



This item was submitted to Loughborough's Institutional Repository (<https://dspace.lboro.ac.uk/>) by the author and is made available under the following Creative Commons Licence conditions.

 **creative commons**
C O M M O N S D E E D

Attribution-NonCommercial-NoDerivs 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

 **Attribution.** You must attribute the work in the manner specified by the author or licensor.

 **Noncommercial.** You may not use this work for commercial purposes.

 **No Derivative Works.** You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the [Legal Code \(the full license\)](#).

[Disclaimer](#) 

For the full text of this licence, please go to:
<http://creativecommons.org/licenses/by-nc-nd/2.5/>

High Performance Reliability Analysis of Phased Mission Systems

S. Reed, S. J. Dunnett and J. D. Andrews

Department of Aeronautical and Automotive Engineering, Loughborough University, Loughborough, Leics, UK

Abstract

Systems often operate over a set of time periods, known as phases, in which their reliability structure varies and many include both repairable and non-repairable components. Success for such systems is defined as the completion of all phases, known as a phased mission, without failure. An example of such a system is an aircraft landing gear system during a flight. The Binary Decision Diagram (BDD) method provides the most efficient solution to the unreliability of non-repairable systems whilst for repairable systems Markov or other state-space based methods have been most widely applied. For systems containing both repairable and non-repairable components the repairable modelling methods are normally used, despite having far higher computational expense than the non-repairable methods, since only they are able to handle the dependencies involved. This paper introduces improvements to the BDD method for analysing non-repairable systems as well as an entirely new method that utilises a new modelling technique involving both BDD and Markov techniques.

Introduction

Many real world systems operate in phased missions where the reliability structure varies over consecutive time periods, known as phases, which must be completed without failure. Typical examples include aircraft flights and nuclear power station safety systems. Calculating the reliability of a mission is computationally expensive particularly if the system has components that are repairable and/or have multiple failure modes. Increased solution efficiency is an important goal as it increases the size of problem that can be analysed and increases the possibilities for performing importance measure and real time analysis.

Reliability engineering research has developed methods that allow the mission reliability to be found from the set of fault trees describing the system reliability in terms of component level failures for each phase, along with the component level failure probabilities (or failure probability time distributions). Two key areas in which progress has focused is in widening the scope of the techniques so that they are applicable to a larger range of system types and improving the efficiency of analysis to increase the size of system that can be analysed and reduce computational effort. The earliest phased mission analysis methods involved the direct manipulation of the fault trees. In the first known work, Esary and Ziehms [1] introduced a fault tree based method to transform a phased mission into an equivalent single phased mission. Each component basic event in the phase fault trees is replaced by an OR gate with the performance of the component up to and including that phase as inputs. The transformed phase fault trees are then combined into a single fault tree

and standard fault tree methods used to derive the system's reliability. This method is unsuitable for solving larger systems since the basic event transformation leads to significant complexity.

To reduce the computational burden several researchers such as Dazhi and Xue [2], Kohda, Wada and Inoue [3], Somani and Trivedi [4] and La Band and Andrews [5] developed phased algebra rules that deal with the dependencies between events belonging to the same component in different phases. Not only did the use of the phased algebra reduce the computational burden but it also allowed individual phase reliabilities to be obtained. Even with the use of phased algebra rules, the fault tree based methods remain unsuitable for analysing large systems within reasonable time frames, particularly those with non-coherent fault trees, and this led to the adoption of the more efficient and powerful Binary Decision Diagram (BDD) technique. Bryant [6] demonstrated the power of the technique in manipulating Boolean functions. To enable the application of the technique to reliability analysis Rauzy [7] introduced algorithms to analyse fault trees through the BDD method. Zang, Wang, Sun and Trivedi [8] then extended the BDD algorithms to the analysis of systems containing components with multiple competing failure modes. A component has multiple competing failure modes if it can fail in distinct and mutually exclusive ways, e.g. a valve failing stuck closed or stuck open.

Zang, Sun and Trivedi [9] were the first to use the BDD method to analyse the reliability of phased mission systems. They made changes to the BDD build and evaluation procedures to encode the phase algebra and deal with the phase dependencies. This work marked a significant step forward by enabling large phased mission systems to be analysed that were beyond the scope of the earlier fault tree methods. Tang and Dugan [10] made the obvious improvement of combining this with the multiple competing failure mode BDD extension [8], resulting in a BDD method that can be used to analyse phased mission systems containing multiple competing failure modes.

For systems containing repairable components the combinatorial techniques cannot be used to quantify reliability without the use of another method, due to the additional dependencies involved. State-space models such as Markov [11, 12, 13] and Petri Nets [14] or simulation techniques such as Monte Carlo are normally used. The downside to these methods is that they suffer from increased computational requirements and decreased accuracy compared to the exact BDD method for non-repairable systems – particularly when modelling large systems or those with many phases.

To reduce the computational burden, Wang and Trivedi [15] developed an approach that utilises the BDD method for analysing systems that operate in phased missions and contain repairable components. The method relies on an assumption that repaired components are only integrated back into the system at the start of the next phase. This allows the BDD to be built assuming independence between events in different phases, using the algorithm from [7], before resolving the phase dependencies between the basic events on each path through the BDD with a Markov chain. Although the method modelled multiple failure modes at the component level it did not

do so at the structural level. Unfortunately, due to the assumption of event independence during BDD construction, the method does not benefit from increased performance if the system analysed contains some non-repairable components.

Currently obtaining the mission reliability of a phased mission system that contains both non-repairable and repairable components requires the use of one of the repairable methods discussed above. Since these methods have far greater computational complexity than the non-repairable methods, the ideal method would obtain the best of both worlds – the ability to analyse the non-repairable parts of a system with great efficiency whilst retaining the power to correctly resolve the dependencies involved in the repairable parts. This paper helps move closer to this goal by introducing new techniques to boost solution efficiency of the BDD based non-repairable analysis techniques. A new method is also introduced that integrates BDD and Markov techniques to enable the reliability of systems containing both non-repairable and repairable multiple failure mode components to be obtained with greater efficiency than any existing methods.

This paper begins with a definition of a phased mission and an example system, an explanation of the BDD method and the introduction of several new improvements to the analysis of non-repairable phased missions. The Markov model used in the analysis of repairable systems is then explained, followed by the description of an entirely new method for efficient analysis of systems with non-repairable and repairable components. The paper concludes with a demonstrative application of a code that has been developed based on these new methods to the example system.

Phased Mission Definition and Example Mission

The methods and models discussed in this paper make the following assumptions for a phased mission:

- The mission consists of a set of consecutive phases.
- The time duration for each phase is known and fixed.
- For mission success all phases must be completed.
- All components work at the start of the mission.

A phased mission can be represented through a fault tree as an OR top gate with inputs being individual fault trees for each phase, since mission failure occurs if the system fails in any phase. The fault tree shown in Figure 1, representing a system operating in a three phased mission, will be used to demonstrate the methods expressed in this paper.

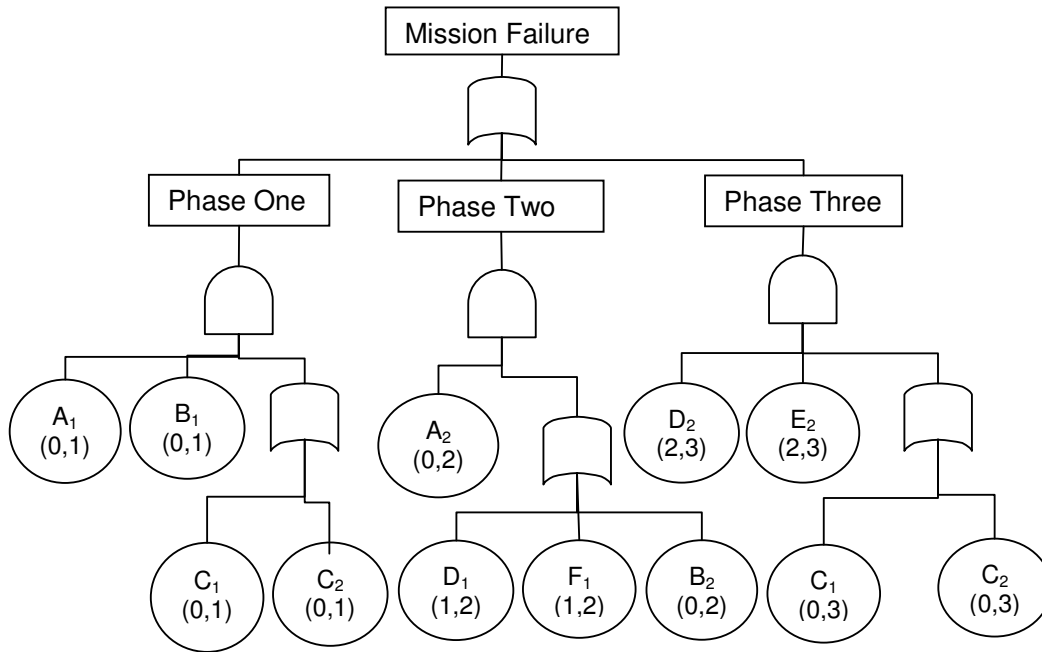


Figure 1 - Mission Fault Tree for example mission

The example system has 6 components: *A*, *B* and *C* which are non-repairable and *D*, *E* and *F* which are repairable. Each component has 2 mutually exclusive failure modes, named 1 and 2. Basic events are labelled in the form $I_j(p_1, p_2)$ where *I* is the component name, *j* is the failure mode name and p_1 and p_2 indicate the end of the phase which failure occurs after and before, respectively. The start of the mission is defined as the end of phase 0. Components *A*, *B* and *C* have failure modes with a constant failure rate of 0.001 failures per hour, whilst the repairable components have failure rates of 0.001 failure per hour when in the working state and repair rates of 0.01 repairs per hour when failed. Each phase in the example mission lasts 100 hours.

