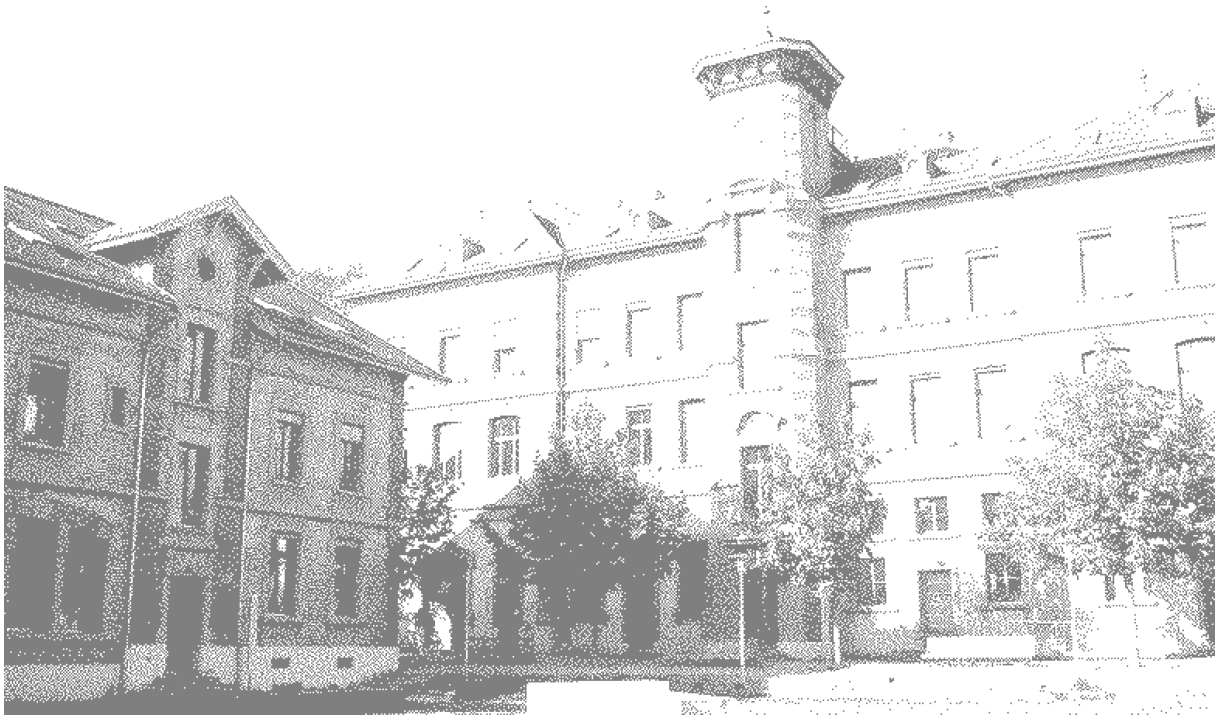
UNIVERSITÄT
BERN

Reliability Evaluation and Diagnostics with Propositional Directed Acyclic Graphs

J. Jonczy, R. Haenni

Technischer Bericht iam 09-004 vom 24. Juni 2009

Institut für Informatik und angewandte Mathematik, www.iam.unibe.ch



Reliability Evaluation and Diagnostics with Propositional Directed Acyclic Graphs

Jacek Jonczy, Rolf Haenni

Technischer Bericht iam 09-004 vom 24. Juni 2009

CR Categories and Subject Descriptors:
[I.2.4 Knowledge Representation Formalisms and Methods]: Representation languages; [C.4 Performance of Systems]: Reliability, availability, and serviceability; [G.3 Probability and Statistics]: Reliability and life testing; [G.2.2 Graph Theory]: Network Problems;

General Terms:
Theory, Reliability, Design, Algorithms

Additional Key Words:
reliability theory, network reliability, modular systems, diagnostics, propositional directed acyclic graphs, algebraic path problems

Institut für Informatik und angewandte Mathematik, Universität Bern

Abstract

The theories of reliability and diagnostics, despite their close relationship, have been mostly treated as two separate fields in the literature. However, it has been shown in the past few years that their dual character can be exploited in the context of modular systems. Such systems have a great practical impact on reliability computation and diagnostics, since their structure can be used to significantly reduce the computational effort needed for their evaluation. It is also known that the underlying structure function can be represented in a compact way by Propositional Directed Acyclic Graphs (PDAGs), which form a powerful Boolean representation language. Recently, PDAGs have been also applied successfully in the context of network reliability.

The present paper continues this line of research and proposes several extensions. First, we introduce a unifying formal model for system description, reliability evaluation, and diagnostics. Besides the formal specification of modular systems and reliability networks, the model allows networks and modules to be combined arbitrarily, leading to so-called hybrid systems. The underlying computational machinery provided by PDAGs allows to evaluate reliability in a convenient and efficient way. Second, we provide concise algorithms for both generating structure functions with respect to network reliability problems as well as for PDAG manipulations, and we also examine their complexity. Third, we show that posterior probabilities of system elements can be efficiently computed using PDAGs for diagnostic purposes, in particular in modular systems.

Contents

1 Introduction

Failures in complex technical systems are inevitable; there is probably no communication network, power plant, or production system which operates free of failures throughout its whole life time. Despite the overwhelming advances of the past years in modern technologies, it is still virtually impossible to produce hundred per cent reliable technical equipment. As a consequence, failures of system components are caused by material deterioration, construction flaws, unexpected random failures, or other defects. A system failure on its part may be caused by potentially any faulty component or a combination thereof. As a matter of fact, the spectrum of today's dependable systems is large and their correct operation is accordingly crucial for economy and society. To guarantee thus a high level of reliability, such systems demand a careful and profound analysis. This is the subject of reliability theory: the investigation of the behaviour (correct functioning) of a system given the behaviour of its components. We give a brief overview of reliability in the first part of Section 1.1.

But this is only one part of the game. Another fundamental problem arises when the system (or some parts thereof) is (are) observed to be malfunctioning. The goal is then to determine all possible diagnoses that explain the cause of the observed malfunction. Typically, when the system breaks down, one needs to know which component(s) is (are) responsible for the crash. This in general non-trivial task is called diagnostics and is an important field in Artificial Intelligence research. We look more closely on this topic in the second part of Section 1.1.

In the light of reliability and diagnostics, it is fundamental to study the internal structure of a system since it affects its dynamic and static properties, typically the fault-tolerance and vulnerability. Consequently this has a significant impact on reliability analysis and diagnostics. In the context of this paper, two structure-related issues are of particular importance: modules and networks. They are both discussed in Section 1.2.

It turns out that both reliability analysis and diagnostics are closely interconnected problems. Surprisingly, existing literature often lacks this view; the fields are mostly discussed separately. Exceptions include [1, 2] and [3]. In the latter, the authors already proposed a common computational framework for reliability and diagnostics in the context of modular systems. This paper continous the approach from [3] and extends the framework by considering reliability networks, hybrid systems (a combination of modular systems and reliability neworks), and multistate systems as further subjects of analysis. This results in a new unifying framework for reliability

and diagnostics in the context of modular, network, hybrid, and multistate systems.

1.1 The Theories of Reliability and Diagnostics

This section presents a short overview of reliability and diagnostics. It provides a classification of approaches and surveys the respective techniques used in the context of each model. The goals are to underline the relationship between reliability and diagnostics and to help the reader to better position our approach with respect to existing methods.

1.1.1 Reliability

In the early 1960s reliability gained enormous attention, resulting in a wide range of mathematical models and analysis techniques, including distribution functions, renewal theory, importance measures, and optimization techniques. Many of them still form the theoretical fundament in today's reliability theory. Important branches of reliability like network reliability, software reliability, and system reliability use and share many of these methods. See [4] and [5] for excellent historical surveys of reliability theories. Despite their variety both in theory and application, reliability methods focus on a common goal: to assess the reliability of a given system. At this point we shall look at the notion of reliability more closely. Reliability is generally concerned with the ability of a system to carry out a desired operation. The definition of reliability according to ISO 8402 (1986, 3.18) goes in a very similar direction:

> *The ability of an item to perform a required function under stated conditions for a stated period of time. The term "reliability" is also used as a reliability characteristic denoting a probability of success or a success ratio.*

Inspired by this definition we make a first general distinction between *static* and *dynamic* models of reliability. The study of dynamic models is motivated by the fact that physical systems usually alter in the course of time. These models take this into account by investigating various lifetime distributions of a system. Probably the most important measure in the research of dynamic models is the reliability function, also known as survivor function. It expresses the probability of an item to operate without failure for a given amount of time. Closely related is the failure rate, which reveals

information about an item's conditional probability of failure (on the limit), after a certain period of survival. The failure rate has played an essential role in past efforts to provide a proper mathematical classification of reliability functions, which finally resulted in the well-known IFRA closure theorem for coherent systems. Dynamic models allow a more realistic and accurate analysis of a system's behaviour, but they are inherently more complex than static models. The latter assume a fixed time interval or point in time, which reduces the probability distributions to single probability values. Of course this is a considerable simplification, but it allows to concentrate on the structure of a system and to leave out certain aspects like material deterioration which are not relevant for certain applications.

A common categorization encountered in reliability as well as in diagnostics are *qualitative* and *quantitative* methods. Qualitative methods primarily aim at accurately representing a system's *structure function*. Roughly speaking, the structure function describes the logical dependence between the operation of a system and the operation of its constituent components. There exist two fundamental representations of the structure function, namely *pathsets* and *cutsets*. A pathset is a subset of components whose operation guarantees the system operation, and a cutset is a subset of components whose failure implies system break-down. Pathsets and cutsets are thus dual representations. In monotone systems it suffices to consider *minimal pathsets* and *minimal cutsets*, as we shall see later. Much more sophisticated representations exist; we return to this topic in more detail in Subsection 3.1. From a quantitative point of view, one is interested in computing some meaningful measure of system operation or failure, whether its nature is dynamic or static. The range is considerable and goes from a simple probability of operation (or failure) and continues with reliability functions, failure rates, system state distributions, expectation value of operation, and many more. Besides applying exact methods, such measures are often approximated using Monte-Carlo techniques, or estimated using bounding techniques.

