Proceedings of the 35th European Safety and Reliability & the 33rd Society for Risk Analysis Europe Conference
Edited by Eirik Bjorheim Abrahamsen, Terje Aven, Frederic Bouder, Roger Flage, Marja Ylönen
©2025 ESREL SRA-E 2025 Organizers. *Published by* Research Publishing, Singapore.
doi: 10.3850/978-981-94-3281-3_ESREL-SRA-E2025-P2278-cd

Constructing binary decision diagrams using machine learning

Arne Bang Huseby
Department of Mathematics, University of Oslo, Norway. E-mail: arne@math.uio.no

Binary decision diagrams are a highly popular method for calculating system reliability. By representing the structure function of a binary monotonic system as a binary decision diagram, the calculation of the system's reliability can, in principle, be performed efficiently. However, constructing such diagrams can still be challenging. To ensure that calculations are done quickly, it is important that the diagrams are as compact as possible. In this article, we will show how binary decision diagrams can be constructed using machine learning. The method assumes the existence of a dataset with corresponding values of component states and system states. Such a dataset can easily be generated of any size if the structure function is known and can be calculated efficiently. However, the method can also be used to approximate an unknown structure function based on similar experimental data. The number of possible component states naturally grows exponentially with the number of components in the system. Consequently, if the number of components in the system is high, it will, in practice, not be possible to obtain sufficient data to perfectly describe the system's structure. In the article, we will compare different strategies for handling this problem. Specifically, it is of interest to compare methods that aim to approximate the structure function as accurately as possible with methods that instead focus on estimating system reliability as accurately as possible. The methods will be illustrated with a few examples.

Keywords: Binary decision diagrams, system reliability, structure functions, machine learning.

1. Introduction

When we use the term *system*, we think of some technological unit consisting of a finite set $C = \{1, \dots, n\}$ of *components* which are operating together. A *binary* system has only two possible states: *functioning* or *failed*. Moreover, each component is either functioning or failed as well. The state of the system is represented as a function ϕ of the component state variables. The function ϕ is referred to as the *structure function* of the system, and the ordered pair (C, ϕ) is referred to as a *binary monotone system*. A main goal is to compute the reliability of the system, i.e., the probability that the system is functioning given the joint distribution of the component state variables. In general, this is an NP-hard problem. Still in many cases it is possible to find algorithms which can compute the reliability fairly efficiently for large classes of systems.

In this paper we will focus on methods based on binary decision diagrams. By representing the structure function of a binary monotonic system as a binary decision diagram, the calculation of the system's reliability can, in principle, be performed efficiently. However, constructing such diagrams can still be challenging. To construct a binary decision diagram of a binary monotone system, one would typically start with some form of precise representation of the system's structure function. This could, for example, take the form of a network where the system operates if a set of terminals can communicate via the edges in the network. In other cases, the system's structure function is represented by a compact mathematical formula. However, there may also be situations where such a precise representation is not available. Instead, one might have a dataset consisting of corresponding observations of component states and system states. Based on these observations, the goal is to estimate a structure function.

In this article, we will show how binary decision diagrams can be constructed using a simple machine learning method. The method assumes the existence of a dataset with corresponding values of component states and system states. Such a dataset can easily be generated of any size if the structure function is known and can be calculated efficiently. However, the method can also be used to approximate an unknown structure function based on similar experimental data.

2. Binary decision diagrams

Binary decision diagrams or BDDs have many different applications in computer science, and dates back to the papers Lee (1959) and Akers (1978). More efficient implementations were introduced in Bryant (1986), Brace et al. (1990) and Burch et al. (1992). Applications of BDDs in reliability theory were introduced in Coudert and Madre (1993) and Rauzy (1993) and adapted to network reliability in Bobbio and Terruggia (2007). For a survey of these and other related methods see Rauzy (2008).