Boolean Variables

Boolean variables are used to represent a basic event associated with a particular component in a particular failure mode over some subset of phases of a mission, through their mapping to a value of 0 or 1. These event definitions are defined below:

$I_j(p_1, p_2) = 1$ is defined as the failure event of component *I* in failure mode *j* between the end of phase p_1 and end of phase p_2 .

$I_j(p_1, p_2) = 0$ is defined as the success event of component *I* in failure mode *j* between the end of phase p_1 and end of phase p_2 ; the component is either operational or in any other failure mode of component *I*.

The BDD method

The BDD forms the basis of the modelling methods used in this paper and a brief description of the method and terminology used are explained here. A BDD is a compact data structure, in the form of a rooted, directed, acyclic graph, and is used here to model the Boolean function representing a system's reliability over a phased mission. They were introduced by Bryant [6] and are based upon Shannon decomposition theory [16].

A BDD consists of decision nodes and two types of terminal node named terminal 0 and terminal 1. Each decision node is labelled with a Boolean variable and has two edges, a 0 edge and 1 edge, each connecting to a different child node. The 0 edge represents an assignment of 0 to the node's Boolean variable, whilst the 1 edge represents an assignment of 1. The Boolean variables are ordered such that if an edge from a node labelled with variable x connects to a node labelled with variable y then $\text{index}(x) < \text{index}(y)$. A BDD starts at a single node known as its top or root node. The compact nature results from two important reduction features:

- The merging of isomorphic subgraphs, thus removing repetition.
- The elimination of any node whose children are isomorphic by replacing it with its child.

The value of the Boolean function represented by the BDD, corresponding to the mapping of Boolean values to variables on any route from the root node to a terminal node is given by the terminal nodes value. A value of 1 represents mission value whilst a value of 0 represents mission success. These routes are known as the paths of the BDD and represent mutually exclusive Boolean value to variable assignments. Often not all variables in a BDD will appear on a path and these are variables for which their value assignment has no bearing on the value of the path. Such variables are sometimes known as 'doesn't matter variables'. The probability of the Boolean function modelled by the BDD evaluating to 1 or 0 is the sum of all paths through it that end at a terminal one or terminal zero node respectively. Since the sum of the probabilities of the Boolean function evaluating to either 1 or 0 is always 1, it is only necessary to find the first and the other follows immediately. For this reason only the probability of the function evaluating to 1 is normally found directly and from now on the term "path" will refer to a path that ends in a terminal one node.

An example of a BDD is shown in Figure 2. In this example the BDD nodes are labelled with Boolean variables A , B and C and the routes through the BDD to the terminal 1 node encode Boolean value mappings $\{A=1, B=x, C=1\}$, $\{A=1, C=0, B=1\}$ and $\{A=0, B=1, C=x\}$ where an assignment of x indicates that assigning 1 or 0 to that Boolean variable on the path has no effect on the value of the Boolean function the BDD represents. These paths therefore form a set of mutually exclusive Boolean value variable assignments resulting in a 1 value for the Boolean function represented by the BDD.

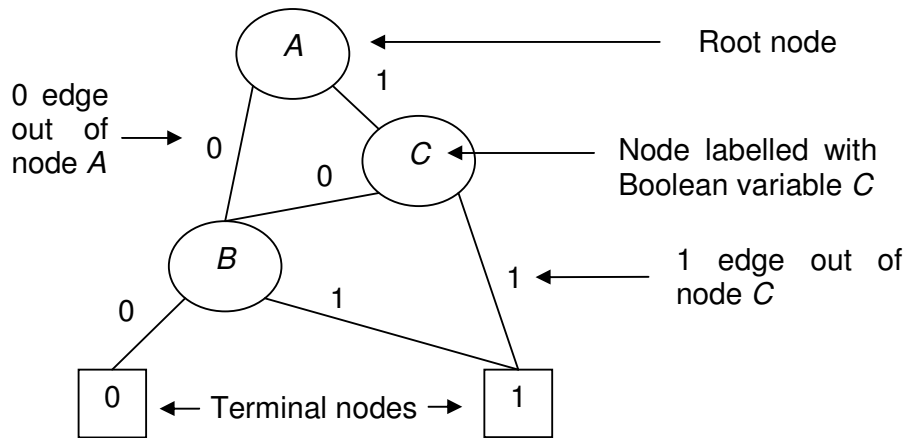


Figure 2 - Example of a BDD

The BDD structure can be presented in terms of a series of nested *if-then-else* structures, each representing a decision node in the BDD. The *if-then-else* structure represents the decomposition of a Boolean function f , of random variables (x_1, x_2, \dots, x_n) , around x_i as given by Equation 1.

$$f = x_i \bullet f_{x_i=1} + (1 - x_i) \bullet f_{x_i=0} \quad (1)$$

Where, $f_{x_i=1} = f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$ and $f_{x_i=0} = f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$.

Using the *if-then-else (ite)* format this becomes Equation 2.

$$f = ite(x_i, f_{x_i=1}, f_{x_i=0}) \quad (2)$$

Equation 2 describes the following situation, if variable x_i occurs (fails) then $f_{x_i=1}$ is considered, else $f_{x_i=0}$ is considered. To form the BDD from a fault tree, the first step is to convert each of the basic events in the fault tree into their *if-then-else* structure representations.

The BDD for a component, A, is represented as shown in Figure 3.

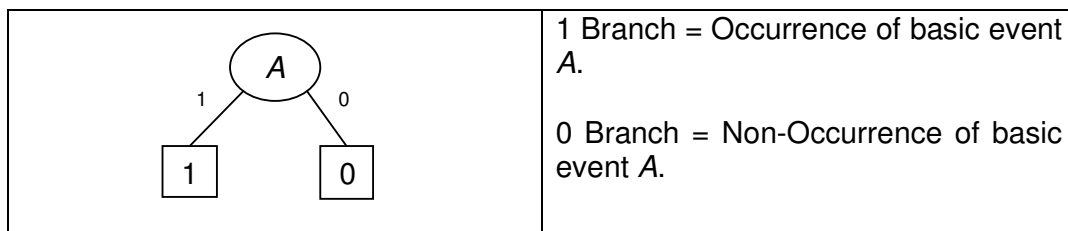


Figure 3- Binary decision diagram vertex for component A failure

Using the *if-then-else* notation the BDD in Figure 3 can be represented in the form shown by Equation 3. In this notation the first variable inside the brackets, for example A in Equation 3, represents a Boolean variable, whilst the second and third variables represent the edges connected to assignments

of 1 and 0 to the Boolean variable respectively. The second and third variables can be 1 or 0 representing the equivalent terminal node, or alternatively further if-then-else structures representing decision nodes.

$$A = ite(A,1,0) \tag{3}$$

The *if-then-else* structure representation of a gate in the fault tree is formed by performing the appropriate logical operation on the *if-then-else* structures representing the gate inputs. If the gate has more than two inputs, then the Boolean logical operation is first performed on the initial two inputs and the resultant *if-then-else* obtained. The logical operation is then performed on this structure and the next gate input. This process of combining the resultant *if-then-else* structure from previous gate inputs and the next gate input's *if-then-else* structure is continued until all gate inputs have been processed and the final *if-then-else* structure obtained.

The output of a logical operation such as AND or OR between nodes F and G presented in *if-then-else* form in Equation 4, is given by Equation 5. Equation 5 assumes independence between the events represented by the node variables and is extended to phased missions and multiple failure modes in [8, 9, 10] and later in this paper.

$$\begin{aligned} F &= ite(x, F_{x=1}, F_{x=0}) = ite(x, F1, F0) \\ G &= ite(y, G_{y=1}, G_{y=0}) = ite(y, G1, G0) \end{aligned} \tag{4}$$

Where $R_{s=t}$ represents node R with all instances of variable s (including its children) mapped to value t and $t \in \{1,0\}$.

$$ite(x, F1, F0) \oplus ite(y, G1, G0) = \begin{cases} ite(x, F1 \oplus G1, F0 \oplus G0) & index(x) = index(y) \\ ite(x, F1 \oplus G, F0 \oplus G) & index(x) < index(y) \\ ite(y, F \oplus G1, F \oplus G0) & index(x) > index(y) \end{cases} \tag{5}$$

Where \oplus represents a logical operation (AND or OR) and $index(x)$ is the order of the Boolean variable x .