Another distinction is made between a *success-oriented* and a *failure-oriented* approach. In the former, one examines different scenarios of correct system behaviour (the success space), while in the latter one concentrates on various failure scenarios (the failure space). This is reflected by two popular diagram-based methods, namely Reliability Block Diagrams (RBD) and Fault Tree Diagrams (FTD).[1] Both are often used in the context of modular systems, see Subsection 1.2. The key difference

[1] As long as not quantified, diagram-based models such as RBD and FTD are sometimes refered to as *non-state-space models*.

between these methods is that block diagrams represent the system's success space while fault trees represent the failure space, resulting in the same dual relationship as between pathsets and cutsets. RDBs graphically represent the logical (reliability-wise) arrangement of a system's components which however may differ from how the components are physically arranged. Usually, an RBD is constructed bottom-up, i.e. starting from the lowest-level components, and successively grouping them together to modules which are in turn grouped to even larger modules. In this way, the success state of the system or module is expressed in terms of the success states of its individual components. The dual representation provided by fault trees allows a failure-driven view by logically arranging events in form of a tree: the nodes represent undesired events (often failures of system elements) and are connected by logical gates (*and*, *or*, etc.) to higher-level nodes. The construction is usually top-down: the system failure (called top-level event) is expressed in terms of intermediate events (its causing or initiating events), and so on, until component failures (called basic events) are reached. The construction of a fault tree is usually part of an elaborate process known as *fault tree analysis* (FTA).
Normally it is easy to convert a fault tree into a corresponding RBD, with some exceptions though.[2] The converse however is in general more difficult, especially if complex (e.g. network-like) configurations are involved.
Both RBD and FTD have been and still are extensively used by engineers for reliability and risk assessment of industrial systems, and they are integrated in many reliability-oriented software tools. Since these methods are diagram-based, they offer an excellent and intuitive means to understand, model, and analyse complex systems. Their usage is thus by far not restricted to scientists and engineers, but is also very common among system designers and operators, managers, and other people involved in systems analysis. Throughout this paper we will also make use of FTDs and RBDs, but only for illustrative purposes.

1.1.2 Diagnostics

Let us now turn the attention to diagnostics, the second topic of interest in this paper. Diagnostics is concerned with the malfunctioning of systems, or more precisely with finding the causes of a system's undesired behaviour. Most approaches are *model-based*, collectively known under the term *model-based diagnostics*. In a model-based approach, a suitable description (the *model*) of the original, physical system (the *artifact*)

[2] The *xor* gate for instance has no RBD equivalent.

is available.[3] The level of granularity or detail of the model may vary significantly, depending on the availability of empirical data about the artifact and also on the particular purpose of the model, e.g. troubleshooting or model refinement. The model may comprise the logical interconnectivity of the components, component definition, input and output of components, structure and design information, signal flow, and so on. Such a description is usually expressed in some formal language, typically sentences in constraint or first-order predicate logic, see [6, 7, 8, 9, 10]. The ultimate goal of the diagnostic task is to find all possible explanations for the behavioral discrepancy between the predicted behaviour of the model and the observed behaviour of the artifact. In the literature, different notions exist for the concept of explanations: [7] use the term *minimal candidates*, while [6] defines equivalent *minimal diagnoses*. The minimal candidates or diagnoses correspond to minimal sets of faulty components. Later, due to the discovered inadequacy of the concept of minimal diagnoses, the term *kernel-diagnoses* has been introduced by [8].

The methods for diagnostics as described above are mostly qualitative, but they are sometimes combined with quantitative methods. As for the qualitative part, the inference of candidates or diagnoses is typically accomplished by means of assumption-based truth maintenance systems (ATMS), at least in earlier literature: see [7] and [11]. Later approaches use probabilisitic argumentation systems (PAS), as in [9]. Quantitative methods include Bayesian techniques or minimum entropy methods, see [7]. The latter are used in conjunction with the incremental process of candidate generation, in which successive measurements are necessary to identify the set of effectively faulty components. The minimum entropy method allows the diagnostician to decide what measurement is the best to make next: it is the one that minimizes the expected entropy of the candidate probabilities resulting from this measurement.

Generally, diagnostics can be approached from two different sides. From an engineering perspective, diagnostics consists in troubleshooting technical systems to identify broken components. From a scientific point of view, the objective is rather to successively refine the model of a physical system based on empirical data. In either case, the observed model-artifact differences are crucial for the ultimate task: in the case of troubleshooting the model is assumed to be correct, so any model-artifact discrepancies indicate failures within the artifact, while in the case of model refinement the artifact is assumed to work properly, hence observed differences require

[3] In the engineering domain, often very accurate models of physical devices such as electronic circuits exist, based on structural information about the original device.

the model to be adapted.
In this paper we adopt the former approach for diagnostics, that is we presume the correctness of the model so that any observed failures indicate malfunctions of the artifact. As for the model, we use a compact structure function representation, in fact the same as for reliability. The structures and computations resulting from reliability evaluation are then reused for diagnostics for which we apply a probabilistic (Bayesian) approach.

1.2 The Structure of Complex Systems

Besides the stochastic properties of system components, the way in which they are assembled may have a significant impact on the overall system performance. To know the structure of a system is thus essential with regard to reliability analysis and diagnostics. In this subsection we briefly survey two common system architectures, namely modular systems and (reliability) networks.

1.2.1 Modularity of Systems

Technical systems often consist of components which themselves also constitute proper subsystems. Such subsystems may in turn comprise even smaller subsystems, and so on. This allows to look at the system from different levels of granularity: on the highest (or most coarse) level, the system consists simply of components which are considered as a whole, so that any underlying details are hidden. The system operation then just depends on the operation of those components. On a deeper (or finer) level however these components are considered themselves as systems which contain other subsystems or components. Then such a component is considered operational only if the underlying subsystem is operational. The decomposition of a system into smaller subsystems, also called *modules*, can be continued recursively until atomic components are reached, i.e. elements which cannot be further decomposed. This process leads to a so-called *modular decomposition* of the system, see [12]. Since modules are supposed to be mutually disjoint in terms of atomic components they comprise, a modular decomposition of a system may be represented in form of a hierarchical structure which is in fact a tree, called *organizing tree*. The root of this tree is formed by the top-level module and describes the overall system, intermediate nodes correspond to modules, and the leaves represent (atomic) system components.

The modular view of a system obviously has great practical importance since many technical systems are built up from nested modules. An important advantage of a modular structure is that in case of a defect it is often sufficient to exchange the faulty module as a whole without affecting the rest of the system. Moreover, a modular structure reduces the computational effort in both reliability analysis and diagnostics compared to a non-modular structure, as we shall see in the sequel.

1.2.2 Networks

Numerous technical large-scale systems appear in form of networks, typically communication and transportation networks, electrical power networks, and many others. Several decades ago, with the advent of networks as new, decentralized system architecture (e.g. to build fail-proof computer networks), considerable efforts have been invested in the exploration of various network topologies and the reliability analysis of the resulting networks. Since then, a wide range of models and techniques has been devoted especially to the computation of the reliability of networks, collectively known under the term *network reliability*, see [13, 14, 15, 16].

Today the task of network reliability computation arises in two fundamentally different situations. In the first case, one wishes to assess the reliability of a physical network, i.e. an existing technical system, based on its structure and stochastic properties. In the second case, some collection of data is modelled in terms of a graph or network, so that any relevant problems (queries) to be solved with respect to this data can be formulated as network reliability problems.[4]

The standard representation of networks by means of graphs allows to view a network simply as a set of nodes and edges. This is a simple model, but it is sufficient for many applications. Hence not surprisingly, graph problems and network problems are intimately related. As a consequence, the broad palette of graph-theoretical methods can be used to solve basic network reliability problems. Unfortunately, such methods are often not scalable in practice since their complexity grows exponentially with the network size. This has lead to the fact that theoretical complexity bounds for network reliability computations are often obtained by analogy with the solution of corresponding graph problems. This has been explored by [17] to show that most relevant network reliability problems are #P-complete in

[4] Such applications include certification or trust graphs, networks of biological data, social networks, citation networks, and many others.

the general case.[5]
To assess the reliability of a network, many reliability measures have been introduced which can be divided in separate classes: performability, vulnerability, connectivity, and many others. Most research in network reliability however is devoted to connectivity-based measures. As the name suggests, these measures are concerned with the connectivity of a specified subset of network nodes. We shall stick to connectivity-based measures in this paper as well.

1.2.3 The Structure Function and Computational Aspects

The key to all reliability computations is the structure function of the system: once the component reliabilities are known, the structure function allows to compute the reliability of the system. This raises two questions: first, how to obtain the structure function, and second, once the structure function is given, to what extent does the system layout affect the structure function and hence subsequent computations.
In a modular system, the physical structure is reflected in the logical structure, hence the structure function. This means that the structure function can be directly derived from the organizing tree. Reliability computation can be then performed bottom-up: starting at the leaves, compute the reliabilities of intermediate modules, and continue recursively until the root is reached at which the overall system reliability is obtained. The crucial point is now that a modular decomposition allows to compute the reliability of the system by computing intermediate structure functions which are possibly much less complex than the overall (non-modular) structure function. This reduces the computational effort needed for exact reliability analysis with respect to a non-modular approach which is #P-hard in general. The module reliabilities can be computed independently of each other, and furthermore the tree structure of the system guarantees that the respective computations remain feasible. We show how this works in Subsection 3.2. Modularity also comes into play for the diagnostic task, as we will see in Section 4: once the system and module reliabilities have been computed, we can use these results to compute posterior probabilities of modules and components given some observations about the system.
In contrast to a modular system, a network does not inherently reveal a logical structure unless a specific reliability problem has been specified.

[5] It is nevertheless desirable to compute the exact reliability in certain applications, although this is only feasible for small-scale networks or other restricted network classes.

So the structure function of a network depends on two criteria: on the one hand, it is affected by the network topology, and on the other hand it is determined by the specified problem, that is, some reliability measure. In the general case, exact network reliability computation is provably infeasible with respect to most reliability measures, see [17]. For this reason, research concentrates more and more on polynomial-time methods for restricted network classes or on approximation techniques. Our approach is based on a compact structure function representation and can be used in fact for both exact[6] and approximate reliability computation, though the emphasis lies on exact computation in this paper.
In systems analysis it is very common to consider binary systems, i.e. systems which can assume two possible states: operational or failed. This means that the underlying structure function is a Boolean function. Since the explicit specification of this function may become prohibitive in case of large systems (the number of configurations grows exponentially with respect to the number of variables), there is a strong need for sophisticated techniques to represent and manipulate Boolean functions. A novel and competitive technique based on Probabilistic Directed Acyclic Graphs (PDAGs), see [18], is the method of choice in this paper. Of particular importance is the subclass of decomposable and deterministic PDAGs, which are more compact with respect to many other representations and allow efficient probability computations. We return to this topic in more detail in Subsection 3.1.