A binary decision diagram can be interpreted as a representation of a binary function $\phi : \{0,1\}^n \to \{0,1\}$ in the form of a rooted directed acyclic graph. In our context the function ϕ is typically the structure function of some binary monotone system. This means that we only consider the special case where the binary function is non-decreasing in each argument. The *root* and the *intermediate* nodes are drawn as circular nodes, and labelled with indices of the binary input variables. The edges represent *decisions* regarding the values of the input variables, i.e., whether the value of an input variable is fixed to be either 0 or 1. The square *leaf* nodes represents cases where the associated binary function is trivial, and the labels in this case represent the corresponding binary value, i.e., either 0 or 1.

Figure 1 shows a binary decision diagram of a 2-out-of-3 system. There are three binary input variables, x_1, x_2, x_3. The binary function ϕ, represents the structure function of the system, and is given by:

$$\phi(\boldsymbol{x}) = x_1x_2 + x_1x_3 + x_2x_3 - 2x_1x_2x_3$$

To each node in the diagram we associate a binary function defined as a function of the remaining binary variables whose values are not yet fixed. In particular, the root node in the diagram, labelled 1, is associated with the function ϕ itself since at this stage none of the variables are fixed. The lower

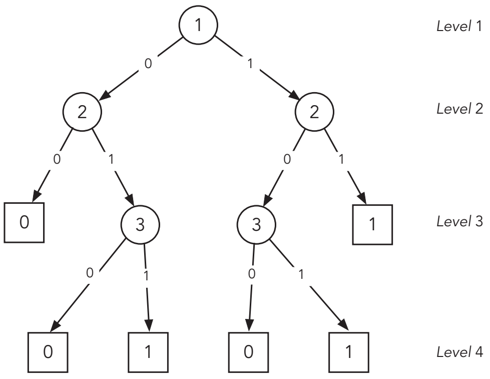

Fig. 1. An ordered binary decision diagram of a 2-out-of-3 system.

level nodes are associated with structure functions of minors of the system under consideration. Since each input variable has two possible values, 0 and 1, each node has exactly two *children*, i.e., nodes connected by edges to the *parent* node, where as a convention the lefthand child represents the binary function given that the respective input variable is fixed to be 0, while the righthand child represents the binary function given that the respective input variable is 1. In Figure 1 there are two nodes labelled 2 which are children of the root node 1. To the leftmost node we associate the function:

$$\phi(0_1, \boldsymbol{x}) = x_2x_3,$$

while the rightmost node we associate the function:

$$\phi(1_1, \boldsymbol{x}) = x_2 + x_3 - x_2x_3$$

At the next level of the diagram there are two intermediate nodes and two leaf nodes. From left to right these nodes correspond to the following binary functions:

$$\begin{aligned}\phi(0_1, 0_2, \boldsymbol{x}) &= 0,\\ \phi(0_1, 1_2, \boldsymbol{x}) &= x_3,\\ \phi(1_1, 0_2, \boldsymbol{x}) &= x_3,\\ \phi(1_1, 1_2, \boldsymbol{x}) &= 1.\end{aligned}$$

We observe the binary functions associated with the two leaf nodes are trivial with values 0 and 1 respectively. The binary functions associated with the two intermediate nodes depends on the remaining input variable 3. Thus, for these nodes we proceed to the final level consisting of four leaf nodes. From left to right these nodes correspond to the following binary functions:

$$\phi(0_1, 1_2, 0_3) = 0,$$
$$\phi(0_1, 1_2, 1_3) = 1,$$
$$\phi(1_1, 0_2, 0_3) = 0,$$
$$\phi(1_1, 0_2, 1_3) = 1$$

Note that since the binary functions associated with the two intermediate nodes labelled 3 are identical, it follows that the subgraphs rooted at these nodes are *isomorphic*[a].