Figure 4 shows a fault tree and its equivalent BDD representation.

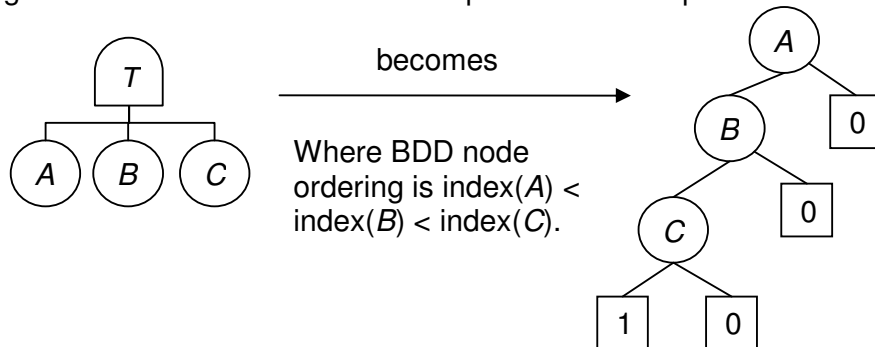


Figure 4 – An example of fault tree to BDD conversion

The fault tree to BDD conversion can be carried out by an algorithm that performs a depth-first left-to-right traversal of the fault tree, converting each fault tree node encountered into its equivalent BDD representation. To demonstrate this process, the step by step conversion of the fault tree shown in Figure 4 will now be explained. The fault tree consists of four nodes, namely the top AND gate node, which we shall denote as node T , and its three basic events which we shall denote as nodes A , B , and C following their labelling in the figure. According to the traversal strategy the leftmost node, node A , is visited first and converted into its BDD representation. Next, node B is visited and converted into its BDD representation. Since nodes A and B are siblings of an AND gate, the next step is to perform the Boolean AND operation between their BDD representations, resulting in a new BDD node, which can be denoted as node z (denoted in lower case here to avoid confusion with the capitalised fault tree node labels). Although z does not represent any of the fault tree nodes (A , B , C or T) its creation is a necessary intermediate step in the representation of T . The next node to be visited is node C and, again, this is converted into its BDD representation. Since node C is another child of the AND gate, the final step is to perform the Boolean AND operation between C 's BDD representation and z . The result of this operation is the BDD representation of the top gate of the fault tree, since all the fault tree nodes have been processed. Alternatively, the conversion in terms of the *if-the-else* notation is shown below:

$$\begin{aligned} \text{Top Event} &= \text{ite}(A,1,0) \cap \text{ite}(B,1,0) \cap \text{ite}(C,1,0) = \text{ite}(A, \text{ite}(B,1,0), 0) \cap \text{ite}(C,1,0) \\ &= \text{ite}(A, \text{ite}(B, \text{ite}(C,1,0), 0), 0) \end{aligned}$$

The chosen variable ordering can have a significant impact on the size of the resultant BDD although this particular problem is not considered in this paper, instead see Bartlett and Andrews [17].

Improved BDD Methods for phased mission analysis

This section introduces several new improvements to the BDD method for the analysis of phased missions. Different forms of the BDD method are developed for analysing systems with only non-repairable components, only repairable components and those consisting of a combination of both. Each of these methods is described in turn in the following sections.

For non-repairable component systems

Non-repairable components are defined as components that remain failed for the duration of the mission upon initial failure. Much research has been carried out to maximise the efficiency of the BDD method with application to non-repairable systems in phased missions. In this section further new improved techniques are introduced.

BDD Construction

In this work the methodology for encoding the system reliability structure from phase fault trees containing only non-repairable components into a BDD is similar to the approach taken by Tang and Dugan [10] but with some important changes that increase solution efficiency.

The algorithms presented by Tang and Duggan order variables first by component, then phase and then failure mode. Instead the methods and algorithms in this paper assume variables are ordered first by component, then by failure mode and finally by phase, such that earlier phases have a lower index (forward phase ordering). For non-repairable components the end of the phase which failure occurs after will be 0, the start of the mission, and hence the labelling of the basic events will be in the form $I_j(0, p)$. The phase ordering will therefore only involve comparing the failure before end phases. This ordering when used in conjunction with the BDD construction algorithm described later ensures the BDD does not contain paths with mutually exclusive events. In turn, this allows the efficient creation of the path data structures, developed later in this paper, to be used. For example, a system consisting of two components, A and B , each with two failure modes, 1 and 2, when operating in a two phased mission would have its Boolean variables ordered $A_1(0,1) < A_1(0,2) < A_2(0,1) < A_2(0,2) < B_1(0,1) < B_1(0,2) < B_2(0,1) < B_2(0,2)$. Modifications to convert the methods and algorithms for use with a backward phase ordering are simple but have been left out due to space restrictions. Unlike the Tang and Duggan method the choice of forward or backward phase ordering makes no difference to the performance of the method.

The BDD is constructed from the mission fault tree, as before, by converting the fault tree basic events into BDD nodes and then applying Boolean logical operations between BDD nodes, including the nodes resulting from previous computations, according to the logical gates in the fault tree.

A non-repairable component remains in a failure mode for the remainder of the mission upon entry and therefore a basic event for component I and failure mode j appearing in the phase p fault tree is converted into *if-then-else* format as: $A = ite(I_j(0, p), 1, 0)$.

The two nodes F and G , shown in *if-then-else* format in Equation 4, will be used to explain the application of logical operations between BDD nodes in the BDD construction. If F and G have variables that are of equal index in the ordering scheme then the rule from Equation 5 still applies, otherwise, the following rules assume that node F is the node with the lower index.

Applying a logical operation \oplus (e.g. AND or OR) between these two nodes will output a node representing the result. As with any other BDD node, this node is defined in terms of its variable and its failure and success child nodes. Since x and y are the variables of nodes F and G respectively, then since $\text{index}(x) < \text{index}(y)$ due to F 's definition above, the resultant node's variable will be x . We will now denote the failure child and success child of the

resultant node as U and V respectively, leading to the if-then-else representation of the output from the Boolean operation shown in Equation 6.

$$F \oplus G = \text{ite}(x, U, V) \quad (6)$$

where $U = (F \oplus G)_{x=1} = F_{x=1} \oplus G_{x=1}$ and $V = (F \oplus G)_{x=0} = F_{x=0} \oplus G_{x=0}$.

All that remains is to compute nodes U and V . V is calculated from Equation 5 as before, i.e. $V = F0 \oplus G$. The method to compute node U depends on the Boolean variables from nodes F and G and a set of rules are shown in Table 1, replacing those from Equation 5 (which assume variable independence). The relationship between the variables of nodes F and G , namely x and y , determine which of the exhaustive set of rule conditions is met and accordingly, which of the rules to apply in the computation of node U .

Rule	U and V computation	Condition
1	$U = F1 \oplus G1$	If x and y are variables belonging to the same failure mode of the same component.
2	$U = F1 \oplus G0_{x=1}$ where $G0_{x=1}$ is equal to $G0$ if $G0$'s variable belongs to a different component to x , otherwise it is the first node with a variable belonging to a different component encountered during a traversal down the success children of the BDD starting from $G0$.	If x and y are variables belonging to different failure modes of the same component.
3	$U = F1 \oplus G$	Otherwise.

Table 1 - Rules for computing nodes U and V

The reasoning behind each rule from Table 1 is now explained:

Rule 1: If x and y are variables of the same failure mode of the same component, then the phase of x must be earlier than or equal to the phase of y (due to the variable ordering) and $x=1$ implies $y=1$ since the failure mode is non-repairable. This implies that the resultant node's failure child, node U , should be formed from the computation of the failure children of node U and V as shown in rule 1 of Table 1.

Rule 2: If x and y are variables belonging to the same component but different failure modes then $x=1$ implies that y , and any other variable belonging to a different failure mode of the same component as x , equals 0. This is true since a component's failure modes are mutually exclusive. The failure child of the resultant node, U , is therefore computed from the failure child of node F ,

$F1$, and the first success child of node G which has a variable not belonging to the same component as x , $G0_{x=1}$. The recursive nature of finding $G0_{x=1}$ through the success child traversal is shown by Equation 7 where $H_{x=1}$ is determined until its variable belongs to a different component to x at which point it becomes equal to $G0_{x=1}$.

$$H_{x=1} = (w \cdot H1 + \bar{w} \cdot H0)_{x=1} \text{ if the variable of } H \text{ belongs to the same component as } x \quad (7)$$

Where $H = \text{ite}(w, H1, H0)$, initially $H = G$

$H_{x=1} = G0_{x=1}$ when the variable of H belongs to a different component to x .

