for a given number of variables. We have described this algorithm here as well, as having a typical worst-case ROBDD may prove to be useful for other applications than size predictions.

Our approach for this development is different from what we have found in current literature, in that while it is mathematically rigorous, its development is highly based on intuitions gained from experiment.

7 FUTURE WORK

There are several obvious shortcomings to our intuitive evaluation of statistical variations in ROBDD sizes as discussed in Section 3.2. For example, we stated that judging from the small sample in Figure 14, it would appear that for large values of n, $|ROBDD_n|$ is a good estimate for average size. We would like to continue this investigation to better justify this gross approximation.

The number of samples we take when constructing the plots in Figure 10 is constrained by the computation-time at our disposal. As shown in Figure 12 computing approximately 3000 samples of 10-variable ROBDDs takes around 50 hours. We would like to extend our program to work in a multi-threaded environment, thus exploiting more cluster nodes for shorter periods of time. It may also be possible to exploit other Common Lisp features such as dynamic extent objects, or weak hash tables to better manage the memory footprint of our computations, thus achieving more ROBDDs computed per unit time

When using ROBDDs, or presumably 0-Sup-BDDs, one must use a hash table of all the BDDs encountered so far (or at least within a particular dynamic extent). This hash table, mentioned in Section 2, is used to assure structural identity. However, it can become extremely large, even if its lifetime is short. Section 3 discusses the characterization of the worst-case size of an ROBDD as a function of the number of Boolean variables. This characterization ignores the transient size of the hash table, so one might argue that the size estimations in 3 are misleading in practice. We would like to continue our experimentation and analysis to provide ways of measuring or estimating the hash table size, and potentially ways of decreasing the burden incurred. For example, we suspect that most of the hash table entries are never re-used. We would like to experiment with weak hash tables: once all internal and external references to a particular hash table entry have been abandoned, that hash table entry can be removed, thus potentially freeing up the children nodes as well.

As discussed in Section 5, Minato [Min93] claims that using the BDD variant called 0-Sup-BDD is well suited for sparse Boolean equations. We see potential applications for this in type calculations, especially when types are viewed as sets, as in Common Lisp. In such cases, the number of types is large, but each type constraint equation scantly concerns few types. We would like to experiment with 0-Sup-BDD based implementations of our algorithms, and contrast the performance results with those found thus far.

It is known that algorithms using BDDs tend to trade space for speed. A question naturally arises: can we implement a fully functional BDD which never stores calculated values. The memory footprint of such an implementation would potentially be smaller, while incremental operations would be slower. It is not clear whether the overall performance would be better or worse. Castagna [Cas16] suggests a lazy version of the BDD data structure which may reduce the memory footprint, which would have a positive effect on the BDD based algorithms. This approach suggests dispensing with the excessive heap allocation necessary to implement Andersen's approach [And99]. Moreover, our implementation (based on the Andersen model) contains additional debug features which increase the memory footprint. We would like to investigate which of these two approaches gives better performance, or allows us to solve certain problems. It seems desirable to attain heuristics to describe situations which one or the other optimization approach is preferable.

Even though both Andersen[And99] and Minato [Min93] claim the necessity to enforce structural identity, it is not clear whether in our case, the run time cost associated with this memory burden, outweighs the advantage gained by structural identity. Furthermore, the approach used by Castagna [Cas16] seems to favor laziness over caching, lending credence to our suspicion.

CUDD [Som] uses a common base data structure, DdNode, to implement several different flavors of BDD, including Algebraic Decision Diagrams (ADDs) and ZDDs. We have already acknowledged the need to experiment with other BDD flavors to efficiently represent run-time type based decisions such as the Common Lisp run-time type reflection [NVC17, NV18] in performing simplification of type-related logic at compile-time. We wish to examine the question of whether the Common Lisp run-time type reflection can be improved by searching for better ordering of the type specifiers at compile-time. The work of Lozhkin [LS10] and Shannon [Sha49] may give insight into how much improvement is possible, and hence whether it is worth dedicating compilation time to it. CUDD, as well as the system described by Brace *et al.* [BRB90], both provide a subsystem for cache management. Although our Common Lisp implementation already provides basic cache management which can be specified by dynamic context, and is thus managed by the global GC, we have observed a need to purge nodes from the cache which are no longer referenced. In this case, the GC cannot currently purge them because they are referenced by the cache itself. We propose the use of weak hash tables to address this issue. Weak hash tables are available in several Common Lisp implementations. Measuring the effectiveness of weak hash tables is ongoing research.