We now consider the states of the input variables as independent stochastic variables, denoted X_1, X_2, X_3, and let $P(X_i = 1) = p_i$, $i = 1, 2, 3$. Then each node in the diagram corresponds to an *event* defined by the state of the stochastic variables that are fixed. The event corresponding to the root node, has probability 1 since none of the variables are fixed at this stage. By propagating the event probabilities downwards through the diagram all the way to the leaf nodes, we can easily compute the probabilities of each event. Finally, given the probabilities of the leaf nodes, we find the probability distribution of $\phi(\boldsymbol{X})$ by adding the probabilities associated with the leaf nodes labelled 0 and 1 respectively. In the diagram shown in Figure 1 there is a one-to-one correspondence between the input variables and the levels of the diagram. That is, input variable 1 is handled at level 1, input variable 2 is handled at level 2, etc. Such diagrams are referred to as *ordered binary decision diagrams*. When the diagram has a rooted tree structure, like the one shown in Figure 1, however, we do not need to handle the input variables in an ordered way. In Figure 2 the input variables 2 and 3 are handled in a different order in the lefthand and righthand parts of the diagram. Thus, we observe that the two nodes at level 2 do *not* refer to the same input variable.

[a]Two directed graphs G and H with node sets $V(G)$ and $V(H)$ respectively are said to be *isomorphic* if there exists a bijective mapping, $\psi : V(G) \to V(H)$, such that G contains a directed edge from node u to node v if and only H contains a directed edge from node $\psi(u)$ to $\psi(v)$.

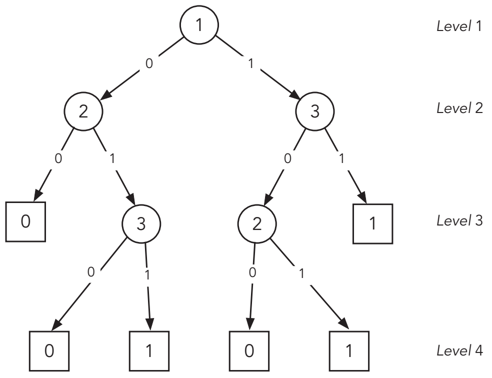

Fig. 2. An unordered binary decision diagram of a 2-out-of-3 system.

The event probabilities of the nodes in the diagram is computed event by event, and level by level in the diagram. Since each event probability is calculated in constant time, the computational complexity of computing all event probabilities, and thus also the distribution of $\phi(\boldsymbol{X})$, is proportional to the number of nodes in the diagram.

Note that when a binary decision diagram has been constructed, this diagram represents the binary function of interest, regardless of the distributions of the input variables. Thus, we can use the same diagram to compute the distribution of $\phi(\boldsymbol{X})$ for any number of distributions of the input variables. This allows us to e.g., use the diagram for various types of sensitivity analysis.

A *generic* Python script for constructing BDDs is given in Huseby and Dahl (2024). This Python script can be used on many different types of systems. However, in order to apply this script to a given system, some of the generic methods must be tailored to fit the system under consideration. In particular, the following methods must be implemented in a system specific way:

- A method for representing the binary functions obtained as a result of fixing the values of the input variables (*restriction* and *contraction*)
- A method for determining if a binary function is trivial.
- A method for selecting input variables for each node in the BDD

The different implementations of these methods depend strongly on the class of systems under consideration, and on the chosen representation of the binary functions. Specialized Python scripts for various classes of systems can also be found in Huseby and Dahl (2024).