Where \bar{w} is the complement of w .

In order to illustrate the procedure to determine $G0_{x=1}$ an example is considered below.

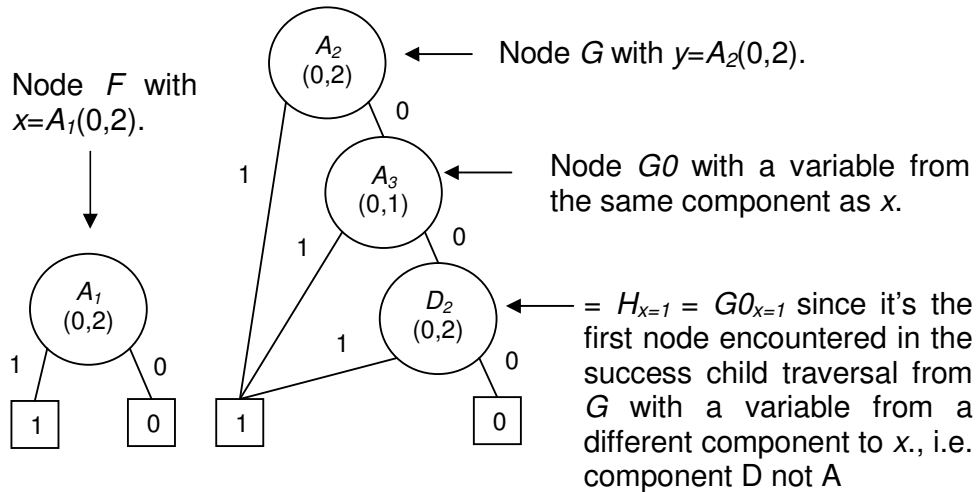


Figure 5 - Example application of Equation 7

In the example shown in figure 5

$$F = \text{ite}(A_1(0,2), 1, 0)$$

$$G = \text{ite}(A_2(0,2), 1, (\text{ite}(A_3(0,2), 1, (\text{ite}(D_2(0,2), 1, 0))))))$$

To determine $G0_{x=1}$ equation (7) is applied. As $x=1$, i.e. $A_1(0,2)=1$, then $A_j(0,p)=0$ for all $j \neq 1$. Initially $H=G$ hence $w=A_2(0,2)$, $H1=1$, $H0 = \text{ite}(A_3(0,2), 1, (\text{ite}(D_2(0,2), 1, 0)))$. Applying equation (7) gives:

$$\begin{aligned} H_{x=1} &= (H0)_{x=1} \\ &= (\text{ite}(A_3(0,2), 1, (\text{ite}(D_2(0,2), 1, 0))))_{x=1} \end{aligned}$$

The variable of H , $A_3(0,2)$, belongs to the same component as x , hence equation (7) must be applied again with $w= A_3(0,2)$, $H1=1$, $H0=\text{ite}(D_2(0,2), 1, 0)$

$$\begin{aligned} H_{x=1} &= (H0)_{x=1} \\ &= (\text{ite}(D_2(0,2), 1, 0))_{x=1} \end{aligned}$$

In this case the variable of H , $D_2(0,2)$, belongs to a different component to x and hence $G0_{x=1}=\text{ite}(D_2(0,2), 1, 0)$

If $G0$ in the Figure 5 example had a variable from a different component to x , such as $C_7(0,1)$, then $G0_{x=1}$ would instead be $G0$.

Rule 3: In all other cases, x and y belong to different components and are independent, therefore U is calculated according to Equation 5.

Unlike the method and ordering scheme presented in [10], the BDD produced by this method and ordering scheme avoids encoding paths containing mutually exclusive events which allows faster evaluation algorithms to be implemented.

A new reduction rule introduced in this paper, is for the case where x belongs to the same failure mode as node V 's variable, defined here as variable z , such that $x=1$ implies that $z=1$, and $V_{z=1}$ and U are the same node. In this case this node can be replaced by its success child, node V . Figure 6 shows an example of the application of this reduction rule. The top node in Figure 6a is an example of a BDD that was output from an OR operation between two nodes F and G , $FORG = ite(A_2(0,1), U, V)$, where the reduction rule was not used. The reduction rule can be applied to this BDD since the top node and its success son, node V , both have variables from failure mode A_2 and both have node U as their failure child. Applying the reduction rule replaces the top node in Figure 6a with its success child, node V , resulting in the smaller BDD shown in Figure 6b.

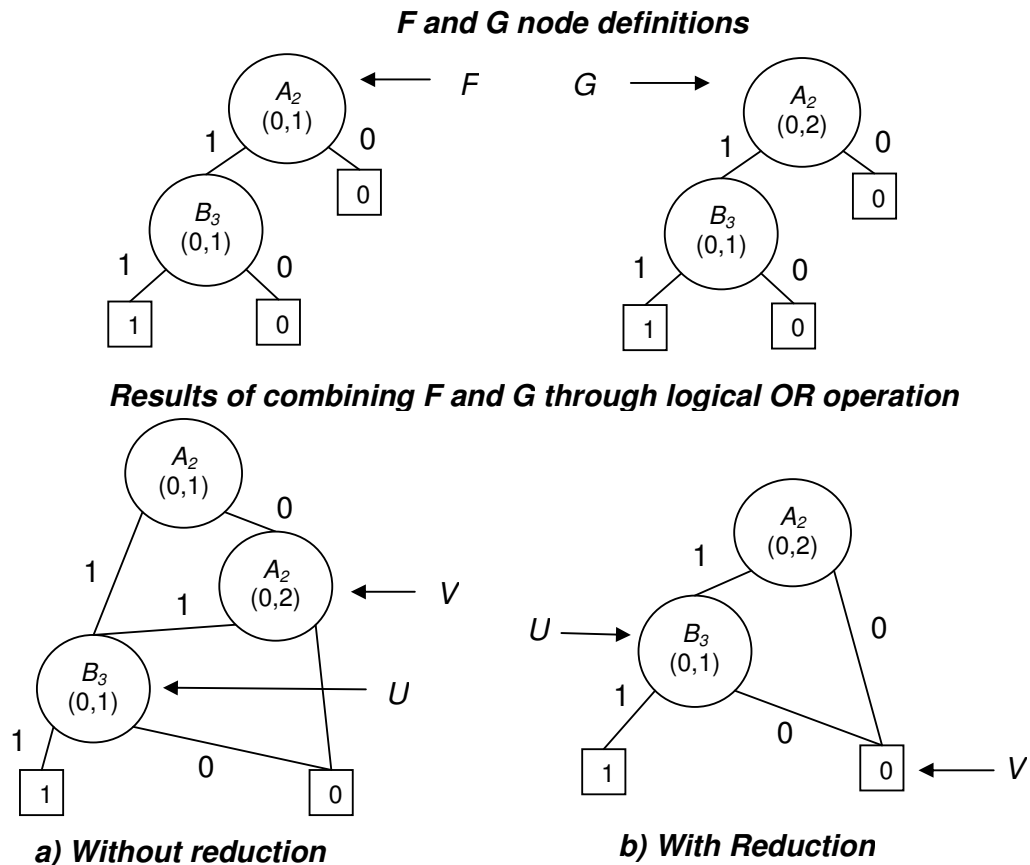


Figure 6 – BDD computation result reduces from a) to b) due to reduction rule

This reduction rule often significantly reduces the number of nodes and paths in the BDD, increasing evaluation speed and reducing the amount of memory required.

If, as is often the case, a BDD path contains multiple basic events associated with the same component then there will exist dependencies between them. These dependencies are caused by the phased and multiple failure mode nature of the basic events. In order to evaluate path probabilities these dependencies must be resolved by transforming the basic event combinations into a set of independent events such that the path probability can be found by summing and multiplying the probabilities of the individual events in the set.

Existing methods such as those presented in [9, 10] use algorithms that traverse the BDD to resolve dependencies with variables in lower nodes during the probability evaluation. These algorithms involve comparisons between node Boolean variables in order to identify the potential dependencies as well as potentially long traversals to distant dependant nodes. Due to the inherent coupling between the probability evaluation and dependency resolution, the dependencies must be resolved each time the BDD probability is evaluated (for example, when re-evaluating the BDD due to a change in failure mode failure rates). In this paper a new method is introduced where a new dependency free data structure, named the implicant tree, is formed prior to evaluation. The absence of dependencies during probability evaluation permits a substantial increase in speed – particularly during repeat evaluations such as when calculating importance measures or when performing real time analysis.

During the construction of the implicant tree a set of rules are used to resolve dependencies between events associated with the same component. The four rules were derived by modifying a subset from a larger set of rules originally designed for accident sequence analysis developed by Kohda, Wada and Inoue [3]. They are all that is necessary to reduce a BDD path into a set of independent events.

