Dynamic and Dependent Tree Theory (D^2T^2): A Framework for the Analysis of Fault Trees with Dependent Basic Events

John Andrews and Silvia Tolo

Resilience Engineering Research Group, University of Nottingham

Abstract

Fault tree analysis remains the most commonly employed method, particularly in the safety critical industries, to predict the probability or frequency of system failures. Whilst it has its origins back in the 1960s, the assumptions employed in the majority of commercial fault tree analysis codes have not changed significantly since this time and restrict the ability of the method to represent features of the design, operation and maintenance of modern industrial systems. The inability to include general dependencies between the basic events, the requirement for invariant failure and repair rates, and the inability to account for complex maintenance strategies are major limitations.

This paper proposes a new fault tree analysis framework which can overcome these restrictions. Whilst retaining the fault tree structure to express the causality of the system failure, the internal calculation method is updated by exploiting features of the Binary Decision Diagram, Stochastic Petri Net and Markov methods. The key elements of the D^2T^2 algorithm are described in detail and the framework demonstrated through application to a case study example of a pressure vessel cooling system.

Keywords: Fault Tree Analysis, Binary Decision Diagrams, Dependency models, Petri Nets, Markov Models.

1.0 Introduction

Fault Tree Analysis has its origins back in the 1960's and is attributed to Watson from Bell Telephone Laboratory who used a logic tree structure when analysing the causes of an inadvertent launch of the Minuteman Intercontinental Ballistic Missile [1]. The time dependent mathematical framework, known as Kinetic Tree Theory (KTT) was added in 1970 by Vesely [2] from the Idaho National Nuclear Laboratory. Traditional fault tree analysis is performed in two stages. Stage 1 delivers the minimal cut sets, lists of necessary and sufficient basic events which cause the top event. The second stage quantifies the system failure mode, top event, probability or frequency. Calculation of importance measures [3-5] can also be performed to establish the contribution of each component, or minimal cut set, to the system failure mode, identifying weaknesses which can be addressed to improve the system performance.

Technological developments and business practices have made significant advances since the 1970s which are reflected in the design, operation and maintenance of modern engineering systems. The basic assumptions required by KTT limit the capability of fault tree analysis to perform accurate assessments of such systems. A significant limitation is that all basic events are assumed to occur independently and so the failure of one component does not affect the

likelihood of another. Dependencies are introduced into the system operation through many aspects such as standby systems, common cause failures and maintenance strategies. There is also a limited capability to account for the sequencies in which failures occur. In Addition, the implementation of KTT, in most commercial fault tree tools, assumes that components experience constant failure and repair rates. Typical models used to evaluate the failure probability of components, based on their maintenance, are:

Non-repairable components:

$$Q(t) = 1 - e^{-\lambda t} \tag{1}$$

Reactive Repair (unscheduled maintenance)

$$Q(t) = \frac{\lambda}{\lambda+\upsilon}\left(1 - e^{-(\lambda+\upsilon)t}\right) \tag{2}$$

Discovered Failures (scheduled maintenance)

$$Q_{Av} = 1 - \frac{\left(1-e^{-\lambda\theta}\right)}{\lambda\theta} \tag{3}$$

Where: Q, is the component failure probability, Q_{Av}, is the average component failure probability, λ, is the constant failure rate, υ, is the constant repair rate, θ, the inspection interval and t is time.

The failure intensity, w, is then given by:

$$w = \lambda(1 - Q) \tag{4}$$

It is common that failure rates are not constant, many systems are operated over periods where components experience wear-out, resulting in increasing failure rates. When failure occurs, repair is not a random process and, as such, it can be questioned if the repair rate is ever constant. In addition to these limitations imposed by constant transition rates between the working and failed states, these equations represent a very restricted view of the maintenance processes which are applied to modern systems.

A more detailed discussion of the situations which commonly occur in engineering systems and which are inadequately represented with KTT is given in section 5.

Other limitations with KTT include the need to exploit approximations. When the fault tree structure produces large numbers of minimal cut sets, it may not be possible to evaluate them all and, culling methods are used to identify those which have the most significant impact on the system performance, with cut-offs applied to either the minimal cut sets order or likelihood [6,7]. Approximations are also used to evaluate the top event probability and frequency [3].

Since the 1970s, fault tree analysis has been the subject of considerable research aimed to address these approximations and limitations. Binary Decision Diagrams (BDD) have been used to recode the Boolean equation represented by the fault tree into a disjoint form which enables exact quantification of the system performance without the need for approximations [8-11]. Included in this research has been approaches to establish efficient orderings for the basic events which will result in concise BDD representations [12,13]. BDDs are now used extensively in system failure modelling [14-16].

2.0 Modelling System Dynamics and Dependencies

Due to the occurrences of dependent failure events in systems reliability modelling, there has been a motivation, right from the development of the initial fault tree analysis methods, to overcome their limitation in this regard. Approaches to this vary. Some use a totally different method to fault tree analysis, such as: Petri nets [17-20], Monte Carlo Simulation [21] and Markov models [3, 22] to model the whole system. Alternatives have tried to retain the fault tree structure to represent the system failure causality and adapt the analysis methodology appropriately. These approaches are discussed separately below.

Using Petri nets, Monte Carlo Simulation and Markov models enable event sequences and dependencies to be represented. Whilst Markov approaches require the movement between system states to be governed by constant transition rates, the first two of these methods will allow any distribution of failure and repair times. However, there can be difficulties in analysing the whole system in this way. System failure events are generally rare and, as a result, simulation-based solution routines, including Petri nets, suffer from issues with the processing requirements to reach convergence to statistically significant results. Markov models can experience a state space explosion for even moderately sized systems. These methods are also limited in their ability to document the failure logic development in comparison to a fault tree, and limits their attractiveness where regulators need to review the assessments.

Of those approaches which retain the fault tree as the basis of the method, Dynamic Fault Trees [23-26] are the most well developed. They incorporate dependency types and dynamics into the fault tree framework. In this case the dependency types incorporated use specialist gate types such as SPARE (for dependencies due to spares or standby components) or SEQ (for sequences of events). These features can be applied when the dependent events appear below one gate type which can be solved in isolation, using a relatively small Markov model, and the results are then substituted back as a super-event which replaces the gate output event and remains independent of the rest of the fault tree structure. This is illustrated in Figure 1.

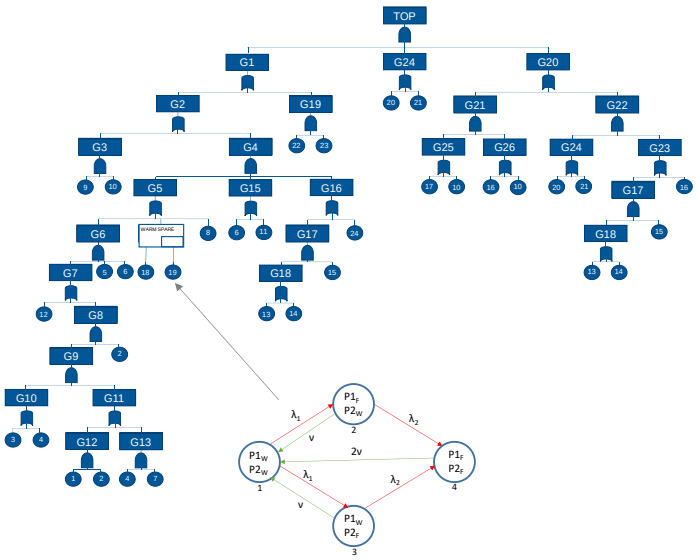


Figure 1 Warm Spare dependency and associated Markov model

Dependency types, occurring for example from maintenance, can affect events topologically distant from each other in the fault tree structure. Consider the example shown in Figure 2 where there is a dependency between basic events 27 and 29. It is possible to identify the highest gate in the fault tree (G22) under which all the dependencies occur. This section can again be analysed separately (using Markov or Petri nets as appropriate) and the result substituted back since it will remain independent. However, this dependency model also includes the basic events 26, 28 and 30, which are independent, and this makes the dependency model inefficient. In the extreme, when the dependent events are located at opposite sides of the fault tree, the highest gate containing the dependencies will be the top event which renders the approach inapplicable. Any approach which exploits Petri nets or Markov models to analyse the dependencies must be restricted to just include the dependent events alone.

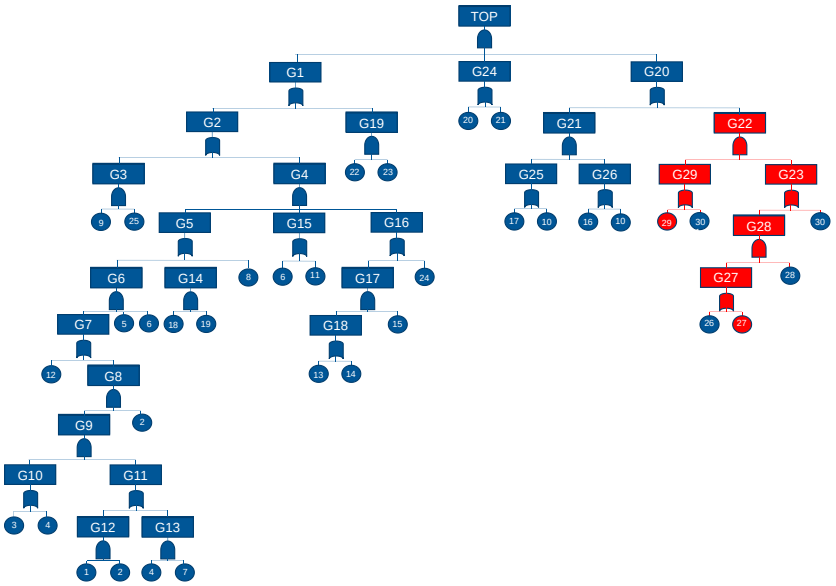


Figure 2 Geographically distant dependencies

There are a vast number of papers over the years which address the issues of dependency and pose a solution for a specific dependency type. However, there has been a problem in generalising these methods into a framework which can be incorporated with other such models and developed into a generic coverage of dependency types. As such these solutions have not found their way into the commercial codes utilised by industry.