In Bryant (1986) a *reduced* form of a binary decision diagram was introduced. See also Brace et al. (1990). We refer to such diagrams as *reduced ordered binary decision diagrams* or ROBDDs. An ROBDD is obtained from an ordered binary decision diagram by merging nodes which correspond to equal binary functions. In particular, such diagrams only have two leaf nodes, one labelled 0 and one labelled 1. For these reductions to work, however, only ordered decision diagram are allowed. The reason for this is that if we use unordered diagrams, nodes at the same level will correspond to binary functions with different input variables. Thus, it is typically not possible to find nodes which correspond to equal binary functions. Figure 3 shows a reduced ordered binary decision

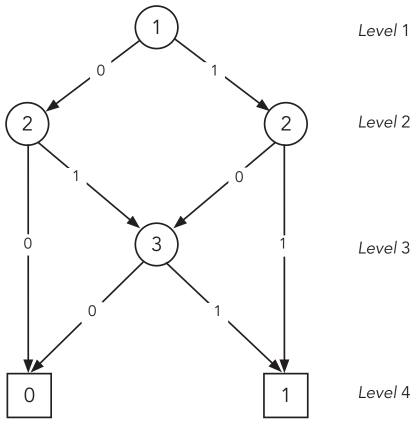

Fig. 3. A reduced ordered binary decision diagram of a 2-out-of-3 system.

diagram for a 2-out-of-3 system. This is obtained from the diagram shown in Figure 1 by merging the leaf nodes labelled 0 into one leaf node and by merging the leaf nodes labelled 1 into one leaf node. Finally, the two intermediate nodes labelled 3 are merged into one intermediate node.

The reduction technique introduced in Bryant (1986) can in many cases reduce the computational complexity of reliability calculations considerably. As noted by Bryant (1986), however, the ordering of the input variables, or components in our context, may have a large impact on the size of the diagram. Unfortunately, finding an ordering that minimizes the size of the graph is itself known to be a *co-NP-Complete* problem (see Bryant (1986)). Thus, most algorithms for constructing reduced ordered binary decision diagrams rely on some sort of heuristic method for ordering the input variables. Here, we will not go further into the problem of finding efficient orderings, but instead assume that an ordering of the input variables has been chosen. Given an ordering of the input variables, the problem of constructing a reduced ordered binary decision diagram is still not trivial. There are two main challenges with this method. Firstly, the size of the ordered binary decision diagram may be very large, as the number of nodes typically grows exponentially in the number of input variables. The second challenge is that identifying isomorphic subgraphs can be computationally difficult as well. In order to manage these problems the reductions should be done along the way as a part of the initial construction.

A *generic* Python script for constructing ROBDDs is also given in Huseby and Dahl (2024). As for the BDD script, the following methods must be implemented in a system specific way:

- A method for representing the binary functions obtained as a result of fixing the values of the input variables (*restriction* and *contraction*).
- A method for determining if a binary function is trivial.
- A method for determining if two binary functions are identical.

Again the actual implementation of these methods depend strongly on the class of systems under consideration, and on the chosen representation of the binary functions.

3. Constructing BDDs based on training data

Let (C, ϕ) be a binary monotone system where $C = \{1, \ldots, n\}$, and let $\boldsymbol{x} = (x_1, \ldots, x_n)$ denote the component state vector. We assume that for each vector $\boldsymbol{x}$, the corresponding system state $\phi(\boldsymbol{x})$ can be *calculated efficiently*.

Training data can then be generated by sampling $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ from a suitable joint distribution, and for each sampled vector, $\boldsymbol{x}_j$, we calculate the corresponding system state $\phi(\boldsymbol{x}_j)$, $j = 1, \ldots, N$. The resulting data can then be represented by a matrix M given by:

$$M = \begin{bmatrix} \phi(\boldsymbol{x}_1) & x_{11} & x_{21} & \cdots & x_{n1} \\ \phi(\boldsymbol{x}_2) & x_{11} & x_{21} & \cdots & x_{n1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi(\boldsymbol{x}_N) & x_{1N} & x_{2N} & \cdots & x_{nN} \end{bmatrix}$$

Assume that we want to find the training data matrices corresponding to the *restriction* and *contraction* with respect to some component $i \in C$. We denote these matrices by M_{-i} and M_{+i} respectively. (*A similar procedure will be applied at lower levels in the BDD.*)