1.3 Contributions and Outline

This paper presents a probabilistic approach for reliability and diagnostics. For the representation of the structure function we use the above mentioned subclass of PDAGs which is superior with respect to similar techniques. This serves as common framework for both reliability and diagnostics and allows the corresponding computations to be carried out efficiently. The method underlines the inherent duality between these two disciplines and also combines qualitative (structure function representation) and quantitative (reliability computation) features. Another contribution is a unifying formal model for modular systems, reliability networks, and hybrid systems which provides a formal basis for the subsequent system evaluation and allows networks and modules to be nested and combined arbitrarily. Algorithms are presented for generating structure func-

[6] In the general case, the structure function representation must be appropriately transformated to allow exact reliability computation to be carried out efficiently.

tions with respect to network reliability problems and subsequent PDAG manipulations, and their complexity is discussed. Furthermore, it is shown how posterior probabilities are efficiently obtained for diagnostics purposes in the context of modular systems.

The remaining part of this paper is organized as follows. Section 2 introduces basic notions and the formal model for our computational framework. Based on this model, Section 3 treats the reliability analysis of modular systems, reliability networks, and hybrid systems which combine modules and networks. In Section 4 it is shown how the results from Section 3 can be used to compute diagnostic queries and to find the most probable diagnoses. Finally, Section 5 concludes the paper by highlighting the key results and discussing open research problems.

2 A Unifying Formal Model for Modular Systems and Reliability Networks

This section provides the mathematical ground for reliability analysis and diagnostics. We start by introducing basic notions and discuss coherent systems in detail. After this we give two introductory examples: a coherent and a non-coherent system. Then we continue with a formal system description of modular systems, networks, and hybrid systems.

2.1 Preliminaries

We adopt in this paper the view of a *stochastic binary system*, or SBS for short [16]. The system consists basically of a set $\mathcal{C} = \{c_1, \ldots, c_m\}$ of $m \geq 1$ components where each component $c_i \in \mathcal{C}$ assumes either of two possible states: *operative* or *failed*. We associate with each component c_i a Boolean variable x_i which takes the value 1 if c_i is operative and 0 if c_i is failed. These m Boolean variables collectively constitute the Boolean state vector $\mathbf{x} = (x_1, \ldots, x_m)$. A *system state* is then one of the 2^m possible configurations (instantiations) of this Boolean vector. We denote the set of all such configurations by $\Omega^m = \{0, 1\}^m$, also called the *state space*.
In view of the reliability analysis presented later in this paper, we interprete the state of each component as a random variable and we moreover assume all those variables to be statistically independent.[7] We assume the existence of a mapping $r : \mathcal{C} \to [0, 1]$ which assigns to every component $c_i \in \mathcal{C}$ a reliability value $r(c_i) = P(x_i = 1)$. This represents the probability[8] that component c_i is working as expected for a given amount of time or a fixed point in time. The failure probability of a component c_i is then $q(c_i) = 1 - r(c_i) = P(x_i = 0)$, where $q = 1 - r$ is the complementary mapping of r.
Further we define a Boolean function[9] $f : \Omega^m \to \{0, 1\}$ which maps m Boolean state variables to an overall system state, namely

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ is an } \textit{operative} \text{ system state,} \\ 0 & \text{if } \mathbf{x} \text{ is a } \textit{failed} \text{ system state,} \end{cases}$$

[7] The independence assumption applies to all atomic elements encountered in this paper (including atomic vertices and edges).

[8] This is also called probability of operation, probability of success, reliability, or availability of c_i.

[9] See [20] for an extensive discussion on Boolean functions.

i.e. $f(\mathbf{x}) = 1$ iff the system is working correctly, and $f(\mathbf{x}) = 0$ otherwise. The function f is called the *structure function* and represents the system's success state, while $f^c = 1 - f$ is the complementary structure function representing the system's failure state. From now on, we denote an SBS by $\Sigma = (\mathcal{C}, r, f)$, or shorter by $\Sigma = (\mathcal{C}, f)$ when the probabilities are not relevant in a given context.
Finally, we denote by Rel a general reliability evaluation function, and we interpret $f(\mathbf{x})$ as a random variable. For a given system $\Sigma = (\mathcal{C}, r, f)$ the general computational problem of interest in reliability can be thus stated as the problem of computing the value

$$Rel(\Sigma) = P(f(\mathbf{x}) = 1) = 1 - P(f(\mathbf{x})^c = 1), \tag{1}$$

given some representation of f and stochastic information with respect to the system components. The *reliability* $Rel(\Sigma)$ of a system Σ is thus defined as the probability of the system to be in an operational state. Section 3 explores the computation of system reliability employing our proposed representation of the structure function.

2.2 Coherent Systems

In reliability theory, the structure function f is often required to be *monotone* which is the defining characteristic of a monotone system. This is the case if the following holds for all Boolean vectors $\mathbf{x}_1, \mathbf{x}_2 \in \Omega^m$:

$$\mathbf{x}_1 \leq \mathbf{x}_2 \Longrightarrow f(\mathbf{x}_1) \leq f(\mathbf{x}_2), \tag{2}$$

i.e. f is non-decreasing in each argument, and moreover it holds that $f(\mathbf{1}) = 1$ and $f(\mathbf{0}) = 0$, where $\mathbf{1} = (1, \ldots, 1)$ and $\mathbf{0} = (0, \ldots, 0)$. A monotone system is called *coherent* if all its components are relevant, i.e. none are irrelevant. A component c_i is irrelevant when its operation is insignificant for the system operation. This means that for vectors $\mathbf{x}_{[1_i]}$ and $\mathbf{x}_{[0_i]}$ with the i-th component x_i being 1 and 0, respectively, $f(\mathbf{x}_{[1_i]}) = f(\mathbf{x}_{[0_i]})$.
The assumption of monotonicity (or coherence) appears to be very reasonable in many cases: it basically states that repairing a failed component within an operative system cannot lead to system failure, and conversely the failure of a component cannot make a failed system operative. In practice, probably most systems of interest can be described by coherent models. Nevertheless, coherence cannot be assumed in general; for this reason we refrain from imposing that requirement in our model since we allow for non-coherent structures in our framework as well, see Example 2.

2.2.1 Deterministic Properties of Coherent Systems

Intimately related with every coherent system are the notions of *pathset* and *cutset*. A pathset $P \subseteq \mathcal{C}$ is a subset of components whose operation makes the system operative, independently of the states of the other components. A cutset $C \subseteq \mathcal{C}$ on the other hand is a set of components whose failure yields system failure, independently of the states of remaining components. A pathset P is called *minimal* if no proper subset of P is a pathset. Minimal pathsets are sometimes called *minpaths*. Minimal cutsets are defined analogously and are called *mincuts*. We denote the set of all minpaths by $\mathscr{P}$, and the set of mincuts by $\mathscr{C}$. In the context of a coherent system, the knowledge of minpaths or mincuts is sufficient to describe the structure function. Speaking in these terms, we can identify an irrelevant component as one that belongs neither to a minpath nor to a mincut, and thus can be removed as far as reliability is concerned. So for coherent systems it holds that

$$\bigcup_{P \in \mathscr{P}} P = \bigcup_{C \in \mathscr{C}} C = \mathcal{C}.$$

Two basic cases of coherent systems arise when $P = \mathcal{C}$ is the system's only pathset, and similarly when $C = \mathcal{C}$ is the system's only cutset. In the former case, the system is operative only if all components are operative at the same time. In the latter case, it is sufficient for one component to work correctly to make the system operative. The resulting systems are then called *series system* and *parallel system*, respectively. This leads us to a more general structure called k*-out-of-*n *system* where $k \leq n$. Such a system is supposed to work correctly only if at least k of its totally n components are operative. It can be easily verified that a k-out-of-n system has exactly $\binom{n}{k}$ minpaths and $\binom{n}{n-k+1}$ mincuts. Note that a series and a parallel system are special cases of a k-out-of-n system for $k = n$ and $k = 1$, respectively.

2.2.2 Probabilistic Properties of Coherent Systems

We shall discuss now another interesting property of monotone structure functions which allows to conveniently deal with interval-valued probabilities instead of fixed-valued probabilities. Relaxing the uniqueness assumption for probability values has gained a lot in importance in statistical reasoning research and Artificial Intelligence. It is inherent for instance in the theory of imprecise probabilities [21] and in the context of credal

networks [22]. The following discussion is thus not especially related to reliability, but rather refers to monotone Boolean functions in general and has, as we shall see, direct impact on reliability computation of coherent systems.

Recall the set $\mathcal{C} = \{c_1, \ldots, c_m\}$ of components together with the corresponding Boolean variables $x_1, \ldots, x_m$ representing the component states. These Boolean variables span the space Ω^m of all m-dimensional Boolean vectors. Consider further the probability space $(\Omega^m, \mathcal{P}(\Omega^m), P)$, where Ω^m is the sample space of interest and P is a probability measure defined by $P : \mathcal{P}(\Omega^m) \rightarrow [0, 1]$. In Subsection 2.1 we have assumed fixed-valued success probabilities $r(c_i) = P(x_i = 1)$ such that $r(c_i) \in [0, 1]$ for all components c_i. This is so far the classical approach in reliability. Now we consider also probabilities $r(c_i)$ which assume arbitrary (but unknown) values in a closed real interval $[a_i, b_i] \subseteq [0, 1]$ associated with each component $c_i \in \mathcal{C}$. This leads to the following family of probability measures:[10]

$$\mathbf{P} = \{P : \mathcal{P}(\Omega^m) \rightarrow [0, 1] \text{ such that } P(x_i = 1) = r(c_i) \in [a_i, b_i]\} \qquad (3)$$

From now on we abbreviate the probability $P(x_i = 1)$ by $P(c_i)$ both for fixed-valued and interval-valued probabilities. The general problem is now to determine the probability $P(E)$ of an arbitrary event $E \in \mathcal{P}(\Omega^m)$ given the probabilities $P(c_i)$. For this we define $\overline{P}(E)$ such that $\overline{P} \in \mathbf{P}$ to be the probability of E given that $P(c_i) = b_i$, and similarly $\underline{P}(E)$ where $\underline{P} \in \mathbf{P}$ is defined as the probability of E given that $P(c_i) = a_i$, for all $i = 1, \ldots, m$. Here we call $\overline{P}$ the *upper probability* and $\underline{P}$ the *lower probability*. In any case, having specified the probabilities $P(c_i)$ and assuming them to be independent, the probability measures P (given fixed-valued probabilities), $\overline{P}$ (given upper interval-bound probabilities), and $\underline{P}$ (given lower interval-bound probabilities) are uniquely determined.