The first two reduction rules, given by Equation 8 and Equation 9, are used to resolve phase dependencies between events from the same failure mode.

$$P(I_j(0, p_1) = 0 \cap I_j(0, p_2) = 1) = P(I_j(p_1, p_2) = 1) \quad (8)$$

$$P(I_j(0, p_1) = 0 \cap I_j(0, p_2) = 0) = P(I_j(0, p_2) = 0) \quad (9)$$

where $p_1 < p_2$.

The third and fourth rules, given by Equation 10 and Equation 11, are used to resolve dependencies between events associated with different failure modes from the same component:

$$P(I_{j_1}(0, p_1) = 0 \cap I_{j_2}(p_2, p_3) = 1) = P(I_{j_2}(p_2, p_3) = 1) \quad (10)$$

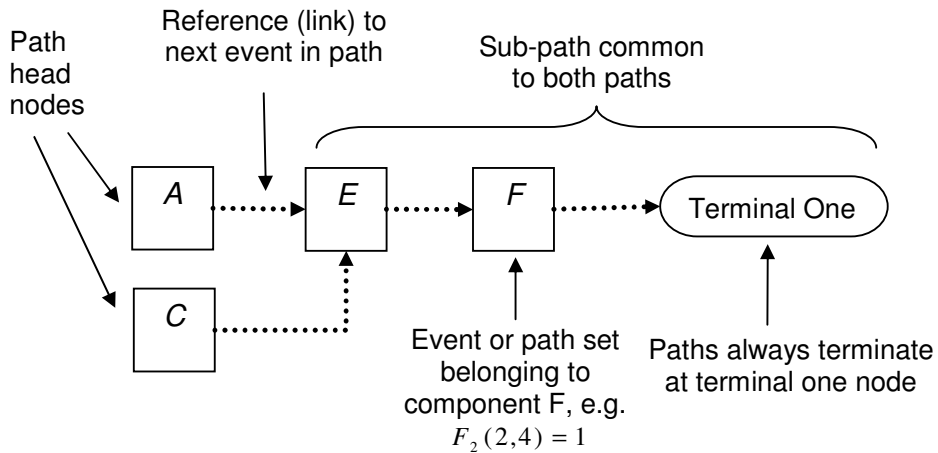
$$\begin{aligned} P(I_{j_1}(0, p_1) = 0 \cap I_{j_2}(0, p_2) = 0 \dots \cap I_{j_n}(0, p_n) = 0) \\ = 1 - P(I_{j_1}(0, p_1) = 1) + P(I_{j_2}(0, p_2) = 1) \dots + P(I_{j_n}(0, p_n) = 1) \end{aligned} \quad (11)$$

where $j_1 \neq j_2 \dots \neq j_n$.

Equation 8 states that the probability of a component not failing in a certain failure mode before the end of a phase, but failing in that failure mode before the end of some later phase is equal to the probability of failure between the two phases. Equation 9 states that the probability that a component does not fail in a certain failure mode before the end of two different phases is equal to the probability that it does not fail prior to the end of the later phase. Equation 10 states that the probability a component does not fail in a failure mode before the end of a certain phase but fails in a different failure mode belonging to the same component at some point between the end of two phases is equal to the probability of this latter event alone. Equation 11 states that the probability of a component not failing in a set of its failure modes before specific phases is equal to 1 minus the sum of the probabilities of each of the failure modes occurring before the end of the phase that they are successful until.

The only other combinations that can occur on a BDD path are between events belonging to different components and since these are already independent no reduction rules are necessary.

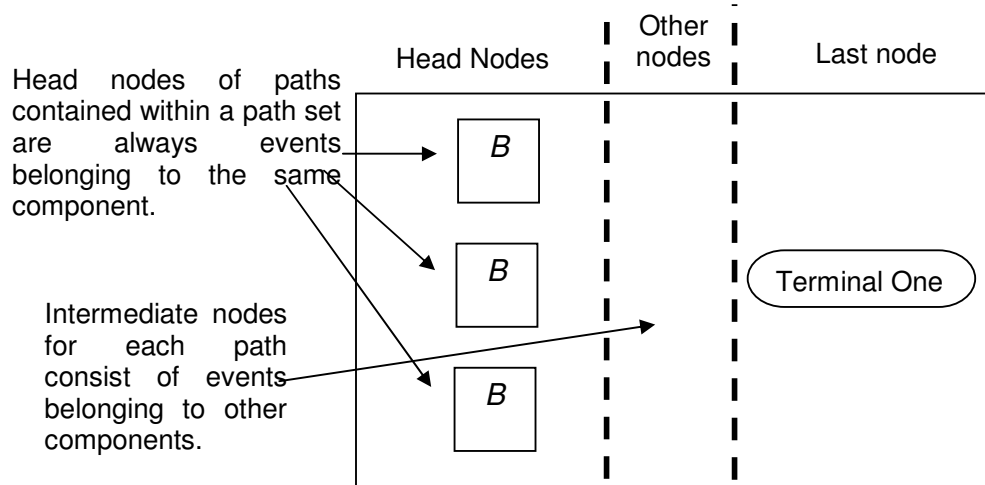
The data structure encoding the dependency resolved set of events on a BDD failure path is now described. A linked list type structure is used where each node of the list represents an event belonging to a different component. Due to this structure, the last event of the list, like the last event of the represented failure path, is a terminal one node. Multiple failure mode success events from the same component, despite involving multiple basic events in resolved form (see Equation 11), are stored on a single node of the list structure. The structure uses shared tail nodes so that common tail sections of a path are not repeated. Since all paths, by definition of a path, terminate at a terminal node, they will all share a common terminal one node. An example of this structure is shown in Figure 7.



Data structure represents a path as a series of events belonging to different components in the form of a shared tail node linked list.

Figure 7 - The path data structure

The event stored in the head node of each path data structure in the set of paths for a particular BDD node will be an event belonging to the node's Boolean variable component. The set of paths for a BDD node can therefore be stored as a list of path structures with head node events belonging to the same component. An example of this data structure is shown in Figure 8.



Data structure for a path set is a list of paths (see Figure 7) with head nodes representing events belonging to the same component.

Figure 8 - The path set data structure

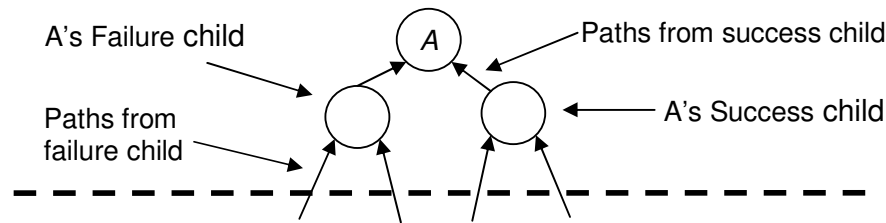
The path data structure described above and shown in Figure 7 is now extended such that a node can represent a path set data structure as well as an individual event. Since the path set data structure itself consists of a set of paths, it can therefore contain further path set data structures. This data structure is the implicant tree that encodes the implicants for any node in the

BDD. The implicant tree representing the paths for a particular BDD node is obtained by following the procedure outlined below:

If the node is a terminal one node, then return an implicant tree containing a single path that has only one node and with the node representing the terminal one event. If it is a terminal zero node then an empty implicant tree, containing no paths, is returned. Otherwise the BDD node must be an intermediate node and steps *A*, *B*, *C* and *D* outlined below should be followed.

Step A – get the path sets from each of the child nodes. In turn each child node will build a path set from its children - this is therefore a recursive process that ends at the terminal nodes.

This step is shown in Figure 9



Binary Decision Diagram continues to terminal nodes.

Figure 9 - Step A of data structure derivation

Step B – create the events to combine with the path sets from each child node.

1. The event to combine for the failure (1) child is $I_j(0,q) = 1$
2. The event to combine for the success (0) child is $I_j(0,q) = 0$.

where *I* is the node variable's component, *j* its failure mode and *q* its phase.

An example of this step is shown in Figure 10.

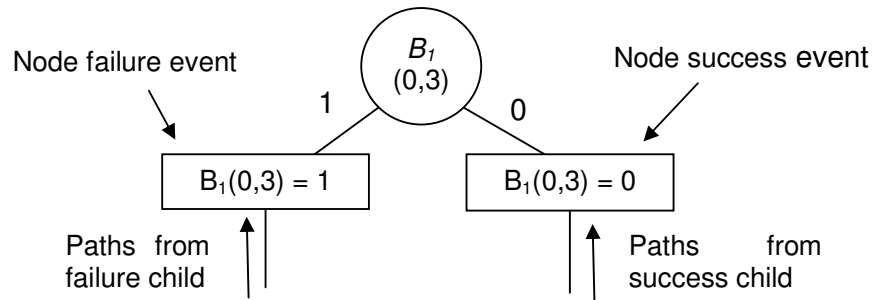


Figure 10 - Step B of data structure derivation

Step C – derive a new set of paths by combining each path in a child's paths set with the appropriate event generated in step B.