In order to determine M_{-i} and M_{+i}, we introduce the following index sets:

$$L_i = \{j : x_{ij} = 0\}$$
$$U_i = \{j : x_{ij} = 1\}$$

M_{-i} is then obtained from M by deleting the column vector $(x_{i1}, \ldots, x_{iN})'$, and then keep only the rows:

$$[\phi(\boldsymbol{x}_j), x_{1j}, \ldots, x_{i-1,j}, x_{i+1,j}, \ldots, x_{nj}],$$

where $j \in L_i$. Similarly, M_{+i} is obtained from M by deleting the column vector $(x_{i1}, \ldots, x_{iN})'$, and then keep only the rows:

$$[\phi(\boldsymbol{x}_j), x_{1j}, \ldots, x_{i-1,j}, x_{i+1,j}, \ldots, x_{nj}],$$

where $j \in U_i$. In order to choose the *pivotal element*, i.e., the component we want to use in the minor operations, we use different methods for BDDs and ROBDDs. For BDDs we choose the remaining component having the state variable with the *highest Birnbaum importance measure* with the structure function, where the Birnbaum importance measure is calculated empirically based on the training data. For ROBDDs we assume that we have sorted the components initially, and then simply choose the component with the lowest index.

In order to determine if a system is trivial, we apply the following rule:

- If $\phi(\boldsymbol{x}_j) = 0$ for $j = 1, \ldots, N$, the system is classified as *trivially failed*.
- If $\phi(\boldsymbol{x}_j) = 1$ for $j = 1, \ldots, N$, the system is classified as *trivially functioning*.

Due to the randomness of the training data, it is not possible to determine with certainty whether or not two systems are identical. An *indication of equivalence*, however, can be obtained by comparing the fraction of remaining rows where the structure function is equal to 1. Thus, two systems with *approximately equal estimated reliability*, could possibly be considered to be equivalent.

4. Numerical examples

In this section we illustrate the proposed method by considering two very simple examples. In the first example we consider a *threshold system*, i.e., a system which is functioning if and only if a weighted sum of the component state variables is greater than or equal to a given threshold value. The system is denoted by (C, ϕ), where $C = \{1, \ldots, 6\}$, and where the structure function is given by:

$$\phi(\boldsymbol{x}) = \mathrm{I}(\sum_{i=1}^{6} a_i x_i \geq b), \qquad (1)$$

where the weight vector, $\boldsymbol{a} = (a_1, \ldots, a_6)$, is given by $(8, 7, 6, 5, 3, 2)$ and the threshold, b, is 20.

As mentioned, for the ROBDD method we need to sort the components. For threshold systems it is fairly obvious that the components should be ordered with respect to their weights, so component 1 is the one with the highest weight, component 2 is the one with the second highest weight etc. In cases where two or more components have equal weights, we simply break ties arbitrarily.

In order to represent the binary functions obtained by fixing the value of say, the jth input variable, we apply Eq. (1) and get:

$$\phi(0_j, \boldsymbol{x}) = I(\sum_{i \neq j} a_i x_i \geq b),$$
$$\phi(1_j, \boldsymbol{x}) = I(\sum_{i \neq j} a_i x_i \geq b - a_j)$$

From this it follows that the restriction and the contraction of (C, ϕ) with respect to component j are both threshold systems. Thus, the class of threshold systems is closed under minor operations.

If the sum of the remaining weights are less than the threshold, we know that the minor is trivial with structure function identical to 0. Conversely, if the threshold value is less than or equal to zero, then the minor is trivial with structure function identical to 1 . Thus, in order to check whether a binary function is trivial, we only need to compute the sum of the weights and compare it to the threshold. Based on these observations it