Let us go back to the context of Boolean functions and consider an arbitrary BF $f : \Omega^m \rightarrow \{0, 1\}$ and the event

$$E = \{\omega_i \in \Omega^m : f(w_i) = 1\} \qquad (4)$$

which occurs whenever f evaluates to 1 given a m-dimensional Boolean vector $\omega_i \in \Omega^m$. Let us assume interval-valued probabilities $P(c_i)$. Now the crucial observation is the following: when the Boolean function f under consideration is monotone, the upper probability $\overline{P}(E)$ corresponds exactly to the largest possible probability of E, and analogously the lower

[10] In the literature of credal sets such family $\mathbf{P}$ is also called *closed convex set of probability measures*, see [23].

probability $\underline{P}(E)$ corresponds exactly to the smallest possible probability of E. This result is captured by the following theorem:

Theorem 2.1. Let be $E \in \mathcal{P}(\Omega^m)$ the event (4) and $\mathbf{P}$ the set of probability measures specified by (3). If the Boolean function f from (4) is monotone, then

$$(1)\ \ \underline{P}(E) = \min_{P \in \mathbf{P}} P(E) \quad \text{and} \quad (2)\ \ \overline{P}(E) = \max_{P \in \mathbf{P}} P(E).$$

Proof: see appendix.

This finding obviously has immediate consequences for monotone (or coherent) systems. To make it concrete, let be $\Sigma = (\mathcal{C}, r, f)$ a coherent system for which the component reliabilities $P(c_i)$ have been specified by intervals $[a_i, b_i]$. When we set $P(c_i) = b_i$, then $\overline{P}(f(\mathbf{x}) = 1)$ yields the *maximum* system reliability, and similarly for $P(c_i) = a_i$ in which case we obtain the *minimum* reliability. In this way we get the interval-valued system reliability $Rel(\Sigma) \in [\underline{P}(f(\mathbf{x}) = 1), \overline{P}(f(\mathbf{x}) = 1)]$, independently of the representation of the structure function f. This will be illustrated at the end of Example 1.

Remark In complex inference structures such as (compiled) credal networks where the underlying Boolean function is non-monotone, an exact computation of the upper and lower probabilities as described above is not possible. At the best these probabilities can be approximated, cf. [24].

Example 1. As an introductory example consider a simple mountain transit system. The overall system called alp-transit A consists of a mountain pass p and an underground transport system U. The latter consists of a railway section R which in turn devides into independent tracks r_1 and r_2, and a tunnel system t. The ability of the system to handle the transalpine (over- or underground) traffic load is expressed by the state of A. The latter is supposed to be operative if either p or the underground U are functioning as desired. For U to be accessible, both the tunnel and the railway system must be in service. The operation of R is guaranteed if at least one of the tracks is serviceable. This configuration and the success behaviour of the system is shown in Fig. 1a as an RBD, and the FTD in Fig. 1b depicts the system's failure-driven behaviour. The system is monotone and we can easily derive its minpaths: $\{p\}$, $\{r_1, t\}$, $\{r_2, t\}$ — and its mincuts: $\{r_1, r_2, p\}$ and $\{t, p\}$.

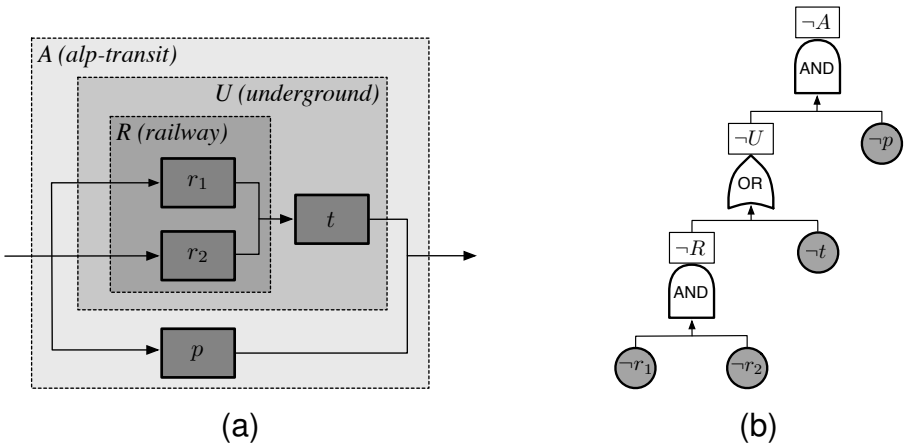


Figure 1: The alp-transit example as RBD (a) and as FTD (b).

We want to further investigate the behaviour and reliability of the system. For this we let the component names also denote corresponding Boolean variables which express the respective component states.[11] The Boolean variable A shall denote the overall system state. The functional relationship between the component states and the system state is established by the structure function f such that $A = f(r_1, r_2, t, p)$. Further we assume the system components to operate independently of each other with the following probabilities:

$$r(r_1) = r(r_2) = 0.9,\ r(t) = 0.8,\ r(p) = 0.7.$$

To compute the reliability of our sample system $\Sigma = (\{r_1, r_2, t, p\}, r, f)$, we may in a first attempt consider all possible configurations of f which reveals eleven success states and five failure states. Summing over all success states yields the system reliability $Rel(\Sigma)$, while by summing over all failure states we obtain the failure probability — or by simply computing $1 - Rel(\Sigma)$. We decide in favour of the former approach since there are less failure states than success states, and so we get the following sum of products and its probability:

$$\begin{aligned} P(A = 0) &= q(r_1) \cdot q(r_2) \cdot q(t) \cdot q(p) + r(r_1) \cdot q(r_2) \cdot q(t) \cdot q(p) \\ &\quad + q(r_1) \cdot r(r_2) \cdot q(t) \cdot q(p) + r(r_1) \cdot r(r_2) \cdot q(t) \cdot q(p) \\ &\quad + q(r_1) \cdot q(r_2) \cdot r(t) \cdot q(p) \\ &= 0.0006 + 0.0054 + 0.0054 + 0.0486 + 0.0024 \\ &= 0.0624. \end{aligned}$$

[11] For simplicity, this convention applies to all examples throughout the paper.

This is the unreliability or failure probability of the system. To obtain the reliability, we compute

$$Rel(\Sigma) = P(A = 1) = 1 - P(A = 0) = 1 - 0.0624 = 0.9376.$$

Obviously, this is not a very efficient approach. Alternatively we can use the above minpaths or mincuts for reliability computation which requires less effort. We consider the mincuts since there are only two of them and make them disjoint by adding the negation of t to the first mincut. This results in the Boolean formula $(\neg t \wedge r_1 \wedge r_2 \wedge p) \vee (t \wedge p)$ in sum-of-disjoint-products form, based on which we obtain the failure probability

$$\begin{aligned} P(A = 0) &= r(t) \cdot q(r_1) \cdot q(r_2) \cdot q(p) + q(t) \cdot q(p) \\ &= 0.0024 + 0.06 = 0.0624, \end{aligned}$$

which can be subtracted from 1 to get the system reliability $P(A = 1) = 0.9376$. In view of Subsection 2.2.2, let us repeat this computation by assigning now interval probabilities to components instead of fixed values. So let be

$$r(r_1) \in [0.8, 0.9],\ r(r_2) \in [0.8, 0.9],\ r(t) \in [0.7, 0.8],\ r(p) \in [0.6, 0.7].$$

Considering again the sum-of-disjoint-products formula from above and complementing the obtained failure probabilities, we get the upper probability $\overline{P}(A = 1) = 0.9376$ and the lower probability $\underline{P}(A = 1) = 0.8688$, and hence the interval-valued system reliability $Rel(\Sigma) \in [0.8688, 0.9376]$.
So far, these standard approaches are applicable in the context of any coherent system. We will show later in Subsection 2.3 how the modularity of a system can be exploited to make the represention and evaluation of the structure function much easier. To conclude this subsection, we present another example which illustrates a typical system that lacks the property of monotonicity.

Example 2. Consider a simple traffic light system for a two-road crossing. Both roads are one-way, and there is a two-signal (red, green) traffic light for either direction (1) and (2), i.e. there are two sets a and b of traffic lights in total, as depicted in Fig. 2. For simplicity, a and b also denote corresponding Boolean variables such that $a = 1$ if the traffic light is *green*, and $a = 0$ if it is *red*; the same holds for b. The reliability of the system is defined in terms of traffic flow: the system is operative only if the traffic flows without interruption and free of collision. This is the case only if one

light is on green while the other light is on red at the same time. Formally this leads to the system $\Sigma = (\{a, b\}, f)$ such that f is specified as follows:

$$f(a, b) = (a \wedge \neg b) \vee (\neg a \wedge b).$$

The structure function f describes the *xor* function, which clearly violates property (2). It can be still conveniently represented in form of a cd-PDAG as depicted in Fig. 4, based on which the reliability is easily computed using the framework from Section 3. The complementary structure function expressing the system failure is illustrated by the (non-coherent) fault-tree in Fig. 3.

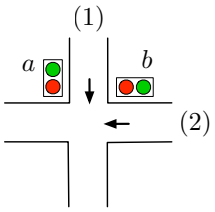


Figure 2: A road crossing.

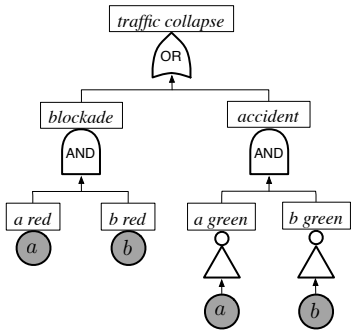


Figure 3: Non-coherent fault-tree for the crossing example.

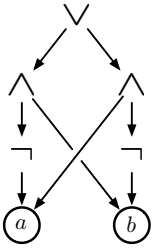


Figure 4: cd-PDAG for f in the crossing example.

Remark In general, whenever *not* logic is necessary to describe the system's structure function, the result is a non-coherent system. This is typically the case in the context of fault tree design when gates like *not*, *nor*, *nand*, and *xor* are used.[12] All these gates have a natural representation in terms of cd-PDAGs.