1. If the node's variable belongs to the same component as the child's variable, then a dependency exists between each head event in the path set, $X_1...X_n$, where n is the number of head events, and event generated for this node, Y . In this case, use 2a to generate the new path set. Otherwise use 2b (this is always the case for the paths from the failure child, due to the BDD construction rules discussed earlier).
2.
 - a. For $i = 1$ to n , combine Y and X_i according to the rules for combining dependant events given by Equation 8 to 11.
 - b. Create new path with Y as the head event, linking to the path set as the subsequent event.

Step D – combine the paths derived in step C into a single path set. This is the path set belonging to this node and returned to any parent node in the BDD.

Since the top node of a BDD represents the BDD itself, the full set of mutually exclusive implicants for a BDD is found by using the above procedure on its top node.

An implicant tree data structure requiring less memory and offering even faster repeat evaluation, at the expense of increased computational effort during construction, can be formed but is omitted from this paper due to space restrictions.

BDD Evaluation

Once the implicant tree representing the paths through the BDD has been constructed the evaluation of mission probability is simple and fast. To carry out the system failure probability evaluation the following 2 rules are followed:

For a path set – sum the probabilities of each path in the set.

For a path – multiply the probability of each event on the path. Note that an event on the path can be a path set (which is evaluated through the path set evaluation rule).

Each node in a path data structure from the implicant tree, which represents either a subsection of a path or set of paths, may be linked to, and thus shared, by nodes from multiple path data structures. Its value is therefore cached in the node upon first evaluation so that repeat computations are avoided. If the BDD is re-evaluated, these cached values may no longer be correct (e.g. phase durations may have been altered) and so they are cleared.

The evaluation stage has been greatly simplified by the resolution of dependencies during the construction of the data structure.

For repairable component systems

The BDD for repairable systems is constructed in a different manner to the non-repairable BDD described in the preceding section. The repairable BDD is used to solely represent the systems reliability structure, or specifically, its Boolean reliability structure function. Evaluation of probabilities is not carried out through BDD paths but instead through the Markov method.

BDD Construction

A separate BDD is built from each of the phase fault trees. The BDD is built using the standard algorithm and the rules given in Equation 5 except when computing the result of a logical operation between nodes with Boolean variables from different failure modes belonging to the same component. In that case rule 2 from Table 1 is used instead. A phased mission for a repairable system is therefore represented by a set of BDDs - one for each phase in the mission.

Evaluation

Distinct from the non-repairable BDD case, direct evaluation from the BDDs is not possible. Instead the BDDs have an auxiliary role during evaluation; they are used as a tool to determine which states in a Markov model represent a system failed state during a particular phase.

The Markov method

A continuous time, discrete space, Markov chain is used to determine the probability of component states at each time point in a phased mission. The method used is based upon that of Kim and Park [13] and extended to multiple failure modes.

Forming and Evaluating Markov models from a repairable BDD

The set of components from the BDD are used to determine the exhaustive set of Markov states such that a state exists for every possible combination of component states. Since failure modes are mutually exclusive, no state can have multiple failure modes from the same component present. The total number of states can be determined from Equation 12.

$$N_{ms} = \prod_i^{N_C} (fm_i + 1) \quad (12)$$

Where N_{ms} is the number of Markov states, N_C is the number of components represented in the BDD and fm_i is the number of failure modes for component i .

A N_{ms} by N_{ms} matrix is now formed, known as the transition matrix, containing the transition rates between the Markov states. The element located at row i

and column j (element (i,j)), where $i \neq j$, of the transition matrix represents the transition rate from Markov state j to Markov state i . Element (i,i) is the negation of the sum of the other elements from column i and represents the transition rate out of state i .

From the full set of Markov states a reduced set can be formed for each phase of the mission by removing all states that represent system failure in that phase. The reduction is possible because the system failure states are absorbing, since the mission ends upon failure, and therefore transition out of these states is impossible whilst transitions into them are contained within the diagonal elements of the transition matrix.

The Markov states representing system failure in each phase are determined from the phase's repairable BDD. Checking the system state for each of the Markov states is straightforward. The top node for the phase BDD is entered and the failure mode of its Boolean variable is checked against the set of present component failure modes defining the Markov state. If the failure mode exists in the Markov state then the BDD node's failure (1) child is next to be evaluated against the Markov state, otherwise the success (0) child is checked next. This continues down the BDD until a terminal node is reached, the value of which indicates the state of the system in that phase for the Markov state. Specifically, reaching the BDDs terminal 1 node indicates that the Markov state represents a system failure state whilst reaching the terminal 0 node indicates that the Markov state represents a system success state.

A N_{ums_i} by N_{ums_i} transition matrix for phase i is formed, where N_{ums_i} is the number of states in the reduced set for phase i , known as the reduced transition rate matrix. This contains the transition rates between the reduced set of Markov states. The reduced transition rate matrix is formed by deleting the rows and columns from the transition rate matrix corresponding to the system failed states for the phase. Using the reduced state transition matrix benefits the computation of the model in terms of both memory usage and speed.

An initial conditions vector is formed containing the probability of being in each of the Markov states from the first phase transition matrix. Element i of the initial conditions vector is the probability of the system being in state i at the start of the mission. Due to the assumption that all components are in the working state at the start of the mission, the element corresponding to the Markov state of all components working will be 1 and all other entries will be 0.

The method proceeds by finding the initial conditions for each subsequent phase until the end of the mission is reached, at which point the reliability of the mission is found from the probability that the system resides in any of non-absorbing states from the final phase (i.e. those represented in its reduced transition matrix). Equation 13 is used to calculate the probability vector at the end of a phase. Many methods exist for solving the matrix exponential in Equation 13 and the choice can affect the numerical error introduced by computation, for more detail see Moler and Loan [18]. The software

implementation in this work uses scaling and a Padé approximation which gives accurate results.

$$\rho_i(t_i) = e^{(\beta_i t_i)} \rho_i(0) \quad (13)$$

where $\rho_i(t)$ is the probability vector at time t from the start of phase i , t_i is the duration of phase i and β_i is the transition rate matrix for phase i .

Due to the removal of absorbing states in each phase, the reduced transition rate matrix for each phase of the mission will contain a distinct set of Markov states. This means that the elements in the end of phase probability vector will not correspond to the elements of the initial probability vector of the following phase; in fact the vectors could even be of different size. The initial probability vector for the following phase is therefore formed by copying across the value from the end of phase vector to the element corresponding to the same Markov state. Elements in the initial probability vector corresponding to a system failed Markov state in the previous phase, and therefore not present in the previous phase's end of phase probability vector, are initialised to a value of 0. Another possibility is that an element in an end of phase probability vector does not represent a system failed state in that phase but does represent a system failed state in the next phase, and is therefore not represented in its initial probability vector. Summing these elements gives the probability of transitional failure between the phases.

Analysing Systems containing both repairable and non-repairable components

A commonly encountered class of phased missions involve systems containing a mixture of repairable and non-repairable components. Determining the reliability of such systems traditionally requires using a method optimised for repairable systems, such as the Markov technique discussed in the preceding section. However these methods could be considered unnecessarily expensive both in terms of resources and time due to the high computational resources used even for the non-repairable parts of the system that, in isolation, could be solved through methods optimised for non-repairable systems with far greater efficiency. To remedy this situation, a new method is introduced here that allows the reliability of systems containing both repairable and non-repairable components, including those operating in phased missions, to be analysed in an efficient manner.

This new technique is derived from the methods for the analysis of non-repairable and repairable systems that were introduced earlier. Fundamental to its power are the use of the BDD technique to prudently separate the repairable from the non-repairable elements of the system and an evaluation scheme that combines the direct, combinatorial, path based and Markov chain procedures.

Implicit to the method is an approximation of the situations in which failure of repairable components causes system failure, and as such the method gives approximate reliability results.

Fault tree to BDD conversion

A BDD formed from a system's mission fault tree is used to represent its mission reliability structure, isolate the repairable from the non-repairable elements of the system and directly evaluate the reliability of the non-repairable parts.

The first step is to set the ordering scheme so that variables belonging to the system's repairable components, components with one or more repairable failure modes, are indexed higher than its non-repairable components (and thus appear as lower nodes in the BDD).

The BDD is then built using the non-repairable method described earlier for Boolean operations between two nodes, using the rules from Table 1, except when both nodes have Boolean variables from repairable components. In the case of a Boolean operation between two nodes with Boolean variables from repairable components then the same method as when building a repairable BDD methods applies – i.e. Equation 5 and rule 2 from Table 1.