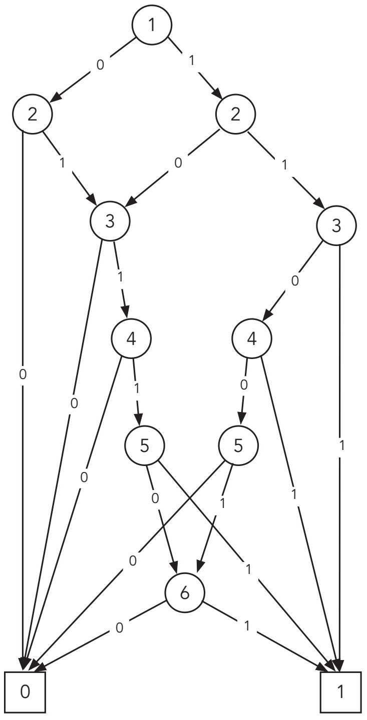

Fig. 4. A reduced unordered binary decision diagram of the threshold system (C, ϕ).

is fairly easy to construct a BDD or an ROBDD for this system. In Figure 4 we have shown the ROBDD constructed directly from the mathematical definition of the system. We note that at level 3 in this diagram two nodes have been merged. To see why this can be done, we note that:

$$\phi(0_1, 1_2, \boldsymbol{x}) = \mathrm{I}(\sum_{i=3}^{6} a_i x_i \geq b - a_2)$$
$$= \mathrm{I}(\sum_{i=3}^{6} a_i x_i \geq 13),$$

and:

$$\phi(1_1, 0_2, \boldsymbol{x}) = \mathrm{I}(\sum_{i=3}^{6} a_i x_i \geq b - a_1)$$
$$= \mathrm{I}(\sum_{i=3}^{6} a_i x_i \geq 12).$$

The fact that the threshold values for $\phi(0_1, 1_2, \boldsymbol{x})$ and $\phi(1_1, 0_2, \boldsymbol{x})$ are different, may suggest that these two structure functions are *not equal*. However, by considering the set possible values of the weighted sum $\sum_{i=3}^{6} a_i x_i$, it turns out that there are no values between 11 and 13. Thus, we may replace the threshold value of $\phi(1_1, 0_2, \boldsymbol{x})$ by 13 without changing the system. Hence, we conclude that:

$$\phi(0_1, 1_2, \boldsymbol{x}) \equiv \phi(1_1, 0_2, \boldsymbol{x})$$

By a similar argument we can justified that two nodes have been merged at level 6 as well.

In order to demonstrate the training method proposed in the previous section, we generated a sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, where $N = 10000$, and calculated the corresponding values of the structure function ϕ. In order to span the outcome space as much as possible, the component state variables were generated as independent, identically distributed variables such that $P(X_i = 1) = \frac{1}{2}, i = 1, \ldots, 6$. Based on these simulated observations we constructed both a BDD representation and an ROBDD representation of the system. Focussing on the ROBDD case the resulting diagram was almost identical to the one illustrated in Figure 4. The only difference was that the training method did not identify the two nodes at level 6 as identical, and hence did not merge these nodes. Still the resulting diagram matched the structure perfectly.

In the second example we consider a two-terminal undirected network system illustrated in Figure 5.

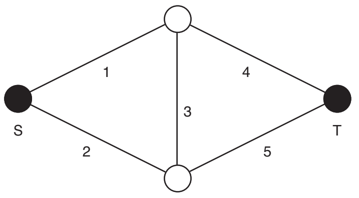

Fig. 5. A two-terminal undirected network system.

For the ROBDD method we again need to sort the components. However, for threshold systems finding the optimal ordering of the components is not so easy. In the test we simply ordered the components according to the indices indicated in Figure 5. This resulted in an ROBDD which was a bit complex. As in the previous example we we generated a sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, where $N = 10000$, and calculated the corresponding values of the structure function ϕ. Again, the component state variables were generated as independent, identically distributed variables such that $P(X_i = 1) = \frac{1}{2}$, $i = 1, \ldots, 5$. Based on these simulated observations we constructed both a BDD representation and an ROBDD representation of the system. The ROBDD representation essentially replicated the ROBDD we got based on the mathematical representation except that the training method failed to identify some identical nodes. Still the resulting diagram matched the structure perfectly. However, the BDD representation, shown in Figure 6, was actually slightly simpler than the ROBDD representation. The main reason for this was that the BDD training method managed to find better choices for the pivotal elements. In fact the BDD training method clearly benefitted from not being restricted to ordered diagrams.