ACKNOWLEDGMENTS

Many thanks to Dr. Alexandre Duret-Lutz, Dr. Maximilien Colange, and Dr. Guillaume Tochon, for the many white board discussions and brainstorming which were extremely useful in clarifying many of the tedious points in the development leading up to this article.

A GLOSSARY OF NOTATION

Symbol	Definition
$\mathscr{B}$	The index of the belt row of a worst-case ROBDD. The belt row is the highest row index, i, such that the number number of nodes in row i is 2^i.
μ	The mean value of a set of numbers. In this Section 3.3 we refer to μ_i as the weighted average of the points in a histogram curve $\mathscr{C}_n$
n	The number of Boolean variables in expression whose ROBDD is under consideration.
$m^{\underline{a}}$	Number of permutations of m things taken a at a time. $m^{\underline{a}} = \frac{m!}{(m-a)!}$.
ψ	$\psi : \mathbb{R}^+ \mapsto \mathbb{R}$ such that $2^{2^{\psi(x)+1}} - 2^{2^{\psi(x)}} = 2^{x-\psi(x)-1}$.
${}^n r_i$	Number of elements in the i'th row of a worst-case ROBDD as a function of top-down exponential growth. The actual number of nodes is $\min\{{}^n r_i, {}^n R_i\}$.
${}^n R_i$	Number of elements in the i'th row of a worst-case ROBDD as a function of top-down double exponential decay. The actual number of nodes is $\min\{{}^n r_i, {}^n R_i\}$.
row_a^b	The set of all elements in the ROBDD between rows a and b inclusive. $row_a^b = \bigcup_{j=a}^{b} row_j$.
$\lvert ROBDD_n \rvert$	Number of nodes in the worst-case ROBDD of n variables.
$\mathscr{P}(S)$	The set of all ordered pairs whose elements are chosen from set S, excluding ordered pairs such as (x, x). $\mathscr{P}(S) = \{(\alpha, \beta) \mid \alpha, \beta \in S, \alpha \neq \beta\}$.
${}^n S_i$	Number of nodes in the worst-case ROBDD in the rows strictly below row i. ${}^n S_i$ is only used when ${}^n R_i < {}^n r_i$, so ${}^n S_i = \sum_{k=i+1}^{n} {}^n R_i$.
θ	The threshold function, also denoted θ_n, defined in Theorem 3.9.
σ	The standard deviation of a set of numbers. In this Section 3.3 we refer to σ_i as the standard deviation of the sample whose histogram curve is $\mathscr{C}_n$.
$\lvert UOBDD_n \rvert$	Number of nodes in an unreduced ordered BDD.
$\lfloor x \rfloor$	The floor function. $\max\{i \in \mathbb{Z} \mid i \leq x\}$.
$\lceil x \rceil$	The ceiling function. $\min\{i \in \mathbb{Z} \mid i \geq x\}$.
$\lvert S \rvert$	The cardinality (number of elements in) set S.

REFERENCES

[Ake78] S. B. Akers. Binary decision diagrams. IEEE Trans. Comput., 27(6):509–516, June 1978.

[Als11] Gerold Alsmeyer. Chebyshev's Inequality, pages 239–240. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[And99] Henrik Reif Andersen. An introduction to binary decision diagrams. Technical report, Course Notes on the WWW, 1999.

[ANO+12] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Valentin Mayer-Eichberger. A new look at bdds for pseudo-boolean constraints. J. Artif. Intell. Res., 45:443–480, 2012.

[Ans94] Ansi. American National Standard: Programming Language – Common Lisp. ANSI X3.226:1994 (R1999), 1994.

[BC16] David Bergman and Andre A. Cire. Theoretical insights and algorithmic tools for decision diagram-based optimization. Constraints, 21(4):533, 2016.

[BRB90] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a bdd package. In 27th ACM/IEEE Design Automation Conference, pages 40–45, Jun 1990.

[Bry86] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, 35:677–691, August 1986.

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Comput. Surv., 24(3):293–318, September 1992.

[Bry18] Randal E. Bryant. Chain reduction for binary and zero-suppressed decision diagrams. In Dirk Beyer and Marieke Huisman, editors, Tools and Algorithms for the Construction and Analysis of Systems, pages 81–98, Cham, 2018. Springer International Publishing.

[Cas16] Giuseppe Castagna. Covariance and contravariance: a fresh look at an old issue. Technical report, CNRS, 2016.

[CL17] G. Castagna and V. Lanvin. Gradual typing with union and intersection types. Proc. ACM Program. Lang., (1, ICFP '17, Article 41), sep 2017.