For systems containing both repairable and non-repairable components this will result in a BDD where the higher nodes are composed like a non-repairable BDD and then may make the transition into repairable BDDs before the terminal nodes are reached. Full repairable BDDs have a root node for each of the phases in the mission as explained earlier, but repairable BDDs appearing in a hybrid BDD may not have root nodes for certain phases. If the repairable BDD is reached through paths consisting of events from non-repairable components such that failure in certain phases is not possible then these phases will not be represented in the repairable BDD. It does not matter where the repairable basic events appear in the fault trees; the technique ensures that the non-repairable and repairable sections are grouped together in the BDD in a correct and efficient manner. Due to the property of BDDs that ensures variables that do not matter are omitted, the sizes of the repairable BDDs are minimised. The hybrid BDD for the example mission from Figure 1 is shown and annotated in Figure 12. The BDD in Figure 12 was formed using an ordering scheme that ordered components as $B < A < C < D < E < F$ and ordered failure modes as $2 < 1$.

Evaluation

The evaluation procedure for the model gives approximate results for the reliability of a phased mission. The paths through the BDD are represented by an implicant tree, as they were in the non-repairable model introduced earlier, but now paths may include nodes representing repairable BDDs. The evaluation of a path in the implicant tree is then calculated as before, i.e. as the product of the path node probabilities. A node representing a repairable BDD is evaluated through the method discussed in the repairable BDD section. Where the repairable BDD contains non-consecutive phases, for example phase BDDs for phases 1 and 3 but not for 2, then the time between present phases is bridged using the full transition matrix (i.e. the transition matrix without states removed) and the absent phase duration.

Since in this procedure the unreliability evaluation of Boolean variables belonging to non-repairable components considers only the phase but not the time of failure, the evaluation of paths with Boolean variables from both repairable and non-repairable components is not exact.

Software implementation and a worked example

A software tool has been developed that implements the techniques discussed in this paper. The inputs to the tool are the phase fault trees and component failure mode failure and repair models. This tool was used to analyse the example system's mission, as represented by the fault tree in Figure 1.

The BDD generated by the software is shown in Figure 12. The BDD shown in Figure 12 contains a total of 3 repairable BDDs and evaluation therefore requires the solution of 3 Markov models. Each of these Markov models is small, the largest consisting of three components (D , F and E) over two phases (2 and 3).

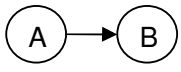
A worked example of the evaluation of the BDD paths that include the repairable BDD on the right in Figure 12, is given here:

As shown in Figure 12 the repairable BDD on the right contains a single phase, phase 2, and two failure modes, D_1 and F_1 . The first step is to determine the full set of states for the Markov model of the components D and F , which are shown in Table 2. The Markov model showing the possible transitions between the states in Table 2 is shown in Figure 11.

State IDs	Failure Modes (1=Present, 0=Absent)			
	D_1	D_2	F_1	F_2
1	0	0	0	0
2	1	0	0	0
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	1	0	1	0
7	1	0	0	1
8	0	1	1	0
9	0	1	0	1

Table 2 - Markov States

State Label	Present Failure Modes
1	None
2	D_1
3	D_2
4	F_1
5	F_2
6	D_1, F_1
7	D_1, F_2
8	D_2, F_1
9	D_2, F_2

where  = transition from state A to state B

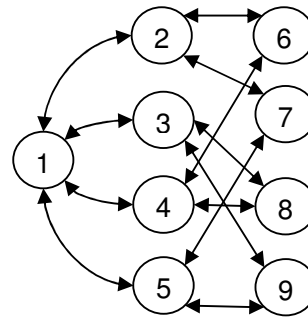


Figure 11 - Markov model for the repairable BDD on the right in Figure 12

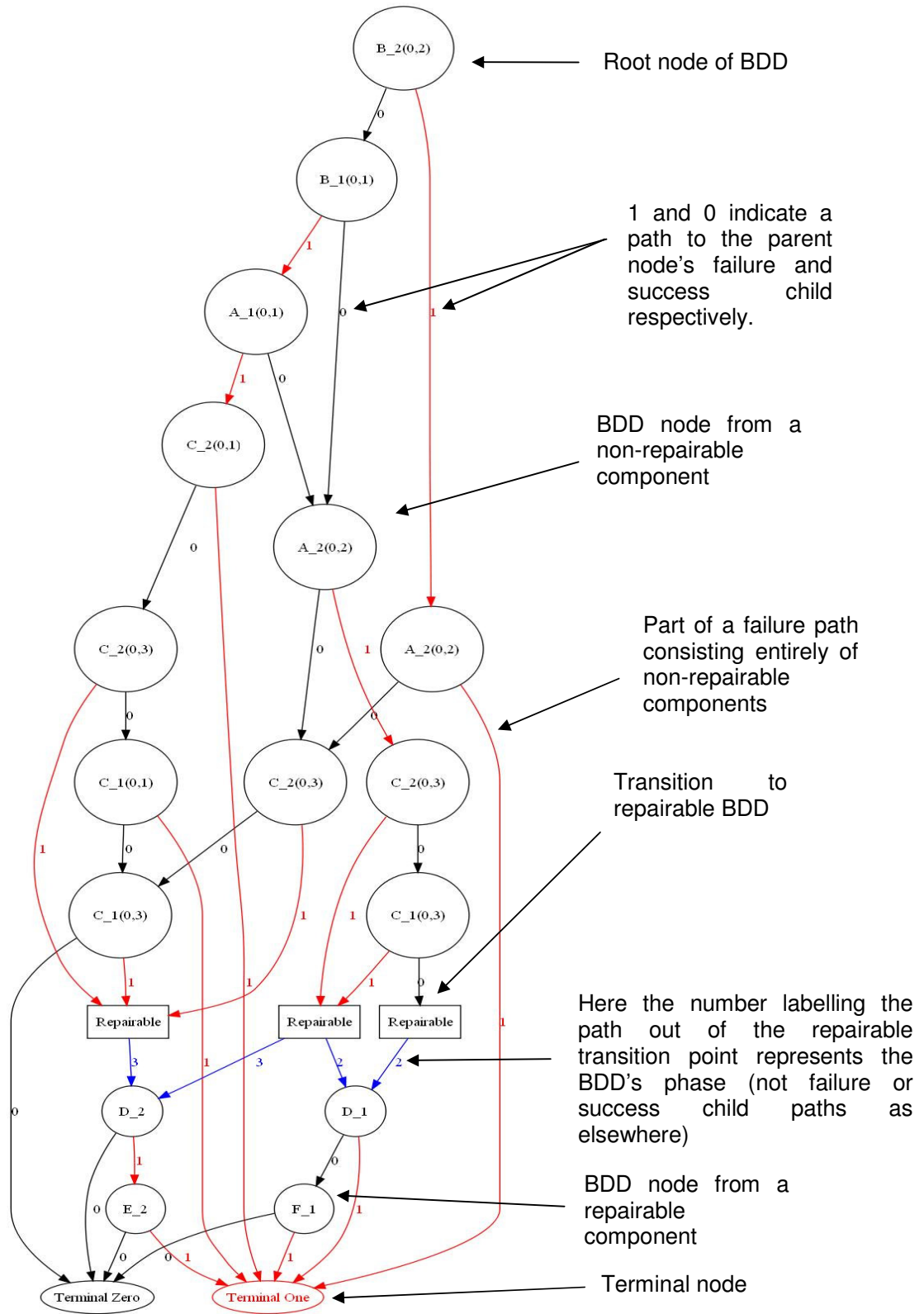


Figure 12 - A hybrid BDD representing mission failure of the example phased mission

State IDs	Phase 1 State Index	Phase 2 State Index
1	1	1
2	2	System Failed State
3	3	2
4	4	System Failed State
5	5	3
6	6	System Failed State
7	7	System Failed State
8	8	System Failed State
9	9	4

Table 3 - Markov state to state transition rate and state probability vector index for each phase

At the start of the mission it is assumed that all components are in the working state, thus the probability of the system residing in state 1 from Table 2 at time 0 is 1 and the probability of the system residing in any other state is 0. This results in the state probability vector for the start of phase 1 as given by Equation 14.

$$P_1(0) = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T \quad (14)$$

Element i in Equation 14 represents the probability of being in the state indexed i .