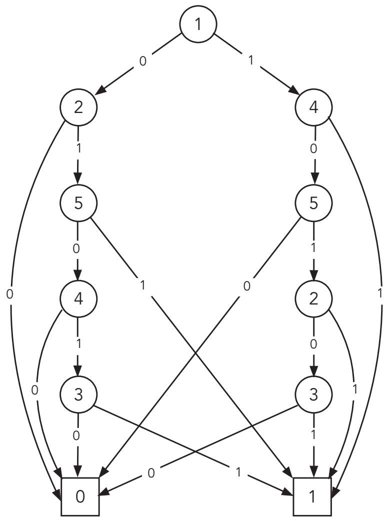

Fig. 6. A BDD of a two-terminal undirected network system.

5. Conclusions and final remarks

In this article, we have demonstrated how machine learning can be used to construct BDD and ROBDD representations of binary monotone systems. The methodology has been illustrated using two very simple systems: a threshold system and a network system. For these systems, the proposed methods were able to construct diagrams that perfectly matched the binary systems. Larger and more complex systems require an increase in the amount of training data to achieve satisfactory precision.

For effective ROBDD representations, it is crucial that the method can identify identical structures during the process. In both examples, we observed that this only partially succeeded. To make the methods more accurate, it would be desirable to have the ability to test whether two given structures are identical. This can be achieved by generating test data during the process that can be used for such comparisons. However, a necessary prerequisite for this is having access to the true structure function in addition to the training data.

In the examples presented here, all training data were generated from a uniform distribution over the space of binary vectors. This was chosen to ensure the data would best span the outcome space. For large, complex systems, the proposed methods can only provide an approximation of the true structure. In such situations, it becomes important that the approximation performs as well as possible for states with high probability. To ensure this, training data should be generated based on a realistic probability distribution rather than a uniform distribution.

Acknowledgement

The author is grateful to the Department of Mathematics at the University of Oslo for funding this project.

References

Akers, S. B. (1978). Binary decision diagrams. *IEEE Transactions on Computers C-27*, 509–516.

Bobbio, A. and R. Terruggia (2007). Binary decision diagrams in network reliability analysis. In *1st IFAC Workshop on Dependable Control of Discrete Systems*, Volume 40, pp. 199–204. IFAC Proceedings Volumes.

Brace, K. S., R. L. Rudell, and R. G. Bryant (1990). Efficient implementation of a bdd package. In *27th ACM/IEEE Design Automation Conference*, Volume 0738, pp. 40–45.

Bryant, R. G. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers C-35*, 677–691.

Burch, J. R., E. M. Clarke, K. L. McMillan, D. L. Dill, and L. Hwang (1992). Symbolic model checking: 10^{20} states and beyond. *Information and Computation 98*, 142–170.

Coudert, O. and J. C. Madre (1993). Fault tree analysis: 10^{20} prime implicants and beyond. In *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'93, Atlanta NC, USA*, pp. Pagestart–Pageend. Publisher Name.

Huseby, A. B. and K. R. Dahl (2024). *Risk- and reliability analysis with applications*. University of Oslo, Norway.

Lee, C. Y. (1959). Representation of switching circuits by binary-decision programs. *Bell System Technical Journal 38*, 985–999.

Rauzy, A. B. (1993). New algorithms for fault tree analysis. *Reliability Engineering and System Safety 40*, 203–211.

Rauzy, A. B. (2008). *Binary Decision Diagrams for Reliability Studies*. Springer Verlag, London.