2.3 Modular Systems

Consider a stochastic binary system Σ that consists of a number of subsystems Σ_i, which are supposed to be arranged according to a predefined hierarchical structure, the so-called *organizing tree*. This construction is reflected by the system structure function f, as we shall see later. There are two different kinds of subsystems: atomic subsystems (called components) c_i with assigned stochastic properties, and those subsystems that

[12] Fault-tree gates involving *not* logic are sometimes called non-coherent gates, resulting in a corresponding non-coherent fault-tree.

constitute proper systems which are further decomposable into yet other subsystems whose structure follows the organizing tree. Such subsystems are also called *modules*. The set of all subsystems Σ_i shall be denoted by $\mathcal{S}$, and $\mathcal{C} \subset \mathcal{S}$ denotes the subset of atomic subsystems. We define further two selector functions with respect to each subsystem in $\mathcal{S}$, namely

$$\mathsf{comp} : \mathcal{S} \to \mathcal{P}(\mathcal{C}) \quad \text{and} \quad \mathsf{succ} : \mathcal{S} \to \mathcal{P}(\mathcal{S}),$$

where $\mathcal{P}(\mathcal{S})$ denotes the power set of $\mathcal{S}$. Given a subsystem Σ_i, $\mathsf{comp}(\Sigma_i)$ returns the set of underlying components which make up the subsystem at its lowest level. If Σ_0 denotes the top-level subsystem (i.e. the enclosing system), then the component set of the whole system is obtained through

$$\mathsf{comp}(\Sigma_0) = \bigcup_{i=1}^{m} \mathsf{comp}(\Sigma_i).$$

The function $\mathsf{succ}(\Sigma_i)$ returns the set of all direct descendants of Σ_i, also called the successor set, within the logical structure of Σ. From a local point of view, the operation of Σ_i depends directly on the operation of its immediate subsystems $\mathsf{succ}(\Sigma_i)$. Note that for atomic subsystems Σ_i we have $\mathsf{succ}(\Sigma_i) = \emptyset$ and $\mathsf{comp}(\Sigma_i) = \{c_i\}$. Given that, we may express the nested structure of a subsystem as follows:

$$\Sigma_i = \begin{cases} c_i & \text{if } \Sigma_i \in \mathcal{C}, \\ (\{\, \Sigma_{i_1}, \ldots, \Sigma_{i_k} \}, h_i\,) & \text{if } \Sigma_i \in \mathcal{S} \setminus \mathcal{C}, \end{cases} \tag{5}$$

where subsystems $\Sigma_{i_1}, \ldots, \Sigma_{i_k}$ constitute the successor set $\mathsf{succ}(\Sigma_i)$ and are recursively defined according to (5). Further, with each subsystem Σ_i a *local structure function* h_i is associated, which is defined as follows:

$$h_i : \Omega^{|\mathsf{succ}(\Sigma_i)|} \to \{0, 1\}, \tag{6}$$

where $\Omega^{|\mathsf{succ}(\Sigma_i)|}$ denotes the set of all Boolean vectors $\mathbf{x}_i$ associated with the direct descendants of Σ_i. The local structure function establishes thus the relationship between the operation of a subsystem Σ_i and the operation of its constituent elements on the uppermost level.
The tree structure of the system implies that the respective component sets of different subsystems are mutually disjoint. This is only the case if each subsystem appears in exactly one successor set. In other words, for any two subsystems Σ_i and Σ_j with $i \neq j$ we have

$$\mathsf{comp}(\Sigma_i) \cap \mathsf{succ}(\Sigma_j) = \emptyset \iff \mathsf{succ}(\Sigma_i) \cap \mathsf{succ}(\Sigma_j) = \emptyset. \tag{7}$$

Such subsystems Σ_i are also called *modules* or *modular subsystems*. Given a system Σ and its top-level structure Σ_0, the nested decomposition

$$\Sigma_0 = (\{\Sigma_1, \ldots, \Sigma_k\}, h_0),$$

where the Σ_i are defined according to (5), is called *modular decomposition* of Σ. A component c_i trivially constitutes a module, in which case h_i plays the role of a unary Boolean function corresponding to the Boolean variable x_i. Any system Σ has two trivial modular decompositions, one consisting of atomic elements only, and another consisting of the system itself. These special decompositions are of course not very useful.
We also define for every subsystem a *non-local structure function* $\phi : \mathcal{S} \rightarrow \{0, 1\}$, which is recursively specified in terms of corresponding local structure functions:

$$\phi(\Sigma_i) = \begin{cases} x_i & \text{if } \Sigma_i \in \mathcal{C}, \\ h_i(\phi(\Sigma_{i_1}), \ldots, \phi(\Sigma_{i_k})) & \text{if } \Sigma_i \in \mathcal{S} \setminus \mathcal{C}, \end{cases} \tag{8}$$

such that $\mathsf{succ}(\Sigma_i) = \{\Sigma_{i_1}, \ldots, \Sigma_{i_k}\}$. The non-local structure function of Σ_i constitutes the structure function of the subsystem represented by the whole subtree rooted at Σ_i down to its bottom elements. Now we come to the main definition of this section:

Definition A *modular system* is a system $\Sigma = (\mathcal{S}, f)$, where $\mathcal{S} = \{\Sigma_0\,, \Sigma_1\,, \ldots, \Sigma_n\,\}$ is the set of subsystems which are defined according to (5), and the structure function f is specified by

$$f(\mathbf{x}) = f(x_1, \ldots, x_k) = \phi(\Sigma_0), \tag{9}$$

the non-local structure function of the top-level module, defined according to (8). The vector $\mathbf{x} = (x_1, \ldots, x_k)$ is associated with the component states for all $c_i \in \mathcal{C} \subset \mathcal{S}$.

The system structure function f reflects thus the hierarchy of local structure functions h_i of the entire system, whereas h_0 expresses the operation of Σ_0 in terms of its immediate submodules only.
In the particular case when there is only one non-atomic subsystem Σ_0, the local structure function h_0 becomes the (non-local) structure function of the corresponding (non-modular) system $\Sigma' = (\mathcal{C}, h_0)$. Conversely, every non-modular system $\Sigma = (\mathcal{C}, f)$ is equivalent to the modular system $\Sigma'' = (\{\Sigma_0\,, c_1, \ldots, c_n\}, f)$, where $\mathsf{succ}(\Sigma_0) = \mathcal{C} = \{c_1, \ldots, c_n\}$.

Example 3. Recall the alp-transit example from Subsection 2.2. To make it more challenging, it has been extended resulting in the following modified system: the overall system A consists now of two modules U (underground) and P (mountain pass). The mountain pass consists of two sections, a rather steep and dangerous ascending part p_1 and a smoother and more safe descending part p_2. As before, the underground system is divided into a railway section R and a tunnel system T. Module R still consists of two independent tracks r_1 and r_2, but T now splits up in three equal tunnel components t_1, t_2, and t_3. Thus the system consists now of five modules and seven components. The new configuration is illustrated by the organizing tree in Fig. 5a, and the behaviour of the alp-transit system is reflected in the RBD in Fig. 5b. Note that the configuration of module T is now a 2-out-of-3 structure: at least two out of the three tunnel sections must be in service to be able to manage the traffic load.

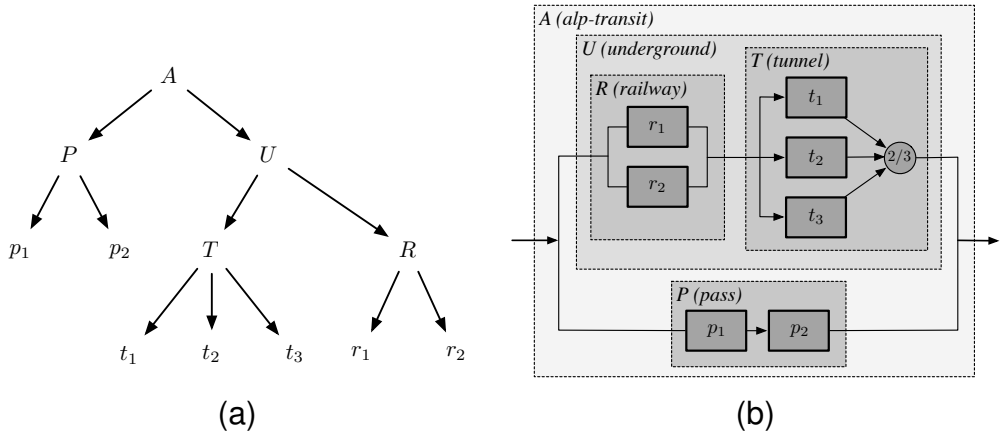


Figure 5: (a) Organizing tree of the extended alp-transit system, and (b) the corresponding RBD.

In terms of the above formal setting we obtain the system $\Sigma_{alptransit} = (\mathcal{S}, f)$ with the set of subsystems $\mathcal{S} = \{A, U, T, R, P, r_1, r_2, p_1, p_2, t_1, t_2, t_3\}$ and structure function f. The top-level module A corresponds to the root node in the organizing tree. As before, we let the names of the subsystems also denote the corresponding Boolean state variables. Like in Example 1 variable A represents the state of operation of the entire alp-transit system which can be expressed in terms of the component states by the global structure function: $f(r_1, r_2, p_1, p_2, t_1, t_2, t_3) = A$.
By following the modular structure of the system and using Def. 2.3, we can express the system state in terms of the subsystem states using the

corresponding local structure functions:

$$A = h_A(h_U(h_R(r_1, r_2), h_T(t_1, t_2, t_3)), h_P(p_1, p_2)),$$

where the local structure functions can be written as Boolean formulae conforming to the RBD in Fig. 5b:

$$\begin{aligned} h_A(U, P) &= A = U \vee P, \\ h_U(R, T) &= U = R \wedge T, \\ h_P(p_1, p_2) &= P = p_1 \wedge p_2, \\ h_T(t_1, t_2, t_3) &= T = (t_1 \wedge t_2) \vee (t_1 \wedge t_3) \vee (t_2 \wedge t_3), \\ h_R(r_1, r_2) &= R = r_1 \vee r_2. \end{aligned}$$

Finally, by inserting the local structure functions into each other, we obtain the (expanded) global structure function which represents the structure function of the entire system:

$$\begin{aligned} A &= f(r_1, r_2, p_1, p_2, t_1, t_2, t_3) \\ &= \big[\, (r_1 \vee r_2) \wedge \big(\, (t_1 \wedge t_2) \vee (t_1 \wedge t_3) \vee (t_2 \wedge t_3) \,\big) \,\big] \vee (p_1 \wedge p_2). \end{aligned}$$

In Subsection 3.2 we will demonstrate how to use PDAGs for the representation of local and non-local structure functions and for efficient reliability computation. In Section 4 we show how to use these results to compute the posterior probabilities of possible diagnoses.

2.4 Reliability Networks

In this section, we discuss systems which in general do not exhibit a modular structure: networks. We assume a probabilistic network model, which allows to define reliability networks, stochastic networks which assume random failures of its components. After this, we discuss network reliability problems which determine the logical structure of a reliability network.