3.0 Requirements of D^2T^2, Dynamic and Dependent Tree Theory

Dynamic and Dependent Tree Theory (D^2T^2) is developed and presented in this paper with the objective of overcoming some of the limitations of the traditional KTT. Specifically, the objectives were to:

- enable component failure and repair times to be represented by any probability distribution.
- incorporate the ability for dependencies of any type (due to system structure, operation or maintenance) to be accommodated between components or sub-systems
- facilitate the representation of complex maintenance processes to represent the sophisticated asset management strategies employed on modern systems.

- permit dynamics in the forms of event sequences to contribute to the system failure logic.

These objectives were formulated following discussions with industrial partners:

Aeronautical Industry: Rolls-Royce Aero Engines.

Nuclear Industry: Rolls-Royce Submarines, Small Modular Reactors Group.

Railway Industry: Rail Safety and Standards Board (RSSB), Network Rail, HS1, Network Rail High Speed.

As a result, the D^2T^2 framework requirements were formulated as:

- Retain the fault tree structure to represent the causality of the system failure in terms of the failure of its components, human errors, software failures etc. This representation is familiar to engineers, lends itself to the visualisation of the system failure causes and facilitates transparency, peer review and assessment by regulators. It also would enable fault tree models, evolved over many years to be upwardly compatible with D^2T^2. The new methodology should allow the analysis to be completed with as little manual involvement as possible and would be a direct replacement for the conventional KTT in the engine of the code.
- Where the system features complexities such as components with non-constant failure or repair rates, dependencies between component failure events, complex maintenance processes or event sequences which contribute to the system failure, these shall be analysed with the smallest possible Petri net or Markov model. This is required to maintain efficiency and the practicality of the approach for large systems analysis. So, no matter how far apart in the fault tree structure the dependent events were located, the dependency model should only feature those components. For the commonly encountered situations modelled the code would contain an appropriate Petri net or Markov template which would be parameterised from the input data. To ensure the generality and that one-off situations specific to one system can also be incorporated it was also possible to enter a particular Petri net or Markov model.
- It is necessary to develop a method which integrates the Petri net and Markov model solutions back into the fault tree structure for analysis. In the interests of efficiency and accuracy the fault tree section of the model which features independent component failure events will be analysed by converting the fault tree logic function to a Binary Decision Diagram (BDD). Since BDDs can feature a large number of paths, which represent disjoint component conditions which result in a system failure, it is necessary to keep these to a minimum to enable a practical means of integrating the dependency model results. This can be achieved by identifying the minimal sized sub-trees solved by a BDD. Current modularisation techniques account for dependencies due to repeated basic events in the tree structure. A modularisation approach which also accounted for dependencies between different events would need development.

This paper describes the D^2T^2 methodology developed to calculate the system (Top event) failure probability and failure intensity. Since the method retains the fault tree structure, the methods to deliver the minimal cut sets for qualitative fault tree evaluation remain

unchanged. With the ability to calculate the system failure probability and intensity, is unlocked by the means to evaluate component and min cut set importance measures. It is not possible to address these in a single paper and will be considered in a follow up publication.

This paper describes a summary of the fundamental building blocks used in the methodology and then goes on to describe the D^2T^2 code structure, how the independent sub-models are identified and the means by which the dependency results are integrated into the fault tree The methodology is then demonstrated through the use of a pressure vessel cooling system.

4.0 Traditional Fault Tree Methodology

A coherent fault tree expresses the causality of a system failure mode in terms of AND and OR combinations of component failure modes, human errors, etc., as illustrated by the simple example shown in Figure 3. KTT performs fault tree analysis in two stages. The qualitative stage forms the Boolean equation which represents the causes of the top event, *TOP,* in terms of the basic events. It then manipulates it into disjunctive normal form (minimal sum of produces form):

$$TOP = C_1 + C_2 + \cdots . + C_{N_c} \tag{5}$$

from which the minimal cut sets, C_i , $i=1,\ldots, N_c$ (the necessary and sufficient combinations of component failures which cause the system failure) can be extracted [6]. The C_i terms are conjunctions of the basic event variables:

$$C_i = X_1.X_2.X_3 \ldots\ldots X_{n_{C_i}} \tag{6}$$

'+' represents the disjunction (OR) and '.' represents the conjunction (AND) in the logic equations.

From the minimal cut sets and the component's failure probability and frequency it is possible to calculate the Top Event probability, Q_{SYS} and failure intensity, w_{SYS}.

4.1 Top Event Probability

Assuming independence of the basic events the Top Event probability is given by:

$$Q_{SYS} = \sum_{i=1}^{N_c} P(C_i) - \sum_{i=2}^{N_c}\sum_{j=1}^{i-1} P(C_i \cap C_j) + \sum_{i=3}^{N_c}\sum_{j=2}^{i-1}\sum_{k=1}^{j-1} P(C_i \cap C_j \cap C_k) - \cdots \\ \cdots + (-1)^{N_C+1} P(C_1 \cap C_2 \cdots \cap C_{N_C}) \tag{7}$$

For even moderate sized problems, it is impractical to evaluate equation 7 and, approximations such as the Minimal Cut Set Upper Bound are employed:

$$Q_{SYS} \leq 1 - \prod_{i=1}^{N_c} (1 - P(C_i)) \tag{8}$$

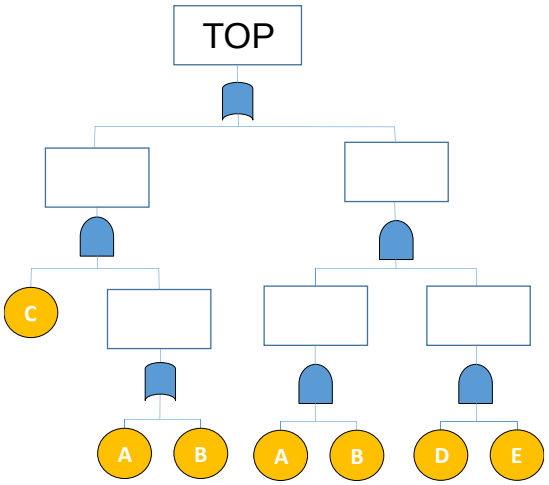


Figure 3 Simple Example Fault Tree

4.2 Top Event Failure Intensity

To calculate the top event failure intensity, a distinction can be made between initiator and enabler events [5]:

Initiating events: perturb system variables and place a demand on control / protection systems to respond.

Enabling events: are inactive control / protection systems which permit an initiating event to cause the top event.

The top event occurs when all component failures in a minimal cut set occur. For this to happen, all events in the minimal cut set will exist except one and then that last event occurs. The last event must be an initiator which puts a demand on the enablers to respond and since they have failed this does not occur, resulting in the top event. This introduces the concept of a *Critical System State* for an initiator.

A *critical system state***,** for initiating event i, is a state of the other components in the system such that the failure of component i causes the system to pass from the functioning to the failed state.

It can be shown that the probability of being in a critical system state for initiator i, the criticality function, G_i, (Birnbaum's measure of importance) [5] is given by:

$$G_i(\boldsymbol{q}) = \frac{\partial Q_{SYS}}{\partial q_i} \tag{9}$$

Since Q_{SYS} is linear in each q_i this can be expressed as:

$$G_i(\boldsymbol{q}) = \frac{\partial Q_{SYS}}{\partial q_i} = Q_{SYS}(1_i, \boldsymbol{q}) - Q_{SYS}(0_i, \boldsymbol{q}) \tag{10}$$

where:

$Q_{SYS}(1_i, \boldsymbol{q})$ is probability that the system fails with component i failed

$Q_{SYS}(0_i, \boldsymbol{q})$ is probability that the system fails with component i working

By calculating the criticality function, G_i, for each initiator gives the system failure intensity, w_{SYS}:

$$w_{SYS}(t) = \sum_{\substack{i \\ initiators}} G_i(\boldsymbol{q}).w_i\,(t) \tag{11}$$

Where, w_i is the failure intensity of initiator event *i*.

5.0 Complexities and Dependencies in Engineering Systems

Dependencies can occur between component failure events in many ways. It can be due to the way that the system is designed, for example employing standby dependency, it can be due to limitations on the maintenance resources where components are queued for repair, or it can be due to operational practices which make advantage of opportunistic maintenance.

Some commonly encountered sources of dependencies which occur across multiple industries are listed in table 1.

When dependencies exist in a system, conventional fault tree analysis is not an appropriate means to predict its failure characteristics.

Type	Description	Example
Standby	**Hot Standby (independent events)** Both primary and standby components are operational but only the primary component is providing a function to the system. On its failure the standby component takes over its role. **Warm Standby (dependent events)** The standby is not operational. It becomes operational when the primary system fails. The backup unit can fail in standby but with a lower rate than when operational. **Cold Standby (dependent events)** The standby is not operational. It becomes operational when the primary system fails. It cannot fail in standby.	The primary power source fails and the backup, a diesel generator, is started to provide an alternative power source.
Secondary failure	When one component fails it increases the load on a second component which then experiences an increased failure rate.	Two pumps both operational and sharing the load. Each pump has the capability to satisfy the full demand should the other pump fail.
Opportunistic Maintenance	A component fails which causes a system shutdown or requires specialist equipment for the repair. This opportunity is taken to do work on a second component which has not failed but is in a degraded state.	Components on a circuit board. When one fails the whole board is returned to the factory and all components are refurbished. The same applies to components in a sub-sea production module.

Common Cause / Common Influence	When one characteristic (eg materials, manufacturing, location, operation, installation, maintenance) causes the degraded performance or failure in several components.	Incorrect maintenance done on several identical sensors An impact breaks the circuit on cables routed in the same way to different redundant channels.
Queueing	Failed components all needing the same maintenance resource are queued. Then repaired in priority order.	Limited number of maintenance teams, equipment or spares.