The BDD contains no nodes for phase 1 and therefore the full transition matrix is used to find the state probability vector at the end of phase 1, as shown in Equation 15.

$$\beta_1 = \begin{bmatrix} 0.004 & 0.01 & 0.01 & 0.01 & 0.01 & 0 & 0 & 0 & 0 \\ 0.001 & -0.012 & 0 & 0 & 0 & 0.01 & 0.01 & 0 & 0 \\ 0.001 & 0 & -0.012 & 0 & 0 & 0 & 0 & 0.01 & 0.01 \\ 0.001 & 0 & 0 & -0.012 & 0 & 0.01 & 0 & 0.01 & 0 \\ 0.001 & 0 & 0 & 0 & -0.012 & 0 & 0.01 & 0 & 0.01 \\ 0 & 0.001 & 0 & 0.001 & 0 & -0.02 & 0 & 0 & 0 \\ 0 & 0.001 & 0 & 0 & 0.001 & 0 & -0.02 & 0 & 0 \\ 0 & 0 & 0.001 & 0.001 & 0 & 0 & 0 & -0.02 & 0 \\ 0 & 0 & 0.001 & 0 & 0.001 & 0 & 0 & 0 & -0.02 \end{bmatrix} \quad (15)$$

Element (i,j) , $i \neq j$, in Equation 15 represents the transition rate from the state indexed j to the state indexed i and element (i,i) represents the transition rate out of the state indexed i , using the Markov state index for phase 1 shown in Table 3.

Substituting the initial state probability vector and state transition rate matrix for phase 1 into Equation 13, along with the phase duration of 100 hours, gives the end of phase 1 state probability vector shown in Equation 16.

$$P_1(100) = [0.7806 \quad 0.05145 \quad 0.05145 \quad 0.05145 \quad 0.05145 \quad 0.003391 \quad 0.003391 \quad 0.003391 \quad 0.003391] \quad (16)$$

Element i in Equation 16 represents the probability of being in the state indexed i in the Markov state index for phase 1 shown in Table 3.

For phase 2, the BDD shows that the system fails if component D is failed in failure mode 1 or component F is failed in failure mode 1. Thus, only states 1, 3, 5 and 9 from Table 2 represent working states in phase 2 and hence only the elements representing these states from the end of the phase 1 state probability vector, are used to form the initial state probability vector for phase 2 shown in Equation 17.

$$P_2(0) = [0.7806 \quad 0.05145 \quad 0.05145 \quad 0.003391]^T \quad (17)$$

Element i in Equation 17 represents the probability of being in the state indexed i in the Markov state index for phase 2 shown in Table 3.

The elements from the end of phase 1 state probability vector that represent system failure states in phase 2, i.e. states 2, 4, 6, 7 and 8, are summed to give the probability of failure on transition to phase 2 of 0.1131. The phase 2 state transition rate matrix is formed by removing from the full state transition rate matrix, Equation 15, all elements that represent transitions from, and to, phase 2 system failure states. The resultant reduced state transition rate matrix is shown in Equation 18.

$$\beta_2 = \begin{bmatrix} -0.004 & 0.01 & 0.01 & 0.0 \\ 0.001 & -0.012 & 0 & 0.01 \\ 0.001 & 0 & -0.012 & 0.01 \\ 0 & 0.001 & 0.001 & -0.02 \end{bmatrix} \quad (18)$$

The index for the states in phase 2, used in Equation 18, is given in Table 3.

Substituting the initial state probability vector and state transition rate matrix for phase 2 into Equation 13, along with the phase duration of 100 hours, gives the end of phase 2 state probability vector shown in Equation 19. The four states in this equation are those given in Table 3.

$$P_2(100) = [0.6164 \quad 0.05769 \quad 0.05769 \quad 0.005399]^T \quad (19)$$

Since no further phases are represented in the repairable BDD, deducting the sum of the elements in this state probability vector from 1 gives the unreliability for the Markov model of 0.2628.

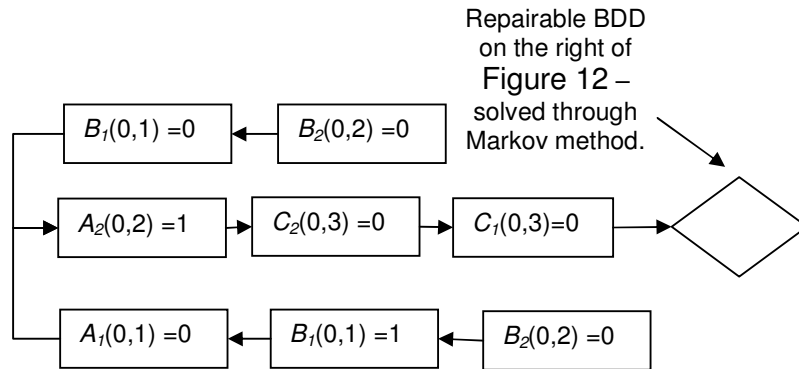


Figure 13 - Part of implicant tree data structure from example mission

The BDD in Figure 12 shows that there are two possible paths from the top node to the repairable BDD on the right of Figure 12. These are shown in the form of an implicant tree in Figure 13. The unreliability of these paths up to the repairable BDD, found using the non-repairable path evaluation procedure, is 0.0807. Multiplying this with the unreliability from the Markov model calculated earlier gives the total contribution to system unreliability from those paths including the rightmost repairable BDD of 0.02121.

Conclusions

An improved BDD build algorithm has been presented that produces a smaller BDD and therefore results in faster evaluation and reduced memory use. This is applicable to all phased mission BDD evaluation algorithms such as those in [10]. A new pre-evaluation data structure, the implicant tree, and techniques for its creation have been introduced that offer faster BDD evaluation – particularly when repeated evaluation is required. This offers great benefits in applications such as importance measure analysis and real time analysis. Finally, a new technique for analysing the reliability of systems operating in phased missions that contain both repairable and non-repairable components has been introduced. This technique uses a novel combination of the BDD and Markov methods to offer efficient analysis of such systems.

The techniques presented in this paper have been integrated into a software tool and successfully used to analyse an example system operating in a phased mission.

References

1. J. D. Esary and H. Ziehms, "Reliability of Phased Missions," *Reliability and Fault Tree Analysis*, Society for Industrial Applied Mathematics, pp. 213-236, 1975.
2. X. Dazhi and W. Xiaozhong, "A practical approach for phased mission analysis," *Reliability Engineering & System Safety*, vol. 25, pp. 333-347, 1989.

3. T. Kohda, M. Wada and K. Inoue, "A simple method for phased mission analysis," *Reliability Engineering & System Safety*, vol. 45, pp. 299-309, 1994.
4. A. K. Somani and K. S. Trivedi, "Boolean Algebraic Methods for Phased-Mission System Analysis," *Proceedings of Sigmetrics*, pp. 98-107, 1994.
5. R. La Band and J. D. Andrews, "Phased Mission Modelling using Fault Tree Analysis," *Reliability Engineering and System Safety*, vol. 78, pp. 45-56, 2002.
6. R. Bryant, "Graph based algorithms for Boolean function manipulation," *IEEE Trans. Computers*, vol. 35, pp. 677-691, 1977.
7. A. Rauzy, "New algorithms for fault trees analysis," *Reliability Engineering & System Safety*, vol. 40, pp. 203-211, 1993.
8. Z. Zang, D. Wang, H. Sun and K. S. Trivedi, "A BDD-Based Algorithm for Analysis of Multistate Systems with Multistate Components," *IEEE Trans. Computers*, vol. 52, pp. 1608, December 2003. 2003.
9. Z. Zang, H. Sun and K. S. Trivedi, "A BDD-Based Algorithm for Reliability Analysis of Phased-Mission Systems," *IEEE Transactions on Reliability*, vol. 48, pp. 50, March 1999. 1999.
10. Zhihua Tang and J. B. Dugan, "BDD-based reliability analysis of phased-mission systems with multimode failures," *Reliability, IEEE Transactions on*, vol. 55, pp. 350-360, 2006.
11. M. Alam and U. M. Al-Saggaf, "Quantitative Reliability Evaluation of Repairable Phased-Mission Systems Using Markov Approach," *IEEE Transactions on Reliability*, vol. 35, pp. 498, 1986.
12. M. K. Smotherman and K. Zemoudeh, "A Non-homogeneous Markov Model for Phased-Mission Reliability Analysis," *IEEE Transactions on Reliability*, vol. 38, pp. 585, 1989.
13. K. Kim and K. S. Park, "Phased-Mission System Reliability under Markov Environment," *IEEE Transactions on Reliability*, vol. 43, pp. 301, 1994 June. 1994.
14. I. Mura, A. Bondavalli, X. Zang and K. S. Trivedi, "Dependability Modeling and Evaluation of Phased Mission Systems: a DSPN Approach," *Dependable Computing for Critical Applications*, vol. 7, pp. 319, 1999.
15. D. Wang and K. S. Trivedi, "Reliability Analysis of Phased-Mission System With Independent Component Repairs," *IEEE Transactions on Reliability*, vol. 56, pp. 540, September 2007. 2007.
16. C. E. Shannon, "A symbolic Analysis of relay and switching circuits," *Transactions of the AIEE*, vol. 57, pp. 713, 1938.
17. L. M. Bartlett and J. D. Andrews, "Efficient basic event ordering schemes for fault tree analysis," *Quality and Reliability Engineering International*, vol. 15, pp. 95, 1999.
18. C. Moler and C. Van Loan, "Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later," *SIAM Rev*, vol. 45, pp. 3, 03. 2003.