2.4.1 Network Model

A network can be naturally represented by a mathematical graph $N = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a set of vertices or nodes[13] and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ of (directed or undirected) edges. We assume that edges represent pairs

[13]The terms *vertices* and *nodes* will be used interchangably throughout the rest of the paper.

of nodes. In case the pairs are ordered, the network is directed and we write $e_{ij} = (v_i, v_j)$ to denote the *directed* edge from v_i to v_j. Then $s(e_{ij}) = v_i$ denotes the edge's *source node* while $t(e_{ij}) = v_j$ denotes its *target node*. If we consider the pairs to be unordered, the network is *undirected* and an edge joining v_i and v_j is written as $e_{ij} = \{v_i, v_j\}$. When it is clear from the context which nodes are connected by an edge, we number the edges by an index just as we do with vertices. Given a directed network and two of its nodes v_s and v_t, an ordered set $P = \{e_1, e_2, \ldots, e_r\} \subseteq \mathcal{E}$ of edges is called *network path* from v_s to v_t if $s(e_{k+1}) = t(e_k)$ for all $1 \leq k < r$, such that $s(e_1) = v_s$ and $t(e_r) = v_t$. If all nodes along the path are distinct, the path is named *simple*. A simple path clearly contains no cycles.
These are so far deterministic properties of a network. In a probabilistic network model, we assume stochastic information associated with the network components to be available. In such model, the main goal consists in computing the *average performance* of the network, under the presence of random failures. This leads us to the following definition:

Definition A *reliability network* is a finite (stochastic) graph $N = (\mathcal{V}, \mathcal{E}, r)$ where $\mathcal{V}$ and $\mathcal{E}$ are respective sets of nodes and edges, and r is a mapping that assigns to each network component an operation probability (or reliability). Further, the nodes and edges are assumed to be subject to random and independent failures which occur with probabilities $q = 1 - r$.

When the probabilities are not relevant in a given context, we just refer by $N = (\mathcal{V}, \mathcal{E})$ to a reliability network. A reliability network in which all vertices and edges are assumed to be atomic will be called *simple network*.

2.4.2 Network Reliability Problems

So far we have described the physical structure and stochastic properties of a network. But what is the logical structure of a network? To obtain the logical structure (i.e. the structure function), we need to specify a certain reliability problem (or query) with respect to the given network. We consider connectivity measures as network reliability problems of interest in this paper, in particular the family of K*-terminal measures*. For a directed network, the most general such measure is called *source-to-*K*-terminal connectivity*, denoted by $conn_{\forall K}$. This is the probability that for a given set of terminal nodes $K \subseteq \mathcal{V}$ with $|K| = k$ and a specified source node $s \in K$ there is an operating network path from s to each node in K. Two important special cases of this measure arise for $k = 2$ and $k = |\mathcal{V}|$: in the former case we get the *source-to-terminal connectivity* (or s, t*-connectivity*

for short) denoted by $conn_{s,t}$, and in the latter we obtain the *source-to-all-terminal connectivity*, denoted by $conn_{\mathcal{V}}$.[14] Another measure we consider here is the *source-to-any-terminal connectivity*, denoted by $conn_{\exists K}$, which is the probability that there is a directed path from $s \in K$ to some (not necessarily all) nodes in K. Note that the measure $conn_{s,t}$ is then a special case of both, $conn_{\forall K}$ and $conn_{\exists K}$. All these measures have corresponding counterparts in an undirected network. With respect to network reliability evaluation discussed in Subsection 3.3, we shall focus on the s,t-connectivity measure, although other above-mentioned problems are also covered by our method.
A reliability network N for which a specific reliability problem (query) $\mathscr{Q}$ has been determined is denoted by

$$N[\,\mathscr{Q}\,] = (\mathcal{V}, \mathcal{E}, r, g),$$

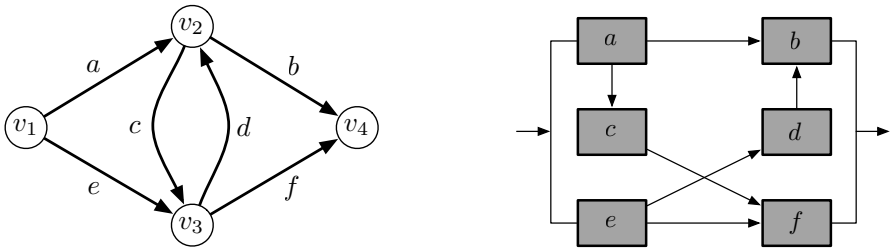
where g denotes the corresponding structure function. The reliability of $N[\,\mathscr{Q}\,]$ is then determined by computing the value $Rel(N[\,\mathscr{Q}\,])$ as stated in (1) by evaluating the structure function g. For example, $N[\,conn_{s,t}\,]$ denotes a network which has been *instantiated* with the s,t-connectivity measure.
To simplify the discussion of the proposed reliability analysis in Subsection 3.3, we consider in this paper only *directed* networks with *edge failures*. The following transformations show that such networks are sufficiently powerful on a conceptual level [14]: First, undirected networks are easily transformed into directed networks by replacing every undirected edge by two corresponding opposing directed edges. Each directed edge inherits then the failure probability (and other stochastic properties) of the original undirected edge. This is allowed as far as reliability is concerned since such transformation preserves the network reliability[15]. Second, any network with node failures is polynomial-time reducible into an equivalent directed network with edge failures only. Hence we assume from now on perfect nodes so that any network failures are due to edge failures.

Example 4. Consider the directed network depicted in Fig. 6a and suppose we are interested in the probability that there is an operating path

[14] The $conn_{s,t}$ measure may be applied for routing problems, e.g. when a message or packet needs to be routed between two specified network nodes. The $conn_{\mathcal{V}}$ measure may be appropriate in situations in which nodes are equally important with respect to receiving a message from the source node. This is the case in a broadcast network for instance.

[15] This claim holds for many reliability measures, e.g. connectivity measures, stochastic path problems, network flow measures.

from v_1 to v_4, i.e. the v_1, v_4-connectivity. The RBD of the network with the imposed reliability problem is shown in Fig. 6b. The minpaths are the following: $\{a, b\}$, $\{a, c, f\}$, $\{e, f\}$, $\{e, d, b\}$, and the mincuts are: $\{a, e\}$, $\{b, f\}$, $\{a, c, f\}$, and $\{e, d, b\}$. Note that the minpaths correspond precisely to the simple paths from source node v_1 to terminal node v_4 in the given network. As in Example 1, we could similarly use the minpaths or mincuts as basis for reliability computation. However, we present in Subsection 3.3 more sophisticated approaches to generate a compact PDAG representation of the structure function directly from the network.



(a) (b)

Figure 6: (a) A simple reliability network, and (b) the corresponding RBD for the v_1, v_4-connectivity.

2.5 Hybrid Systems

Models of large real-world systems are abstractions of their complex intern structure: they hide the details of underlying elements or subsystems. The idea is to leave out certain aspects which may not have a significant impact on the correct behaviour of the system. With the goal of reliability analysis in mind, one is interested in models that are precise and, at the same time, that hide (to a certain extent) the system's underlying complexity. Consequently, one is faced with the tradeoff between constructing an accurate model of the physical system on the one hand, and on the other hand keeping the model as simple as possible to assure the system analysis to be feasible. So we could ask the question what may give reason to increase the level of granularity of a model. It may happen for instance that one observes an element to fail more often than predicted, which makes this element being subject to more intense investigation. In such situation it may be useful to refine the model, i.e. to recursively go deeper into the element's intern structure and to provide a separate model for the corresponding subsystem, which may be of very different nature as the enclosing system.

In view of the above considerations we extend the formal framework of modular systems and reliability networks introduced so far in a way that allows them to be combined and nested arbritrarily, resulting in what we call *hybrid systems*[16] and *hybrid networks*.

2.5.1 Hybrid Networks

Let us first consider networks whose constituent elements — nodes and/or edges — can exhibit non-simple structures. For example, the nodes of a communication network may be transmitting stations which are further decomposable into more complex subsystems. Similarly, the connections of a transportation network may abstract away from more complicated structures like for instance traffic control systems or tunnel systems. In such a setting, we can still adopt the usual abstract appraoch, i.e. to regard a network in its basic structure as a set of nodes and edges, but with the additional feature that elements may now have a modular structure: they can be decomposed into further components or modules in the sense of Subsection 2.3, or they may constitute networks. Given that a network consists of vertices V_i and edges E_j, we can formalize this idea as follows:

$$V_i = \begin{cases} v_i & \text{atomic node,} \\ N_i[\,\mathscr{Q}_i\,] = (\mathcal{V}_i, \mathcal{E}_i, g_i) & \text{reliability network,} \\ \Sigma_i = (\{\Sigma_{i_1}, \ldots, \Sigma_{i_k}\}, h_i) & \text{modular system,} \end{cases} \tag{10}$$

and similarly

$$E_j = \begin{cases} e_j & \text{atomic edge,} \\ N_j[\,\mathscr{Q}_j\,] = (\mathcal{V}_j, \mathcal{E}_j, g_j) & \text{reliability network,} \\ \Sigma_j = (\{\Sigma_{j_1}, \ldots, \Sigma_{j_r}\}, h_j) & \text{modular system.} \end{cases} \tag{11}$$

Hence vertices and edges can be atomic elements, or they can represent self-contained modular systems or networks. In the latter case, the network may be simple or hybrid, see next definition. A modular system is defined according to Definition 2.3. We will see below that the above subsystems may exhibit in fact an even more complex structure. But first let us define a hybrid network:

[16] In this context, the notion of a hybrid system should not be confused with the same term known from physics, where a hybrid system refers to a dynamic system that exhibits continuous as well as discrete dynamic behaviour.

Definition A *hybrid network* is a reliability network $N = (\mathcal{V}, \mathcal{E}, r)$ where $\mathcal{V}$ and $\mathcal{E}$ are finite sets of (possibly non-atomic) nodes and edges such that their members are specified according to (10) and (11), and r is the reliability assignment function. With each vertex $V_i \in \mathcal{V}$ and edge $E_j \in \mathcal{E}$ corresponding structure functions f_{V_i} and f_{E_j} are associated as follows:

$$f_{V_i}(\mathbf{x}_i) = \begin{cases} y_i & \text{if } V_i = v_i, \\ g_i(\mathbf{x}_i) & \text{if } V_i = N_i[\,\mathscr{Q}_i\,], \\ h_i(\mathbf{x}_i) & \text{if } V_i = \Sigma_i, \end{cases} \quad f_{E_j}(\mathbf{x}_j) = \begin{cases} z_j & \text{if } E_j = e_j, \\ g_j(\mathbf{x}_j) & \text{if } E_j = N_j[\,\mathscr{Q}_j\,], \\ h_j(\mathbf{x}_j) & \text{if } E_j = \Sigma_j, \end{cases}$$

where y_i and z_j are respective Boolean variables associated with atomic nodes and edges.