Table 1 Some common sources of dependency in engineering systems

6.0 Enabling Methodologies

In this section, the characteristics of the Binary Decision Diagrams, Petri Nets and Markov models are summarised. It is these methods which provide the enabling techniques to expand the capabilities of fault tree analysis to form D^2T^2.

6.1 Binary Decision Diagrams

Binary Decision Diagrams (BDD) encode Shannon's form of the Boolean Equation. This disjoint form has considerable advantages when quantify the top event, system performance parameters and can be derived directly from the fault tree [9]. It requires the basic event variables to be placed in an order which has a major impact of the efficiency of the representation achieved [12,13]. A BDD equivalent to the logic function represented in the fault tree in Figure 3 is shown in Figure 4 for a basic event ordering: A < B < C < D < E.

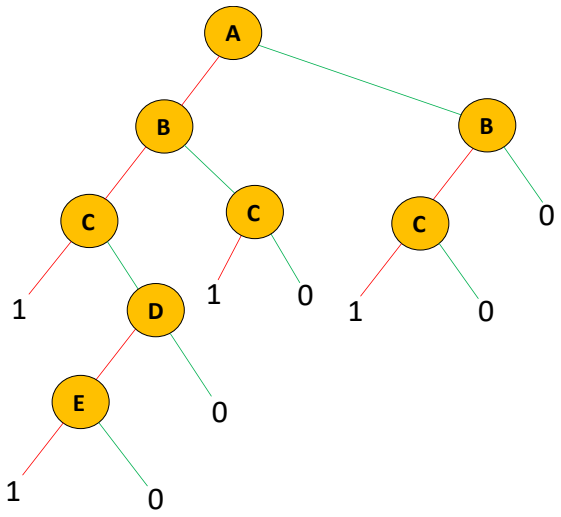


Figure 4 BDD structure for the Faut Tree in Figure 3

The root node appears at the top of the diagram, variable 'A' for the BDD in Figure 4. Each node in the BDD represents a Boolean variable for a basic event, in the fault tree. Should the variable be true (the component failure event occurs), the node is exited on the 1- branch (coloured red in the figure). If the variable is not true, then it exits the node of the 0-branch

(coloured green in the figure). Paths through the nodes in the BDD end at a terminal-1 or terminal-0 node. These represent the top event occurrence and non-occurrence respectively.

Tracing all paths through the BDD to a terminal-1 gives a set of disjoint combinations of events which produce system failure. For the BDD in Figure 4 there are 4 such paths:

$A.B.C \quad A.B.\bar{C}.D.E \quad A.\bar{B}.C$ and $\bar{A}.B.C$

Since each path through the BDD are mutually disjoint:

$$Q_{SYS} = \sum_{j=1}^{N_{path}} P(path_j) \tag{12}$$

Therefore:

$$Q_{SYS} = P(A.B.C + A.B.\bar{C}.D.E + A.\bar{B}.C + \bar{A}.B.C)$$

$$= P(A.B.C) + P(A.B.\bar{C}.D.E) + P(A.\bar{B}.C) + P(\bar{A}.B.C)$$

With the expression for the Top Event probability it is now possible to employ equations 10 and 11 to deliver the failure intensity [9, 10].

6.2 Stochastic Petri Nets

The Stochastic Petri net (SPN) method provides the flexibility in the analysis methodology. It has the ability to solve reliability problems featuring dependencies and complex maintenance processes. Times at which the system changes state can be represented by any distribution. The foundations of the approach were published in the thesis of Carl Adam Petri [17] in 1966. The method has developed significantly since then [18-20].

The Petri net is a bi-partite graphical modelling tool with nodes representing both places and transitions. An example Petri net is shown in Figure 5. A place (shown as a circle) represents a particular state of the system or component, a transition (shown as a square) represents an event, such as a failure or a repair, which causes the system to move from one state to another, thus, incorporating the dynamic behaviour. Tokens are located in the places to indicate the state of the system at any time. The arrows or edges link the places to the transition (for input places) and the transition to the places (for output places). Some edges have an associated number, or multiplicity. Where no multiplicity is indicated then the default value is one.

To model the dynamics of the system, there are rules which govern the transition:

i. First a transition has to be *enabled*. This occurs when the transitions input conditions are satisfied and the input places contain at least their multiplicity of tokens.

ii. The enabled transition then *'fires'* after a time period 't' following its enabling. For stochastic transitions the time is derived from a random sample taken from the appropriate distribution. Other transitions can be of fixed time length and in the case of deterministic

transitions an immediate firing occurs. On firing, the multiplicity of tokens are removed from each input place and a multiplicity of tokens are added to the output places.

A series of transition structures have also been developed, including the reset transition, the inspection transition and the place conditional dependent transition, specifically to account for features commonly encountered in system reliability studies [27, 28]. These transitions enable a more concise representation of the system and enable greater computational efficiency.

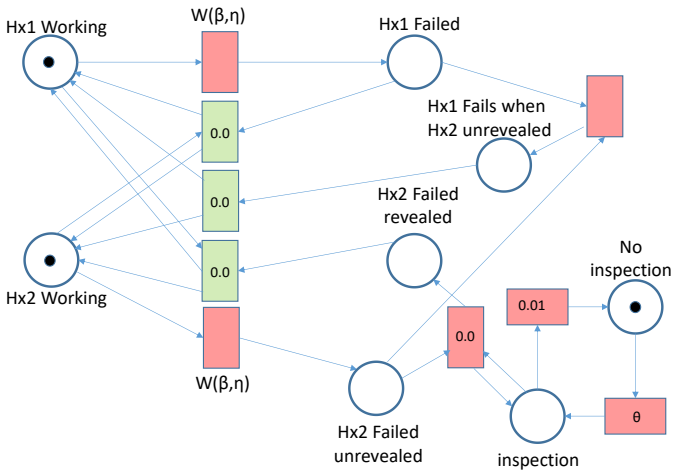


Figure 5 An Example Petri Net

The system performance is achieved through a Monte-Carlo simulation of the Petri net. During the simulations the time duration that the system spends in the different states is logged and this enables the likelihood of these events to be predicted. The number of times the system enters specified states is also logged which delivers the event frequencies.

The Petri net shown in Figure 5 features two components, Hx1 and Hx2, where both components are replaced as soon as either fails. The simulation of the net gives the likelihood that the system is in every possible state for these two components. It also produces the frequency with which each of these states occurs.

6.3 Markov Models

The Markov property requires the system failure and repair processes to be homogeneous and memoryless which gives rise to invariant transition rates and that the immediate future state of the system depends only on its current state. Continuous Time Markov Process models provide the ability to analyse systems with dependencies, where failure and repair states are constant. An example Markov model is illustrated in figure 6.

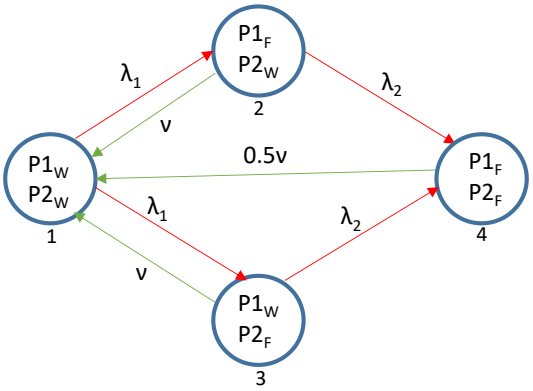


Figure 6 Simple Markov Model

The Markov model has two elements: states (nodes) and transitions (directed edges). The nodes represent the states of the system in terms of the status of the components. The edges represent the transitions which can occur between states and have an associated parameter which is the rate of transition. The model enables the state equations to be formed:

$$\dot{\boldsymbol{Q}}(t) = \boldsymbol{Q}(t)[A] \tag{13}$$

Where Q is the vector of state probabilities and A is the state transition matrix.

A solution of the model [3], commonly obtained by numerical methods, delivers the probability of being in each of the states and the rates of transition to the states. A Petri net could be used to form these models, employing transitions with exponential state residence times. The Markov method has been retained in the framework in the interests of efficiency since, for situations where it is applicable, it can offer faster solution times than simulation.

Markov does offer a more restrictive ability to represent maintenance processes than the SPN. Where maintenance strategies are governed by rules defining what to do when a certain state is achieved, SPNs build the model around a token residing this state. Markov models are unable to replicate this as it is only the probability of being in the state that is known, not the definite occurrence of this condition.

7.0 Dynamic and Dependent Tree Theory (D^2T^2)

Dynamic and Dependent Tree Theory (D^2T^2) extends the capabilities of Kinetic Tree Theory by exploiting the capabilities of the Binary Decision Diagrams, Petri Nets and Markov methods in order to solve Fault Trees which feature dependencies or sequences between the basic events and complexities in the degradation and maintenance processes. It retains the fault tree logic representation to define the system failure causality.

When the complexities are of the form of non-constant failure and repair rates then the Petri net method can be used to deliver the probability or frequency of failure. For components which experience complex asset management strategies, Petri net or Markov models are employed depending on the details of the strategy. Since these situations do not necessarily introduce dependencies between components, it would be possible to substitute the derived probabilities back into the Kinetic Tree Theory.

When parts of the system experience a form of dependency between the component failure events or sequences of events, then equations 7, 8 and 11 are no longer appropriate. To calculate the probability of system failure would require the dependencies between events within each minimal cut set to be calculated and also account for then the calculation of dependencies between the many combinations of minimal cut sets. The computational demands make this method intractable. The cornerstone of D^2T^2 relies on restructuring the fault tree logic into a disjoint form through the Binary Decisions Diagram. Since each path through the BDD is disjoint it removes the need to consider the dependencies between the paths, except in a very limited way for the failure frequency calculations. We then only need to consider the dependencies between the events in any path through the BDD.

Since BDDs, for large scale problems, can feature a large number of paths through the BDD, the size of the BDDs constructed must be minimised. This is achieved by:

- effective variable ordering.
- determining the smallest fault tree to convert to a BDD by an effective modularisation process which is described in the flowing sections.