[CMZ+97] E. M. Clarke, K. L. Mcmillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. Form. Methods Syst. Des., 10(2-3):137–148, April 1997.

[Col13] Maximilien Colange. Symmetry Reduction and Symbolic Data Structures for Model Checking of Distributed Systems. Thèse de doctorat, Laboratoire de l'Informatique de Paris VI, Université Pierre-et-Marie-Curie, France, December 2013.

[FTV16] Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. Bdd-based boolean functional synthesis. In Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II, pages 402–421, 2016.

[GPS98] Clemens Gröpl, Hans Jürgen Prömel, and Anand Srivastav. Size and structure of random ordered binary decision diagrams. In STACS 98, pages 238–248. Springer Berlin Heidelberg, 1998.

[GPS01] Clemens Gröpl, Hans Jürgen Prömel, and Anand Srivastav. On the evolution of the worst-case OBDD size. Inf. Process. Lett., 77(1):1–7, 2001.

[HM94] Mark A. Heap and M. R. Mercer. Least upper bounds on obdd sizes. 43:764–767, June 1994.

[HVP05] Haruo Hosoya, Jérôme Vouillon, and Benjamin C. Pierce. Regular expression types for XML. ACM Trans. Program. Lang. Syst., 27(1):46–90, January 2005.

[KK04] Ilias S. Kotsireas and Kostas Karamanos. Exact computation of the Bifurcation point B_4 of the logistic map and the Bailey-Broadhurst conjectures. I. J. Bifurcation and Chaos, 14(7):2417–2423, 2004.

[Knu09] Donald E. Knuth. The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams. Addison-Wesley Professional, 12th edition, 2009.

[LPR03] Michael Langberg, Amir Pnueli, and Yoav Rodeh. The ROBDD size of simple CNF formulas. In Correct Hardware Design and Verification Methods, 12th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2003, L'Aquila, Italy, October 21-24, 2003, Proceedings, pages 363–377, 2003.

[LS92] Y.-T. Lai and S. Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In Proceedings of the 29th ACM/IEEE Design Automation Conference, DAC '92, pages 608–613, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[LS10] Sergei A. Lozhkin and Alexander E. Shiganov. High accuracy asymptotic bounds on the BDD size and weight of the hardest functions. Fundamenta Informaticae, 104(3):239–253, 2010.

[MD03] D. M. Miller and G. W. Dueck. On the size of multiple-valued decision diagrams. In 33rd International Symposium on Multiple-Valued Logic, 2003. Proceedings., pages 235–240, May 2003.

[Min93] Shin-ichi Minato. Zero-suppressed BDDs for Set Manipulation in Combinatorial Problems. In Proceedings of the 30th International Design Automation Conference, DAC '93, pages 272–277, New York, NY, USA, 1993. ACM.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul., 8(1):3–30, January 1998.

[New15] William H. Newman. Steel Bank Common Lisp user manual, 2015.

[NV18] Jim Newton and Didier Verna. Strategies for typecase optimization. In European Lisp Symposium, Marbella, Spain, April 2018.

[NVC17] Jim Newton, Didier Verna, and Maximilien Colange. Programmatic manipulation of Common Lisp type specifiers. In European Lisp Symposium, Brussels, Belgium, April 2017.

[Oka98] Chris Okasaki. Purely Functional Data Structures. Cambridge University Press, New York, NY, USA, 1998.

[Sha49] C. E. Shannon. The synthesis of two-terminal switching circuits. The Bell System Technical Journal, 28(1):59–98, Jan 1949.

[SIHB97] Tsutomu Sasao, Robert J. Barton III, David S. Herscovici, and Jon T. Butler. Average and worst case number of nodes in decision diagrams of symmetric multiple-valued functions. IEEE Transactions on Computers, 46:491–494, 1997.

[Sla98] S. Slade. Object-oriented Common LISP. Prentice Hall PTR, 1998.

[Som] Fabio Somenzi. CUDD: BDD package, University of Colorado, Boulder.

[Sri02] A. Srinivasan. Algorithms for discrete function manipulation. In Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on, 2002.

[ST98] Karsten Strehl and Lothar Thiele. Symbolic model checking of process networks using interval diagram techniques. In Proceedings of the 1998 IEEE/ACM International Conference on Computer-aided Design, ICCAD '98, pages 686–692, New York, NY, USA, 1998. ACM.

[Weg87] Ingo Wegener. The Complexity of Boolean Functions. John Wiley & Sons, Inc., New York, NY, USA, 1987.