A hybrid network reduces to a simple network if $V_i = v_i$ for all $1 \leq i \leq n$ and $E_j = e_j$ for all $1 \leq j \leq m$. The set of components of a hybrid network N consists of the relative component sets of its modular elements:

$$\mathsf{comp}(N) = \left(\bigcup_{i=1}^{n} \mathsf{comp}(V_i)\right) \cup \left(\bigcup_{j=1}^{m} \mathsf{comp}(E_j)\right).$$

Note that Property (7) is still valid for all modular vertices and edges, but also applies to these systems among each other.

2.5.2 Hybrid Systems

Another kind of hybrid system is obtained by considering also the converse case: networks being integral parts of modular systems. To be more precise, we can allow for modular systems which contain (simple or hybrid) networks besides regular modules and components. This means that modular systems and networks can be nested arbitrarily. In the general case in which the system consists of more than one subsystem, the restriction applies that a network cannot constitute the top-level subsytem Σ_0 since the overall structure of the system is supposed to be modular. Given a system Σ consisting of subsystems Σ_i, the decomposition (5) of each Σ_i can be now extended as follows:

$$\Sigma_i = \begin{cases} c_i & \text{if } \Sigma_i \text{ is a component,} \\ (\{\Sigma_{i_1}, \ldots, \Sigma_{i_k}\}, h_i) & \text{if } \Sigma_i \text{ is a module,} \\ N_i[\,\mathscr{Q}_i\,] = (\mathcal{V}_i, \mathcal{E}_i, g_i) & \text{if } \Sigma_i \text{ is a (hybrid) network } (i \neq 0), \end{cases} \tag{12}$$

where subsystems Σ_{i_j} are recursively specified according to (12). In order to fully specify the structure function of such a system, a specific measure

$\mathscr{Q}_i$ must be imposed with respect to each involved network N_i. Based on the above decomposition the following definition extends the notion of a modular system:

Definition A *hybrid system* is a system $\Sigma = (\{\Sigma_0, \Sigma_1, \ldots, \Sigma_m\}, f)$ consisting of subsystems Σ_i which are specified according to (12), for which the (non-local) structure function ϕ is given by

$$\phi(\Sigma_i) = \begin{cases} x_i & \text{if } \Sigma_i \text{ is a component } c_i, \\ h_i(\phi(\Sigma_{i_1}), \ldots, \phi(\Sigma_{i_k})) & \text{if } \Sigma_i \text{ is a module,} \\ g_i & \text{if } \Sigma_i \text{ is a (hybrid) network } N_i[\,\mathscr{Q}_i\,] \ (i \neq 0), \end{cases}$$

and the system structure function f is given by

$$f(\mathbf{x}) = \phi(\Sigma_0),$$

where vector $\mathbf{x}$ is associated with the states of components c_i.

The fundamental property of modular systems is supposed to apply here as well, so that a hybrid system without networks simply reduces to a modular system in the sense of Def. 2.3. Another special case occurs when a hybrid system consists only of one single subsystem, i.e. $\Sigma = (\{\Sigma_0\}, f)$. When Σ_0 is a network N with structure function $g = f$, then the whole system Σ reduces to the network $N[\,\mathscr{Q}\,] = (\mathcal{V}, \mathcal{E}, f)$.

Example 5. Let us illustrate these concepts by a new example. Consider for this the simple transport network depicted in Fig. 7 connecting four Swiss cities: Berne (b), Zurich (z), Chur (c), and Lugano (l). The labels put in parantheses next to edge names refer (roughly) to the actual highways that join the cities.

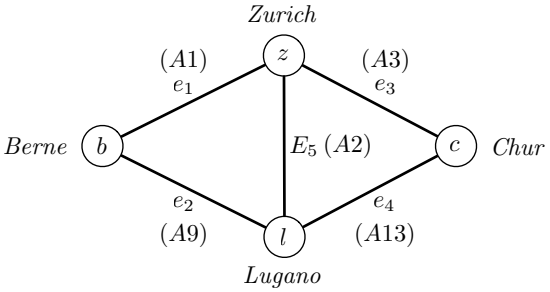


Figure 7: A simple transportation network with four atomic edges and one modular edge.

The picture in Fig. 7 reveals that not all connections are atomic: there are four atomic edges e_1, e_2, e_3, e_4 and one modular edge E_5. We let the modular edge be the alp-transit system from Example 3. This leads to a hybrid network with four atomic vertices, four atomic edges, and one modular edge. In addition, suppose we are interested in getting from Berne to Chur. The appropriate network reliability measure in this case is thus the b, c-connectivity. Formally, the whole system can be now expressed as follows:

$$N[\, conn_{\{b,c\}}\,] = (\{b, z, c, l\}, \{e_1, e_2, e_3, e_4, E_5\}, g),$$

where $g(e_1, e_2, e_3, e_4, E_5)$ is the corresponding network structure function and $E_5 = \Sigma_{alptransit}$.

3 Reliability Evaluation

This section is devoted to reliability analysis based on the formal setting established in the previous section. We adopt a success-oriented and probabilistic approach. Furthermore we follow a static reliability analysis, i.e. we consider the reliability of a system in a fixed point in time[17]. The following subsection establishes the computational core for reliability computation and diagnostics. Its main concern is an efficient representation and manipulation of Boolean functions.

3.1 Representing the Structure Function

In binary systems, the representation of the structure function f is reduced to the problem of representing a Boolean function. The most pragmatic way of representing the structure function is simply to consider all possible system states, also called *complete state enumeration*. But this becomes impracticable with a large number of components. With regard to efficient reliability computation, applying more sophisticated representation techniques is thus crucial. Subsection 3.1.1 briefly surveys the most important existing methods, and after that Subsection 3.1.2 presents in more detail the technique promoted in this paper.

3.1.1 Existing Methods

A common approach in reliability analysis to handling complex structure functions is to translate some system description, usually available in form of a block diagram or fault tree, into corresponding *pathsets* or *cutsets*. From the point of view of propositional logic, this corresponds to a representation of the underlying Boolean function in form of a Disjunctive Normal Form (DNF), or Conjunctive Normal Form (CNF) for a dual representation. In the context of model-based diagnostics, DNFs are used to represent minimal explanations or minimal conflicts.
A DNF representation of a Boolean function has two major drawbacks. The first comes from the fact that the number of terms grows exponentially with the number of Boolean variables. To overcome this diffulty, more compact representations have been proposed for reliability analysis, most prominent thereof is the family of *Binary Decision Diagrams* (BDD), see

[17] Conceptually, it would be possible to consider time-dependent probability values. But to keep the discussion simpler we refrain from doing so in this paper.

[25], and [26] for multistate systems. The second problem of a DNF is the lack of support for efficient probability computation. The classic approach to tackle this problem is to apply the *inclusion-exclusion principle* known from combinatorics and probability. This leads to a representation with disjoint terms, also called *disjoint DNF*. However, this transformation suffers an exponential blow-up in the number of terms. An improvement of this process can be achieved by applying the *domination* method [27] which considers non-cancelling terms from the beginning, resulting in a reduced inclusion-exclusion expansion. Further effort has been invested to solve the disjointness problem resulting in so-called *sum-of-disjoint products* (SDP) methods — notably the algorithms of Abraham [28] based on single-variable inversion, and Heidtmann [29, 30] based on multi-variable inversion. An extension of the Heidtmann algorithm for non-monotone BF has been also discussed in [31].

When represented as a directed graph or tree, however, the resulting structures in most SDP approaches exhibit the flatness property[18], leading to a breadthwise expansion and hence a relatively high space demand. This problem is eliminated in so-called *free* or *ordered BDDs* (OBDD), which also support probability computation in polynomial time with respect to their size [32]. In the context of network reliability, OBDDs are often generated from pathsets, cutsets, and edge expansion diagrams [33], or directly from the graph, see [34, 35].

A similar but even more powerful approach to the SDP problem is provided by *Propositional Directed Acyclic Graphs* (PDAG) [18]. The method proved very useful for reliability and diagnostics [3] and recently also in the context of network reliability evaluation [36]. Of particular interest are members of a certain subclass of PDAGs which are *decomposable* and *deterministic*. These two properties guarantee probability computations to be carried out in polynomial time with respect to the PDAG size. This is crucial since calculating probabilities is an essential operation in the context of reliability analysis and diagnostics. The key feature that sets this particular PDAG family apart from similar techniques like OBDDs and disjoint DNFs is their superior succinctness (i.e. compactness) and at the same time their ability to still support probability computations (and a number of further operations) in polynomial time. This is the main motivation why PDAGs constitute the core of our framework for reliability analysis and diagnostics. A short introduction to PDAGs is presented in the following subsection.

[18] Flatness implies a depth of at most two; this property is discussed in more detail in [18].

3.1.2 Propositional Directed Acyclic Graphs

Propositional Directed Acyclic Graphs (PDAGs) form a graph-based language for the representation of Boolean functions (BF) [18]. The main attention lies on the compactness of the representation provided by PDAGs, which include BDDs, NNFs, CNFs, DNFs, and many other Boolean representation languages as special cases. They are a powerful and flexible tool with many applications in different areas, such as Bayesian networks, formal verification, system reliability and diagnostics, network reliability, and many more. In the sequel, we provide a formal introduction to PDAGs.
Consider a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and a set V of n Boolean variables. Let us further define the *satisfying set* $[\![f]\!] = \{\mathbf{x} \in \{0,1\}^n : f(\mathbf{x}) = 1\}$ of f, which is the set of n-dimensional Boolean vectors $\mathbf{x}$ for which f evaluates to 1. The goal is to find an efficient representation for $[\![f]\!]$. This motivates the following definition [18]:

Definition A *PDAG* is a rooted, directed acyclic graph such that:

1. Leaf nodes are represented by $\bigcirc$ and labeled with $\top$ (true), $\bot$ (false), or $x \in V$
2. Non-leaf nodes are represented by $\wedge$ (logical and), $\vee$ (logical or), or $\neg$ (logical not); $\wedge$-nodes and $\vee$-nodes have at least one child, $\neg$-nodes have exactly one child.

By convention, PDAGs are denoted by lower-case Greek letters. The set of all possible PDAGs with respect to V constitutes a language which is denoted by $\texttt{PDAG}_V$, or just PDAG when the variable set is clear from the context. Leaves labeled with $\top$ ($\bot$) represent the constant BF which evaluates to 1 (0) for all $\mathbf{x} \in \{0,1\}^n$. A leaf labeled with $x \in V$ is interpreted as the assignment $x = 1$, i.e. it represents the BF which evaluates to 1 iff $x = 1$. The BF represented by a $\wedge$-node evaluates to 1 iff the BFs of all its children evaluate to 1. Similarly, a $\vee$-node represents the BF that evaluates to 1 iff the BF of at least one child evaluates to 1. Finally, a $\neg$-node represents the complementary BF of its child, i.e. the BF that evaluates to 1 (0) iff the BF of its child evaluates to 0 (1). The PDAG representation of a BF is not canonical, i.e. a BF may have several (logically equivalent) PDAG representations. Conversely, each PDAG φ represents exactly one BF. The BF f represented by a PDAG φ is denoted by f_φ. The set of all variables included in the PDAG φ is denoted by $vars(\varphi)$, and the set of all child-PDAGs of φ is denoted by $children(\varphi)$. The number of nodes of φ is denoted by $\#(\varphi)$, and the number of its edges is called its *size* and written $|\varphi|$.