7.1 D^2T^2 Algorithm

The overall algorithm is represented in Figure 7. The input data, shown on the left of the figure, is supplied in three files. The fault tree structure file, the component failure and repair model file and the dependency file. The first two of these files would be those required to perform a traditional fault tree analysis. For the D^2T^2 approach the component failure model file is extended to enable data specifying non-constant failure and repair processes to be defined. The dependency file specifies the types of dependency (or sequence) experienced, the components in this dependency group and the parameters governing the dependency processes. For some commonly occurring dependency classes this raw data will be used to generate the appropriate SPN or Markov model. For less common structures, and to facilitate generality, the SPN or Markov model can be explicitly entered in this file.

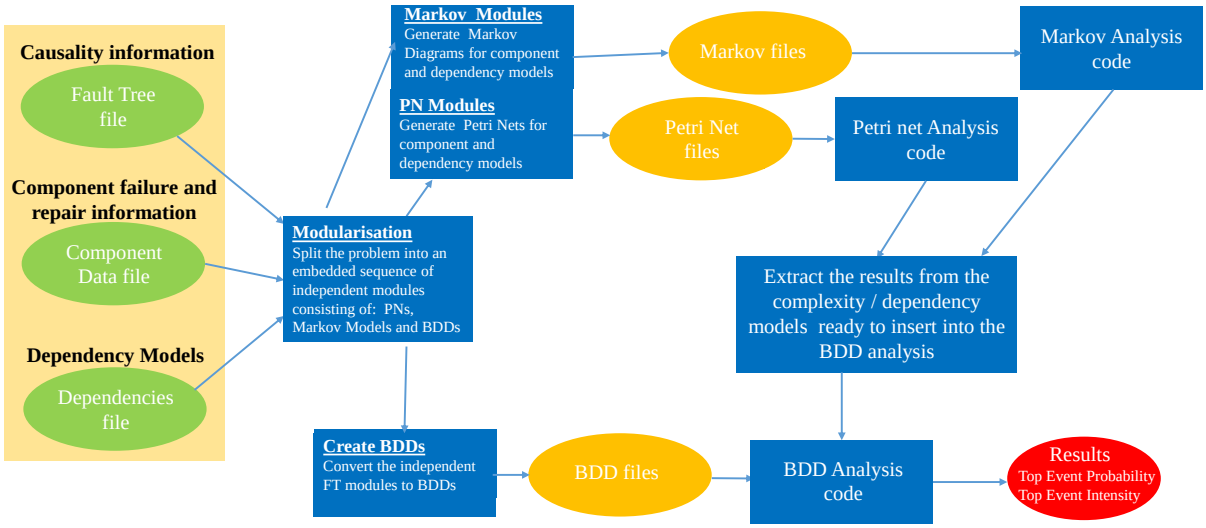


Figure 7 The D^2T^2 Solution Process

The algorithm executes D^2T^2 in the following steps to solve dependent and complex system models:

i. Identify the Initiators and Enablers.
ii. Create Dependency Groups:
    Identify all components which feature in dependency groups such as the group that is independent of all other groups and components.
iii. Quantify Traditional Component Failure Models:
    Quantify probabilities and intensities of components whose failure characteristics fit with the traditional component failure models (equations 1-4).
iv. Modularisation:
    Identify independent modules of the fault tree. Where dependencies exist between basic events they are contained within the same independent fault tree module.
v. Model Generation and Solution for the Complex and Dependent Groups:
    - Petri net model creation and analysis.
    - Markov model analysis.
vi. BDD Construction:
    Construct the BDD for the simplified fault tree structures
vii. Results Integration:
    The final top event probability and intensity predictions are calculated by integrating the results of the PN and Markov models into the BDD.

An in-house code in C++ has been written to execute the analysis of steps iii-vii. The key parts of the algorithm are the modularisation in stage iii and the results integration in stage vii. These are described in detail in the following sections. A case study is then used to demonstrate how each stage of the algorithm is executed.

7.2 Modularisation of the Fault Tree

Modularisation is performed to redefine the original fault tree structure to a series of small independent modules, each of which can be efficiently solved, and the results reassembled to produce the solution of the original problem. Two very effective techniques exist for fault tree modularisation where gate dependencies are created through repeated basic events. The first is an approach developed at Riso and used in their Faunet code [29, 30]. The second is the linear-time algorithm to identify independent gates in the fault tree structure developed by Dutuit and Rauzy [31]. Whilst both of these are effective in identifying small independent modules, they do not necessarily produce the smallest independent modules. Since they utilise differing approaches, applying the two sequentially can produce smaller modules than applying either on its own. Both algorithms require modifications to account for the dependencies between basic events permitted within D^2T^2. These modifications are explained below.

Factor Modularisation Method

The Faunet reduction method reduces the fault tree to its fundamental structure by defining a series of very simple, two element, factors or modules. This method has been re-formulated

to improve efficiency and enable dependent events to be accommodated. The factors are able to contain any number of elements to improve efficiency. Rules defining which types of events can be included within the same factor definition are also provided to ensure the factors defined are all mutually independent. The factor modularisation method executes the repeated application of three stages: contraction, factorisation and extraction.

Contraction:

Subsequent gates of the same type are contracted into a single gate. This makes the fault tree an alternating sequence of AND and OR gates.

Factorisation:

Extracts factors expressed as groups of events that always occur together as inputs to the same gate type. The factors can be any number of events if they satisfy the following:

- All events in the group are independent initiators.
- All events in the group are independent enablers.
- All events in the group feature a dependency and contain all events in the same dependency group.

Extraction:

Structures of the two forms illustrated in Figure 8, where a repeated event is an input to all gates of the same type on one level, are restructured as shown. This is performed to enable further simplification of the fault tree structure.

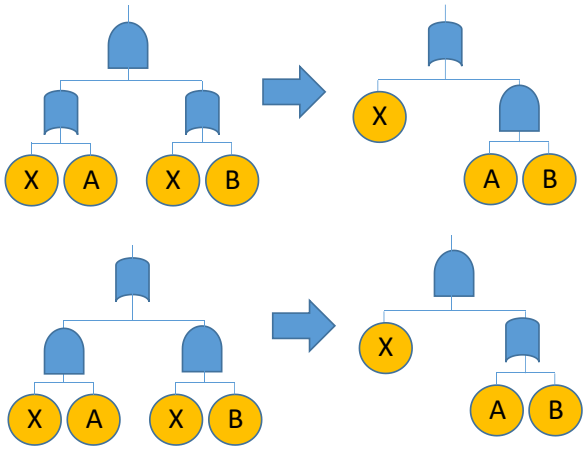


Figure 8 'Extracted' Fault Tree Structures

For the complex factors created the probability, Q_{Cfi}, and failure intensity, w_{Cfi}, can be calculated. For complex factors of OR and AND combinations these are given by:

For factors formed from combinations of independent events

OR combinations, $Cf_i = x_1 + x_2 + \cdots x_n$

$$Q_{Cfi} = 1 - \prod_{j=1}^{n} \left(1 - q_{x_j}\right) \tag{14}$$

If the factor contains only initiating events:

$$w_{Cfi} = \sum_{j=1}^{n} \left(w_j \prod_{\substack{k=1 \\ k \neq j}}^{n} \left(1 - q_{x_k}\right) \right) \tag{15}$$

AND combinations, $Cf_i = x_1 . x_2 . \ldots . x_n$

$$Q_{Cfi} = \prod_{j=1}^{n} q_{x_j} \tag{16}$$

$$w_{Cfi} = \sum_{\substack{j=1 \\ initiators}}^{n} (w_j \prod_{\substack{k=1 \\ k \neq j}}^{n} q_{x_k}) \tag{17}$$

For factors formed from combinations of events from a dependency group

The factor probabilities and intensities are extracted directly from the SPN or Markov dependency model used.

Linear-time Algorithm

Following the application of the factor modularisation process there will be a list of complex factors, created in the factorisation stages, and a core fault tree whose basic events include the factors, which cannot be reduced further. The linear-time algorithm may be able to identify independent gates within this core structure and so it is applied to this reduced fault tree. In its original form the modularisation algorithm was targeted at conventional fault trees whose events are independent. To identify independent sub-trees where some of the basic events are dependent can be accomplished by allocating all basic events in the same dependency group with the same label.

7.3 Integration of the Dynamic, Dependent or Complex Sub-model Results into the Core Fault Tree Analysis

As shown in the calculation procedure in Figure 7, the independent fault tree modules identified following the application of the factor and linear-time modularisation algorithms are converted to BDDs. Concise BDD structures are developed exploiting effective variable ordering schemes.

Models for each dynamic, dependent or complex aspect of the problem, representing non-exponential failure or repair times, complex maintenance strategies or specific dependencies are expressed in an appropriate form such as a Petri net or Markov model, which are solved as outlined in sections 6.2 and 6.3.

The results from the Petri net and Markov models are now integrated into the fault tree quantification. This is achieved by considering each path through the BDDs individually, and it is for this reason that the modularisation has been performed to keep the number of BDD paths as small as possible. On each path the components are grouped according to their dependency status. For example, consider the BDD in Figure 4 where there are two dependency groups: D1={b, c} and D2={d, e}.

There are four disjoint paths through the BDD to a terminal-1 as reproduced in column 2 of Table 2 where the event subscripts relate to the 1 and 0 branches from variables. The next three columns separate the events in each path into those which are independent and those which are members of the two dependency groups. The probability of the dependency group events in each path can be extracted from the dependency models.

j	$path_j$	$Ipath_j$	$Dpath_j^1$	$Dpath_j^2$
1	a_1, b_1, c_1	a_1	b_1, c_1	
2	a_1, b_1, c_0, d_1, e_1	a_1	b_1, c_0	d_1, e_1
3	a_1, b_0, c_1	a_1	b_0, c_1	
4	a_0, b_1, c_1	a_0	b_1, c_1	

Table 2 Component Dependency Groupings

The final stage of the methodology utilises the path calculations to delivers the system, original top event, failure probability and failure intensity.

7.3.1 Top Event Probability

The top event probability is calculated by adapting equation 12, grouping components which are independent or in the same dependency groups along each path to give:

$$Q_{SYS} = \sum_{j=1}^{npath} \left[P\left(Ipath_j\right) \prod\nolimits_{k=1}^{ndep} P(Dpath_j^k) \right] \tag{18}$$

where;

npath – number of paths through the BDD to a terminal-1.

ndep – number of dependency groups.

$Ipath_j$ - is the independent component states in the jth path.

$Dpath_j^k$ – is the component events in dependency group k on path j.

The probability values for dependency terms are obtained from the appropriate SPN or Markov model. Where the components in a dependency group on a path are not the complete set of components on the group then the probability of the subgroup can be calculated from:

$$P(x_1, \ldots\ldots, x_m) = \sum_{x_{m+1=0}}^{1} \ldots\ldots\ldots \sum_{x_{n=0}}^{1} P(x_1, \ldots\ldots, x_n) \tag{19}$$

where:

the dependency group membership is: (x_1,….., x_m, x_{m+1}, ….. , x_n) and the sub-group of components in the path is: (x_1,….., x_m)

7.3.2 Top Event Failure Intensity

Evaluating the system failure intensity, like the system failure probability, uses paths through the BDD structure and the grouping of components into dependency groups to calculate equation 11. This requires the prediction of the criticality function for each of the initiators by evaluating the two probability terms in equation 10.

Consider the two situations where it is required to calculate the criticality function for initiator x_i where the variable is a member of a dependency group, d, or a member of the class of independent variables.

x_i is a member of dependency group d

Consider in the general BDD structure shown in figure 9 where there are paths through the x_i node to a terminal-1 and paths which do not encounter this node.

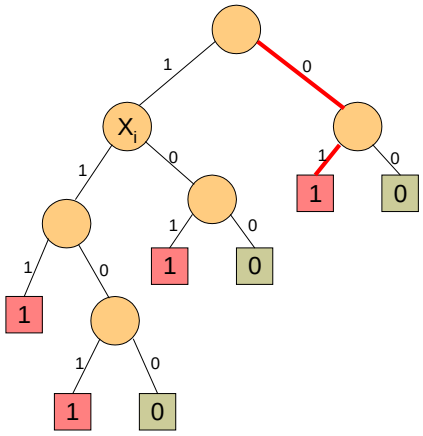


Figure 9 General BDD structure

For the first term in equation 10, the probability of the top event when x_i is failed (x_i=1), this sets the paths that pass through the variable x_i on the 1-branch to do so with certainty and the

paths which pass x_i on the 0-branch do not need to be considered. So the summation of the paths to a terminal-1 will include all paths which pass through variable x_i on the 1-branch removing the variable x_i. It will also include all the paths to a terminal-1 which do not pass through the variable x_i. In these paths it is however possible that other variables in the path could be in the same dependency group as x_i and so must be conditional on x_i=1 as shown in equation 20.

$$Q_{SYS}\left(1_i, \underline{q}\right) = \sum_{x_{i_1} \in path_j} P\left(path_j - x_{i_1}\right) + \sum_{x_i \notin path_j} P(path_j | x_i = 1) \tag{20}$$

Expressing the path probabilities accounting for the independent/dependency groups gives:

$$Q_{SYS}\left(1_i, \underline{q}\right) = \sum_{x_{i_1} \in path_j} \left[P\left(Ipath_j\right). \prod_{\substack{k=1 \\ k \neq d}}^{ndep} [P(Dpath_j^k)] \,.\, P(Dpath_j^d - x_{i_1} | x_i = 1) \right] + \sum_{x_i \notin path_j} \left[P\left(Ipath_j\right). \prod_{\substack{k=1 \\ k \neq d}}^{ndep} [P(Dpath_j^k)] \,.\, P(Dpath_j^d \,| x_i = 1) \right] \tag{21}$$

Considering the second term in equation where x_i=0 and the paths through the x_i node pass on the 0-branch gives:

$$Q_{SYS}\left(0_i, \underline{q}\right) = \sum_{x_{i_0} \in path_j} P\left(path_j - x_{i_0}\right) + \sum_{x_i \notin path_j} P(path_j | x_i = 0) \tag{22}$$

and

$$Q_{SYS}\left(0_i, \underline{q}\right) = \sum_{x_{i_0} \in path_j} \left[P\left(Ipath_j\right). \prod_{\substack{k=1 \\ k \neq d}}^{ndep} [P(Dpath_j^k)] \,.\, P(Dpath_j^d - x_{i_0} | x_i = 0) \right] + \sum_{x_i \notin path_j} \left[P\left(Ipath_j\right). \prod_{\substack{k=1 \\ k \neq d}}^{ndep} [P(Dpath_j^k)] \,.\, P(Dpath_j^d \,| x_i = 0) \right] \tag{23}$$

Subtracting the two terms in equations 21 and 23, for each initiator, gives the criticality function which can be substituted into equation 11.

x_i is not a member of a dependency group

When x_i is one of the independent variables the second terms in equations 20 and 22 are no longer dependent upon x_i giving:

$$G_i(\boldsymbol{q}) = \sum_{x_{i_1} \in path_j} P\left(path_j - x_{i_1}\right) + \sum_{x_i \notin path_j} P(path_j)$$

(24)

$$- \sum_{x_{i_0} \in path_j} P\left(path_j - x_{i_0}\right) - \sum_{x_i \notin path_j} P(path_j)$$

The two terms which provide the probability of passing to a terminal-1 on paths which do not contain x_i now cancel. Expressing the equation in terms of the independent and dependency group probabilities gives:

$$G_i(\boldsymbol{q}) = \sum_{x_{i_1} \in path_j} \left[P\left(Ipath_j - x_{i_1}\right). \prod_{k=1}^{ndep} [P(Dpath_j^k)] \right]$$

$$- \sum_{x_{i_0} \in path_j} \left[P\left(Ipath_j - x_{i_0}\right). \prod_{k=1}^{ndep} [P(Dpath_j^k)] \right] \quad (25)$$

8.0 Pressure Vessel Cooling System Case Study

The case study system is illustrated in Figure 10. The system is a pressure vessel used for an exothermic chemical reaction which requires cooling. Cooling is performed by the primary cooling system which features a water supply from tank, T1, which is driven to the heat exchanger (Hx1) by two pumps (P1 and P2). The pumps are powered by a common supply, PoW.

Failure of the primary cooling system causes an increase in vessel temperature which is detected by the thermocouples, S1 and S2. In the event that either thermocouple registers an increasing vessel temperature, the computer will de-energise the relays R1 and R2 and activate two alternative cooling systems. The first of these auxiliary cooling systems is similar to the primary system with water supply, T2, heat exchanger, Hx2 and a single pump, P3. When relay R2 is de-energised its contacts close, powering pump P3 and opening the motorised valve V1. The second cooling action comes from fan, F, powered by a motor, M. Activation of this system occurs when relay R1 contacts close on the relay de-energisation. Fan, Motor, P3 and V1 also take their power from supply PoW.

In the event that the primary cooling system fails, both auxiliary systems need to activate and operate, without failure, for a further 30 days whilst the process is shut-down.

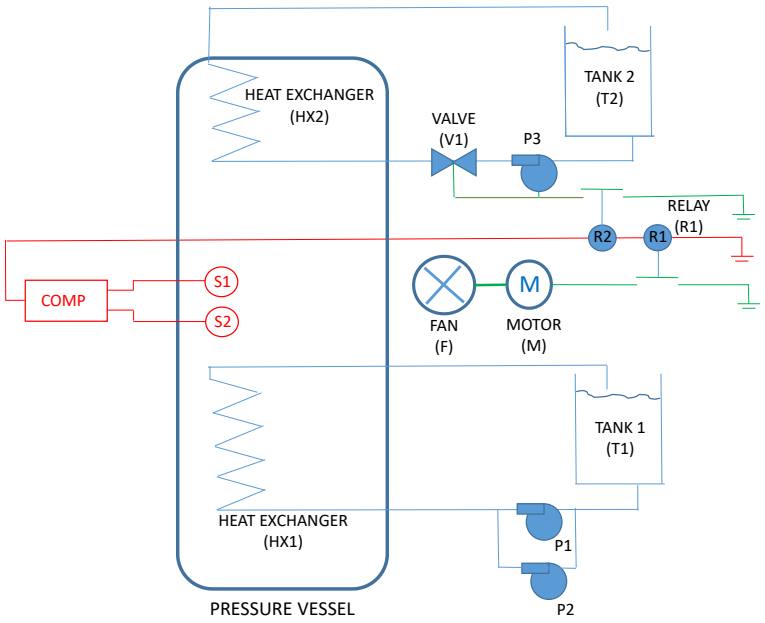


Figure 10 Pressure Vessel Cooling System Case Study

The fault tree for the top event, 'Pressure Vessel cooling fails' is developed in Figure 11, basic event codes are defined in Table 3, which also contains the component failure, repair and inspection data. The objective of the analysis is to calculate the probability and frequency of the top event.

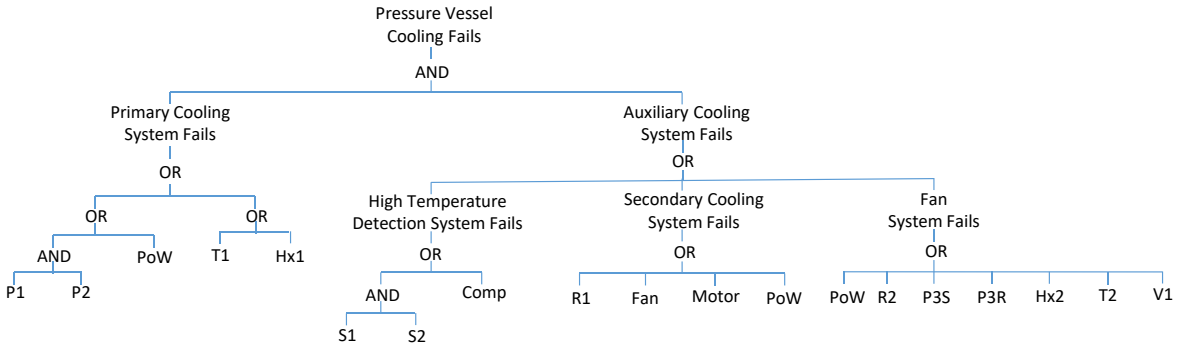


Figure 11 Pressure Vessel Cooling System Fails Fault Tree

Event Code	Description	I / E	D-Group	Failure rate (/hour)	Mean time to repair (hours)	Inspect interval (hours)	q	w
P1, P2	Pumps fail when running	I	D1	Failure rate $\lambda_1 = 2 \times 10^{-5}$ /h under normal load $\lambda_2 = 5 \times 10^{-3}$/h under full load Repair rate ν= 0.041667 (MTTF = 24hrs)				
T1	Water Supply failure	I		$1x10^{-5}$	24		$2.4x10^{-5}$	$9.99976 x10^{-6}$

Hx1	Heat Exchanger fails	I	D2	Failure time = W(β=2.5, η=30,000h) The system is shut down when the repair is undertaken				
PoW	Power supply failure	I		$1x10^{-4}$	10		$1x10^{-3}$	$9.99 x10^{-5}$
S1, S2	Sensor fails to detect a high temperature	E		$5x10^{-4}$	5	730	0.185	
Comp	Computer fails to process sensor signals	E		$5x10^{-5}$	5	2190	0.055	
R1 / R2	Relay contacts fail to close	E		$1x10^{-5}$	24	2190	0.0112	
Fan	Fan fails	E		$2x10^{-6}$	8	2190	$2.206 x10^{-3}$	
Motor	Fan motor fails	E	C1	Failure time = W(β=1.5, η=12,000h) Repair time = LogN(μ=24hrs, σ=4.8h)				
P3S	Pump fails to activate	E	D3				0.05	
P3R	Pump fails when running	E	D3	$1x10^{-4}$				
T2	Water Supply failure	E		$1x10^{-5}$	24	2190	0.0112	
Hx2	Heat Exchanger fails	E	D2	Failure time = W(β=2.5, η=30,000h) The system is shut down when the repair is undertaken				
V1	Valve fails to open	E		$5x10^{-5}$	30	2190	0.05625	

Table 3 Basic Event Definitions

The pressure vessel cooling system will be used to demonstrate the application of D^2T^2 algorithm. There are four elements of the pressure vessel cooling system that introduce complexities or dependencies into the analysis which are analysed using Petri net, Markov and alternative approaches:

- The motor for the fan experiences failures times governed by a Weibull distribution and repair times which are lognormal. This introduces a complexity in calculating the likelihood that the motor will not last for the 30 days required. A Petri net will be used to assess the failure characteristics of the motor.
- The pumps, P1 and P2, in the primary cooling system, contain a dependency. In normal operation both pumps operate. When one pump fails the other takes the full load with a corresponding increase in its failure rate. Since the pumps experience constant failure and repair rates a Markov model can be used.

- A maintenance dependency exists between the two heat exchangers, Hx1 and Hx2. They operate in similar environments and when one reaches a point where replacement is needed, the other will also be close to that condition. Since they have to shut down the process and employ specialist equipment to perform the maintenance, the opportunity is taken to replace both at the same time. A Petri net will be used to model this more complex maintenance process.
- There is another complexity for pump, P3, where the two events relating to this component appear in Table 2, the pump failing to start, P3S, and failing during the 30 day operating period, P3R, which both combine to give the probability of P3 not working for the required duration.

9.0 Case Study Analysis

The D^2T^2 algorithm, presented in section 7.1, is now applied to the case study system in order to predict its failure probability and intensity. The qualitative analysis to deliver the minimal cut sets is not changed in comparison to the traditional approach and, as such, is not considered in this paper. The execution of each step in the algorithm is described in detail.

Step i: Identify the Initiators and Enablers

All events which cause the primary cooling system failure put a demand on control and protection systems to prevent a vessel cooling failure and, as such, are initiating events. Failure of the safety features: the detection system or the two auxiliary cooling systems are enablers. These categorisations are included in Table 3, column 3.

Step ii: Create the Dependency Groups

This step explicitly identifies the complexities and dependencies, along with the components whose failures appear in each. These will be the parts of the analysis which deviate from the traditional KTT quantification. From the description of the problem given above it can be seen that the motor for the fan, Motor, does not have constant failure and repair rates and its probability and failure intensity will be obtained by a Petri net model. This is complexity group C1 and contains only the one component since its failure parameters remain independent from the state of the other components.

C1={Motor}

Pumps P1 and P2 exhibit dependent failures and so are included together in the first dependency group:

D1={P1, P2}

Heat Exchangers Hx1 and Hx2 experience an opportunistic maintenance dependency and are allocated to dependency group 2.

D2={Hx1, Hx2}

The events in the fault tree which relate to the third pump, P3 representing its failure to start and failure once running, P3S and P3R are also dependent and the failure parameters for this component will form another dependency group.

D3={P3S, P3R}

Step iii: Quantify Traditional Component Failure Models

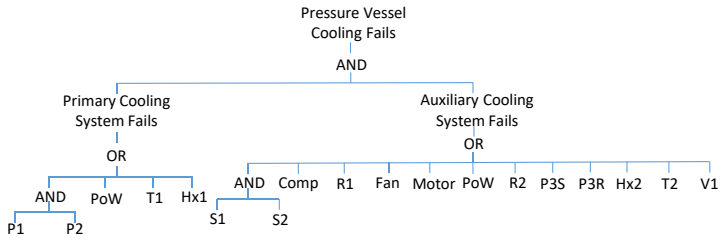
All basic events which do not appear in any complexity or dependency group are independent events with constant failure and repair rates whose probability of failure and failure intensity can be calculated as per conventional approaches using equations 1-4. The failure probability, q, of the initiators and enablers, along with failure intensity, w, of the initiators are thus calculated and shown in the last two columns of Table 3.

Step iv: Modularisation

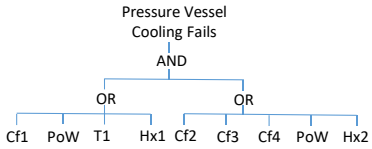
This step employs the two stage process described in section 7.2. First applying the three processes of the Factor approach, and then the linear time algorithm of Dutuit and Rauzy.

Contraction 1

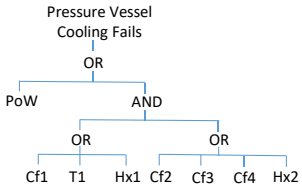
Applying the first contraction phase to the fault tree in Figure 11 produces the fault tree shown in Figure 12A.



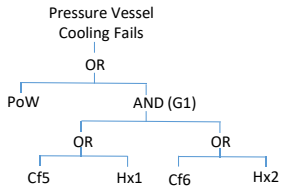
A: after Contraction 1



B: after Factorisation 1



C: after Extraction 1



D: after Contraction 2

Figure 12 Applying the Factor Modularisation Reduction

Factorisation 1

Creates the following factors:

$Cf_1 = P1.P2$ (dependency group D1 – initiators)

$Cf_2 = S1.S2$ (independent enablers)

$Cf_3 = Comp + R1 + Fan + Motor + R2 + T2 + V1$ (independent enablers)

$Cf_4 = P3S + P3R$ (dependency group D3 – enablers)

Replacing the events by their complex factors in the fault tree in Figure 12A delivers the fault tree in Figure 12 B.

Extraction 1

A restructuring of the fault tree logic is enabled by the first application of the extraction process around the event PoW. The resulting fault tree is illustrated in Figure 12C.

Contraction 2

No changes to the fault tree occur as it is already an alternating sequence of AND and OR gates.

Factorisation 2

Although Cf_1 and Cf_4 relate to dependency groups they are independent all other events in the fault tree and can therefore be used to form new factors.

$Cf_5 = Cf_1 + T1$ (initiator)

$Cf_6 = Cf_2 + Cf_3 + Cf_4$ (enabler)

Since Hx1 and Hx2 are in a dependency group together they cannot be defined as a factor with any other events and so no additional factors are possible and the final Factor modularisation simplified fault tree is given in Figure 12D.

Applying the linear time algorithm of Dutuit and Rauzy, modified to account for dependent events, the Top gate and the AND gate (labelled G1) will be identified as modules which can be analysed separately. This effectively defines another factor:

$Cf_7 = PoW + G1$

Step v: Model Generation and Solution for the Complex and Dependent Groups

Complexity Group C1={Motor}

This complexity module creates a simple PN structure, as shown in Figure 13 where the motor failure time distribution is W(β=1.5, η=12,000h) and the repair time distribution is LogN(μ=24h, σ=4.8h), taken from Table 3. Simulating gives the probability of the enabling event, q_{Motor}, failing to operate for 30 hours as 0.005839.

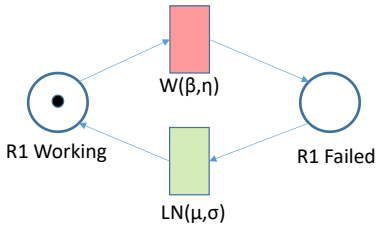


Figure 13 Simple Petri Net Model the Motor Complexity

Dependency Group, D1={P1, P2}

The dependency in module D1 occurs because a failure of one pump allocates the full work-load to the second. The constant rate with which pumps fail under normal operation is $\lambda_1 = 2 \times 10^{-5}$ /hour. When one pump fails and the full load is taken by the other pump its failure rate rises to $\lambda_2 = 5 \times 10^{-3}$ /hour. Repair is undertaken, on average, in 24 hours, giving a repair rate of $\nu = 0.041667$ /hour. The Markov model for the failure/repair process for the two pumps is shown in Figure 6. An analysis of this model, assuming steady state conditions, gives the results shown in Table 4.

State Number	State	State Probability	Intensity Expression	State Intensity (per hour)
1	$P1_W P2_W$	0.99743518	$w1 = (Q2 + Q3).\nu + Q4.(0.5)\nu$	7.12456×10^{-5}
2	$P1_F P2_W$	0.00042747	$w2 = Q1.\lambda_1$	1.99487×10^{-5}
3	$P1_W P2_F$	0.00042747	$w3 = Q1.\lambda_1$	1.99487×10^{-5}
4	$P1_F P2_F$	0.00170988	$w4 = (Q2 + Q3).\lambda_2$	4.2747×10^{-6}

Table 4 State Probability and intensity results for dependency group D1

Additional results needed from this model in the later calculations are:

$$P1_F | P2_F = \frac{P1_F \cap P2_F}{P2_F} = \frac{0.00170988}{0.00170998 + 0.00042747} = 0.8 \quad (26)$$

$$w_{P1_F} = Q_1.\lambda_1 + Q_3.\lambda_2 = 2.208605 \times 10^{-5} \ /hour \quad (27)$$

Dependency Group, D2={Hx1, Hx2}

Module D2 contains the basic events for the failure of the identical heat exchangers, Hx1 and Hx2. Each heat exchanger has failure times which follow a Weibull distribution with β=2.5 and characteristic life, η, 35,000 hours. When one of the heat exchangers reaches the point that it needs replacing, the system is shut down and both are replaced at the same time. Since Hx2 is an enabler, its failure is discovered through an inspection process. Failures of initiator, Hx1, are revealed. The PN, generated to represent this situation, is illustrated in Figure 5. Solving the model using Monte Carlo simulation and tracking the critical states gives the following results.

State Probabilities:

$P(Hx1_W, Hx2_W)$=0.98646987828725829
$P(Hx1_W, Hx2_F)$=0.0135301
$P(Hx1_F, Hx2_F)$=0.0
$P(Hx1_F)$=0.0
$P(Hx2_F | Hx1_F)$=0.0
$P(Hx2_F | Hx1_W)$= 0.0135301 (28)

State Failure Intensities

$w(Hx1_F, Hx2\_unrevealed)$=3.1709792 x 10^{-07} /hour
$w(Hx1_F, Hx2_W)$=1.8161063 x 10^{-05} /hour
$w(Hx1_F)$=1.8478161 x 10^{-05} /hour (29)

Dependency Module, D4={P3S, P3R}

Dependency module D4 is different from the others in that the two failure modes it contains relate to the same component, pump P3, which is an enabler and so a prediction of its failure probability, q_{P3}, is given by:

$$q_{P3} = q_{P3S} + (1.0 - q_{P3S})\lambda_{P3R}.t_{period} \qquad (30)$$

$$= 0.05 + 0.95 \times 10^{-4} \times 30 = 0.05285$$

Step vi: BDD Construction

The sub-tree for G1 in Figure 12D has to be analysed separately and account for the dependency between events Hx1 and Hx2. For this, the fault tree is converted to a BDD. The ordering of the variables chosen is: Cf5<Hx1<Cf6<Hx2 which gives the BDD shown in Figure 14.

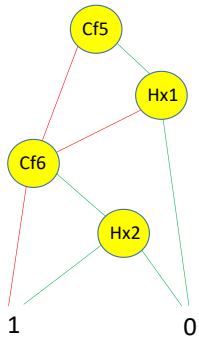


Figure 14 BDD of the Fault Tree below Gate G1

Step vii: Results Integration

Top Event Probability

The final step in the algorithm is to integrate the results from each part of the BDD, Petri net and Markov model quantification. First the calculations are performed for the independent complex factors extracted in the Factor Modularisation algorithm. These results are shown in Table 5. The probabilities of Cf1 and Cf4 obtained directly from dependency models D1 and D3 respectively. Cf2 is calculated as an AND combination of basic event, S1 and S2, whose likelihoods were determined in step iii. Cf3, Cf5 and Cf6 are all OR combinations whose probability is quantified using equation 14. In the case of Cf3 the probability of the motor failing comes from the analysis of complexity model C1.

Event Code	Description	I / E	D-Group	q
Cf1	P1. P2	I	D1	0.00170988
Cf2	S1. S2	E		0.00034225

Cf3	Comp + R1 + Fan + Motor + R2 + T2 + V1	E		0.6035094
Cf4	P3S + P3R	E		0.05285
Cf5	Cf_1 + T1	E		0.0017338
Cf6	Cf_2 + Cf_3 + Cf_4	E		0.6246519
G1	BDD	I		0.001091749
Cf7	PoW + G1		D2	0.0020906577

Table 5 Calculations to determine the Top Event probability

The next part of the calculation process is the evaluation of the probability of Gate G1, whose logic is represented by the BDD shown in Figure 14. There are 4 paths through this BDD to a terminal-1 and two of the events, Hx1 and Hx2, belong to dependency group D3. The paths and the classification of the events featuring in each path are shown in Table 6.

Path j	**$path_j$ elements**	**$Ipath_j$**	**$Dpath_j^1$**
1	$Cf5_1$, $Cf6_1$	$Cf5_1$, $Cf6_1$	
2	$Cf5_1$, $Cf6_0$, $Hx2_1$	$Cf5_1$, $Cf6_0$	$Hx2_1$
3	$Cf5_0$, $Hx1_1$, $Cf6_1$	$Cf5_0$,$Cf6_1$	$Hx1_1$
4	$Cf5_0$, $Hx1_1$, $Cf6_0$, $Hx2_1$	$Cf5_0$, $Cf6_0$	$Hx1_1$, $Hx2_1$

Table 6 Event Classifications for the paths through the BDD for G1

The probability of G1 is then obtained by evaluating equation 18. The probability expressions for each of the 4 paths in this calculation are:

$$Q_{path1} = P(Cf5_1).P(Cf6_1) = 0.0010830 \quad (31)$$

$$Q_{path2} = P(Cf5_1).(1 - P(Cf6_1)).P(Hx2_1) = 8.8052957x\,10^{-6} \quad (32)$$

$$Q_{path3} = (1 - P(Cf5_1)).P(Cf6_1).P(Hx1_1) = 0.0 \quad (33)$$

$$Q_{path4} = (1 - P(Cf5_1)).(1 - P(Cf6_1)).P(Hx1_1, Hx2_1) = 0.0 \quad (34)$$

The probabilities P(Hx2), P(Hx1) and P(Hx1, Hx2) are obtained from the Markov model evaluation of Dependency group D2. All others can be obtained from Table 5. Summing these 4 terms gives a probability for gate G1 = 0.00109175.

Finally, the top event failure probability is then derived by putting the result for G1 into Cf7, giving, $Q_{SYS} = 0.0020906577$.

Top Event Failure Intensity

To calculate the system failure intensity requires the evaluation of equation 11, where the summation is made over the five initiating events, T1, Hx1, P1, P2 and PoW. For all initiating events other than Hx1, calculating their criticality function, G_i, is simply a matter of performing the two top event calculations shown in equation 10, once assuming the component is failed and the second with the component working. Hx1 is more complex due to its residence in dependency group D2. This requires the processing of equations 22 and 23 which apply to the BDD representing G1. The BDD paths classification for the latter are summarised in Table 6.

Considering path 1, its contribution to the criticality function is 0 since all events in the path are independent of initiator Hx1.

The contributions for Paths 2-4 are:

$$G_{Path\_2} = P(Cf5_1).(1 - P(Cf6_1)).[P(Hx2_1|Hx1_1) \text{-} P(Hx2_1|Hx1_0)] \quad (35)$$

$$G_{Path\_3} = (1 - P(Cf5_1)).P(Cf6_1) \quad (36)$$

$$G_{path\_4} = (1 - P(Cf5_1)).(1 - P(Cf6_1)).P(Hx2_1|Hx1_1) \quad (37)$$

The terms in red are obtained from the D2 dependency Petri net model. The results entered into the equation 11 are shown in Table 7.

Variable	Q(var=F)	Q(var=W)	G_i(var)	$G_{i\ (var)}$ w
Hx1				$1.152147238 \times 10^{-5}$
T1	0.6300421	0.0020756	0.6279665	6.2795143×10^{-6}
P1	0.5042367	0.1268205	0.3774162	8.3356331×10^{-6}
P2	0.5042367	0.1268205	0.3774162	8.3356331×10^{-6}
PoW	1.0	0.0010918	0.9989082	9.979093×10^{-5}

Table 7 Failure intensity calculations for the initiating events.

This gives a system failure intensity of:

$$w_{SYS}(t) = 1.342632 \times 10^{-4} / hour \quad (38)$$

10. Discussion

10.1 D^2T^2 Methodology

The foundation of the D^2T^2 method is the BDD. When analysing a conventional fault tree with independent basic events, the method employed to evaluate the BDD would progress bottom-up evaluating each node in the BDD in terms of the probabilities of the events that are connected on its 1-branch and 0-branch. Although the BDD structure has not changed in the methodology presented, to account for the dependencies requires the BDD evaluation to progress one path at a time. This will add to the computational effort and is why it is essential to minimise the size of the BDD and the number of path calculations performed.

In basing the method on the BDD it exploits all the advantages that the BDD offers, it also suffers from the limitations. There are times when solving the fault tree using BDDs that a good ordering of the variables cannot be established. This can add to another source of computational inefficiency and in extreme cases it may not be possible to formulate the BDD. In this circumstance an approximation provides the means for analysis. The fault tree minimal cut sets are obtained, truncated, to yield those providing the main contributions to the top event. Dependent basic events are not permitted to be culled in this process. This reduced set of failure combinations are then used to construct an approximate BDD to be utilised in the methodology.

The methods used to calculate the complexities and dependencies, SPN and Markov, can be expanded and it is easy to incorporate alternative techniques such as Bayesian Networks [32] and Dynamic Uncertainty Causality Graphs [33] into the framework.

As is to be expected, in relaxing the restrictions, more computational resource will be required. Research will now focus on improving the efficiency of the algorithm, just as occurred with KTT. Future advances in computer technology will also add to the speed of the calculations.

10.2 Uncertainty and Transparency

As discussed in Apostalakis [34], uncertainty and errors occur in system assessment methods in several ways. One is the way that the failure model represents the engineering. Uncertainty can come through lack of knowledge of the underlying physics or the inability to represent the failure event combinations in the logic representation, in this case the fault tree. An example of this is provided when considering the causes of a derailment for a train. A combination of train speed and poor track geometry can lead to this hazard. Each of these quantities are continuous and the physics which relates the train speed necessary for derailment to occur, given the vertical track geometry condition (as measured by the standard deviation of the tracks vertical alignment over a defined length such as 35m or 70m), is not fully understood. Expressing these continuous variables in the discrete form of the fault tree basic events is also a limitation in representing this failure combination, and a pessimistic discretisation of these events are necessary.

A common discussion point is the existence of data of sufficient quality or quantity to populate: the failure time and repair time distributions associated with each component failure, the likelihood of human errors or of software failures in a particular failure mode. This issue is even more evident when attempting to fully understand dependencies that exist between elements of a system.

The method chosen for the analysis is also a factor to consider. The traditional fault tree approach requires approximations to be made in the calculations which combine the minimal cut set probabilities and intensities to produce the system performance. It can also be necessary to apply cut-offs to restrict the minimal cut sets considered to those which provide the most significant contributions [7]. This has been shown to have the potential for significant errors and is a major justification for the employment of the Binary Decision Diagram method.

The method presented in this paper tries to address some of these areas, specifically:

- component failure and repair times represented by any probability distribution.
- incorporate dependent events transparently into the assessment and force this consideration from the outset.
- allow the maintenance to be considered in a realistic representation of the processes executed for real systems
- permit event sequences to be considered.

The method aligns to the objectives of risk-informed decision making [34], ensuring that these features are considered within the initial model and that the assumption of independence does not inject optimism into the performance produced [33].

The retention of the fault tree to represent the causes of system failure provides an important contribution to the insight in the causes of system failure. The sub-models which calculate the performance of the dependencies and complexities contained within the model also offer insights into the realistic performance characteristics of that particular sub-system. This insight can be used when considering if some of the imposed safety requirements need to be enhanced due to their poor predicted performance, or, to ensure that resources are not wasted. The D^2T^2 methodology is intended to support rational decision making and effective peer review.

11.0 Summary and Conclusions

- The Dynamic and Dependent Tree Theory, D^2T^2, has been presented in this paper. It enables the evaluation of fault trees which are not limited by the restrictions which apply to conventional fault trees solved by Kinetic Tree Theory.
- The algorithm achieves the new capabilities by the utilisation of BDDs, Petri Nets and Markov Models to change the internal calculation processes of the fault tree analysis and retain the familiar and popular fault tree structure to represent the system failure causality.
- The Petri net and Markov models dedicated to solve the complexity and dependency models are minimal in size. This improves the efficiency of the method since large models can be expensive in computational effort to solve.
- Modularisation of the fault tree reduces the size of the BDD utilised in the system evaluation calculation. This is necessary to reduce the number of, and size of, the paths through the BDD in order to ensure the algorithm is as efficient as possible.

Acknowledgement

This work was supported by the Lloyd's Register Foundation, a charitable foundation in the U.K. helping to protect life and property by supporting engineering-related education, public engagement, and the application of research.

References

1. Watson H.A., Launch Control Safety Study, Bell Telephone Laboratories, Murray Hill, N.J. USA, 1961.
2. Vesely W.E., (1970), A Time Dependent Methodology for Fault Tree Evaluation, Nuclear Engineering and Design, Vol 13, pp337-360.
3. Andrews, J.D. and Moss, T.R., (2002), Reliability and Risk Assessment (2nd edition), Professional Engineering Publishing, 540 pp, ISBN 1-86058-290-7.
4. Zhang Q. and Qizhi, M., Element Importance and System failure frequency of a 2-State System, IEEE Transactions on Reliability, Vol R-34, No 4, 1985.
5. Dunglinson, C. and Lambert, L., Interval Reliability for Initiating and Enabling Events, IEEE Transaction on Reliability, 32(2), 1983, 150-163.
6. Rauzy, A., (2003), Toward an Efficient Implementation of the MOCUS Algorithm, IEEE Transactions of Reliability, 52(2), pp175-180.
7. Čepin M. Analysis of truncation limit in probabilistic safety assessment. Reliability Engineering and System Safety 2005; 87(3):395–403.
8. Rauzy A., (1993), New algorithms for Fault Trees Analysis. Reliability Engineering and System Safety, 59(5), pp203–211.
9. Sinnamon, R.M. and Andrews, J.D., (1996), Quantitative Fault Tree Analysis Using Binary Decision Diagrams, European Journal of Automation, 30(8), pp 1051-1071, ISSN 0296-1598.
10. Sinnamon, R.M. and Andrews, J.D., (1997), Improved Accuracy in Quantitative Fault Tree Analysis, Quality and Reliability Engineering International, 13, pp 285-292.
11. Sinnamon, R.M. and Andrews, J.D., (1997), Improved Efficiency in Qualitative Fault Tree Analysis, Quality and Reliability Engineering International, 13, pp 293-298.
12. Bouissou, M., Bruyere, F., Rauzy, A. BDD Based Fault Tree Processing: A Comparison of Variable Ordering Heuristics. In Proceedings of European Safety and Reliability Association Conference, ESREL'97, Lisbon, Portugal, 17–20 June 1997, Vol. 3, pp. 2045–2052.
13. Bartlett, L. M., Andrews, J. D. Choosing a Heuristic for the Fault Tree to Binary Decision Diagram Conversion, using Neural Networks. IEEE Trans. Reliability, 2002, 51(3), 344–349.
14. Mo,Y., Xing, L. and Amari, S.V., A Multiple-Valued Decision Diagram Based Method for Efficient Reliability Analysis of Non-Repairable Phased-Mission Systems, *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 320-330, March 2014, doi: 10.1109/TR.2014.2299497.
15. Reed, S., An Efficient Algorithm for Exact Computation of System and Survival Signatures using Binary Decision Diagrams, Reliability Engineering and System Safety 2017; 165, 257-267.
16. Xing, L. and Amari, S.V., Binary Decision Diagrams and Extensions for System Reliability Analysis, Wiley, 2015, (ISBN 978-1-118-54937-7).
17. Petri, C.A., Kommunication mit Automaten., Technischen Hoschule Darmstadt (1962), Ph.D. thesis
18. Jensen, K. and Rozenberg, G., High-level Petri Nets: Theory and Application, Springer Science & Business Media, 2012.

19. Chiachío, M., Saleh, A., Naybour, S., Chiachío, J. and Andrews, J., Reduction of Petri Net Maintenance Modeling Complexity via Approximate Bayesian Computation, Reliability Engineering and System Safety, Vol 222, June 2022.
20. Zhou, J. and Reniers, G., Petri-Net Based Cooperation Modelling and Time Analysis of Emergency Response in Context of Domino Effect Prevention in Process Industries, Reliability Engineering and System Safety, Vol 223, July 2022.
21. Ramirez-Marquez, J.E. and Coit, D., A Monte-Carlo Simulation Approach for Approximating Multi-state Two-terminal Reliability, Reliability Engineering and System Safety, Vol 87, No 2, 2005, pp253-264.
22. Yin, J., Cui, L. and Balakrishnan, N., Reliability of Consecutive – (k,1)-out-of-n: F Systems with Shared Components under Non-Homogeneous Markov Dependence, Reliability Engineering and System Safety, Vol 224, 2022.
23. Meshkat, L., Dugan, J.B. and Andrews, J.D., (2001), Maintenance Modelling for Computer Based Systems, IMechE Proceeding Part E, Journal of Process Mechanical Engineering, 215(E3), pp 221-233, ISSN 0954-4089.
24. Meshkat, L., Dugan, J.B. and Andrews, J.D., (2002), Dependability Analysis of Systems with On-demand and Active Failure Modes, Using Dynamic Fault Trees, IEEE Transactions on Reliability, 51(2), 240-251, ISSN 0018-9529
25. Zhou, S., Ye, L., Xiong, S. and Xiang, J., Reliability Analysis of Dynamic Fault Trees with Priority-AND Gates Based on Irrelevance Coverage Model, Reliability Engineering and System Safety, Vol 224, 2022.
26. Chiacchio, F., Iacono, A., Compagno, L. and D'Urso, D., A General Framework for Dependability Modelling Coupling Discrete -Event and Time-dependent Simulation, Reliability Engineering and System Safety, Vol 199, 2020.
27. Audley, M. and Andrews, J.D., (2013), The Effects of Tamping on Railway Track Geometry Degradation, *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, 227)(4), pp 376 – 391.
28. Andrews, J, Prescott, D. and De Rozieres, F., 2014, A Stochastic Model for Railway Track Asset Management, Reliability Engineering and System Safety, Vol 130, pp76-84.
29. Platz, O. and Olsen J. V. "FAUNET: A Program Package for Evaluation of Fault Trees and Networks", Research Establishment Riso, Report No 348, DK-4000 Roskilde, Denmark, Sept. 1976.
30. Reay, K. and Andrews, J.D., (2002), A Fault Tree Analysis Strategy Using Binary Decision Diagrams, Reliability Engineering and System Safety, 78, pp 45-56.
31. Dutuit, Y. and Rauzy, A. A Linear-Time Algorithm to find Modules of Fault Trees, IEEE Trans. Reliability, **45**, No. 3, 1996.
32. Fenton, N. and Neil, M., (2012) , Risk Assessment and Decision Analysis with Bayesian Networks, CRC Press, pp524.
33. Zhou, Z. and Zhang, Q., Model Event/Fault Trees with Dynamic Uncertain Causality Graph for better Probabilistic Safety Assessment, IEEE Transactions on reliability, Vol 66, No 1, March 2017.
34. Apostolakis, G., How Useful is Quantitative Risk Assessment?, Risk Analysis, Vol 24, No 3, 2004.