Decomposability and Determinism. PDAGs can be characterized by a number of different properties, but in the context of this paper, the following two are particularly relevant:

- *Decomposability*: holds if the variable sets of the children of each $\wedge$-node α in φ are pairwise disjoint (i.e. if $children(\alpha) = \{\beta_1, \ldots, \beta_l\}$, then $vars(\beta_i) \cap vars(\beta_j) = \emptyset$ for all $i \neq j$);
- *Determinism*: holds if the children of each $\vee$-node α in φ are pairwise logically contradictory (i.e. if $children(\alpha) = \{\beta_1, \ldots, \beta_l\}$, then $\beta_i \wedge \beta_j \equiv \bot$ for all $i \neq j$).

A decomposable and deterministic PDAG is called cd-PDAG. We use `cd-PDAG` to refer to the corresponding language which is a sub-labguage of `PDAG`. Figure 4 shows a simple cd-PDAG.
Other sub-languages are obtained by considering further properties: `d-DNNF` (decomposable and deterministic negation normal forms) is the sub-language of `cd-PDAG` satisfying *simple-negation*, `FBDD` (free BDDs) is the sub-language of `d-DNNF` satisfying *decision* and *read-once*, `OBDD` (ordered BDDs) is the sub-language of `FBDD` satisfying *ordering*, and `d-DNF` (disjoint DNF) is the sub-language of `d-DNNF` satisfying *flatness* and *simple-conjunction*.[19]

Succinctness. A language $\mathtt{L}_1$ is *more succinct* than another language $\mathtt{L}_2$, $\mathtt{L}_1 \preceq \mathtt{L}_2$, if any sentence $\alpha_2 \in \mathtt{L}_2$ has an equivalent sentence $\alpha_1 \in \mathtt{L}_1$ whose size is polynomial in the size of α_2. A language $\mathtt{L}_1$ is *strictly* more succinct than another language $\mathtt{L}_2$, $\mathtt{L}_1 \prec \mathtt{L}_2$, iff $\mathtt{L}_1 \preceq \mathtt{L}_2$ and $\mathtt{L}_2 \not\preceq \mathtt{L}_1$. With respect to the above-mentioned languages, we have the following proven relationships [18]:

$$\mathtt{PDAG} \prec \mathtt{cd\text{-}PDAG} \preceq \mathtt{d\text{-}DNNF} \begin{cases} \prec \mathtt{FBDD} \prec \mathtt{OBDD} \\ \prec \mathtt{d\text{-}DNF} \end{cases}.$$

For the particular application of this paper, it is important to note that `cd-PDAG`, which supports efficient probability computations (see next subsection), is more succinct than both `OBDD` and `d-DNF` (sum of disjoint products). Examples of Boolean functions with polynomial representations in `cd-PDAG` but exponential representations in `d-DNF` are the parity functions (odd or even) or k-out-of-n structures.

[19] For a more comprehensive overview we refer to [37].

Computing Probabilities. The above discussed properties decomposability and determinism are sufficient for a language to offer efficient (i.e. polynomial-time) probability computations. Let $P(x = 1)$ denote the given marginal probability of a variable $x \in V$ being true. If we assume the Boolean variables in V to be mutually independent, and if φ is a cd-PDAG, then the probability $P(f_\varphi) = P(\varphi)$ of the Boolean function f_φ can be computed by the following recursive procedure:

$$P(\varphi) = \begin{cases} \prod_i P(\beta_i) & \text{if } \varphi \text{ is a } \wedge\text{-node with children } \beta_i, \\ \sum_i P(\beta_i) & \text{if } \varphi \text{ is a } \vee\text{-node with children } \beta_i, \\ 1 - P(\beta) & \text{if } \varphi \text{ is a } \neg\text{-node with child } \beta, \\ P(x = 1) & \text{if } \varphi \text{ is a } \bigcirc\text{-node labeled with } x \in V, \\ 1 & \text{if } \varphi \text{ is a } \bigcirc\text{-node labeled with } \top, \\ 0 & \text{if } \varphi \text{ is a } \bigcirc\text{-node labeled with } \bot. \end{cases} \tag{13}$$

In other words, decomposability and determinism allow to replace $\wedge$-nodes and $\vee$-nodes by products and sums, respectively. Within the language `cd-PDAG`, probability computations are thus possible in time linear to the size of a given cd-PDAG, presumed that addition and multiplication require constant time. Given the above succinctness relationships, it is clear that `cd-PDAG` is the most suitable language for probability computations. This motivates the use of cd-PDAGs for the purposes of reliability analysis and diagnostics.
Another important advantage of cd-PDAGs is the flexibility to obtain their negations just by adding a $\neg$-node on top (or remove $\neg$-nodes from the top). In the context of this paper, this feature allows us to easily switch between the the structure function and its complement, i.e. between the success space and the failure space, respectively.

3.2 Modular System Reliability

For the reliability analysis of a modular system $\Sigma = (\{\Sigma_0, \Sigma_1, \ldots, \Sigma_m\}, f)$, we assume each local structure function $h_i(a)$ ssociated with the corresponding subsystem Σ_i to be represented by a local cd-PDAG ψ_i. The descendants $\mathsf{succ}(\Sigma_i)$ of Σ_i are the leaf nodes of ψ_i. Starting with the top-level cd-PDAG ψ_0, the tree structure now allows to replace in every ψ_i the leaves which are labeled with a module by the cd-PDAG of the corresponding local structure function. This is continued recursively until the leaves of the organizing tree (i.e. components) are reached. The top-down procedure of successively plugging toghether corresponding local cd-PDAGs leads

to the non-local cd-PDAG representing the structure function of the entire system. Note that both decomposability and determinism are preserved during this process. To formalize this idea, we define for each subsystem Σ_i

$$\varphi(\Sigma_i) = \begin{cases} \text{leaf node labeled with } x_i & \text{if } \Sigma_i = c_i \quad \text{(component)} \\ \psi_i(\varphi(\Sigma_{i_1}), \ldots, \varphi(\Sigma_{i_k})) & \text{if } \Sigma_i = \Sigma \quad \text{(module)} \end{cases} \tag{14}$$

to be its non-local (or global) cd-PDAG such that $\text{succ}(\Sigma_i) = \{\Sigma_{i_1}, \ldots, \Sigma_{i_k}\}$. If Σ_i is a module (second case), $\varphi(\Sigma_i)$ is obtained from the local cd-PDAG ψ_i by replacing each of its leaf nodes Σ_{i_j} by the corresponding cd-PDAG $\varphi(\Sigma_{i_j})$ for all $j \in \{1, \ldots, k\}$. Hence $\varphi(\Sigma_i)$ represents the non-local cd-PDAG of the whole subsystem rooted at module Σ_i. In this way we obtain $\varphi(\Sigma_0)$ which represents the global structure function $f(\mathbf{x})$ of the top-level module Σ_0, which in turn corresponds to the structure function of the entire system.

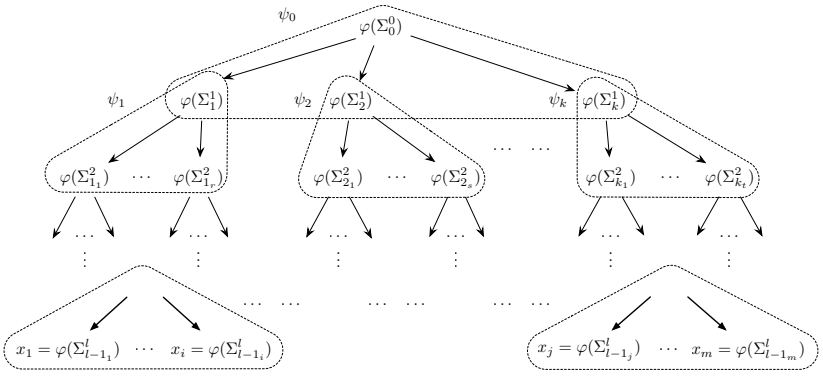


Figure 8: The hierarchy of local cd-PDAGs ψ_i and the corresponding non-local cd-PDAGs $\varphi(\Sigma_i)$ according to the system's organizing tree.

This leads to a hierarchical arrangement of local cd-PDAGs over several levels. To illustrate this, we label additionally each subsystem Σ_i and its corresponding local cd-PDAG ψ_i by a superscript representing the level number $n \in \{0, 1, \ldots, l\}$. Starting at level 0, we have the top-level local cd-PDAG ψ_0^0, then its children on level 1 denoted by $\psi_1^1, \ldots, \psi_k^1$, the children's children $\psi_1^2, \ldots, \psi_t^2$ on level 2, and so on, down to the leaf nodes $x_1, \ldots, x_m$ on the lowest level l. Figure 8 depicts this hierarchy of local cd-PDAGs for the entire modular system.

Given the component reliabilities and a cd-PDAG representation of the structure function, we are able to compute the system reliability by recursively propagating the intermediate module probabilities $P(\varphi(\Sigma_i))$ computed according to (13) along the modular hierarchy up to the root, at which we finally obtain the overall reliability $Rel(\Sigma) = P(\varphi(\Sigma_0))$ of the modular system $\Sigma = (\mathcal{S}, r, f)$.

Example 6. To illustrate the PDAG-based reliability evaluation of a modular system, recall the extended alp-transit system from Example 3. Figure 9 depicts the local cd-PDAGs representing the corresponding local structure functions h_A, h_U, h_P, h_R, and h_T.

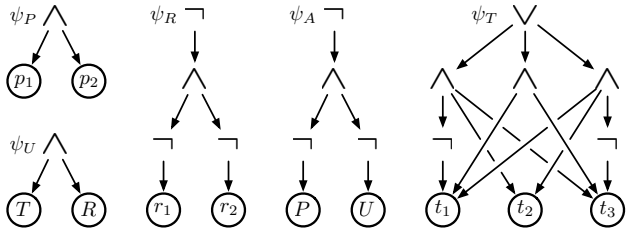


Figure 9: The cd-PDAGs of the local structure functions in the alp-transit example.

The global cd-PDAG $\varphi(A)$ is shown in Fig. 10 and is constructed by plugging together the local cd-PDAGs in the following manner: $\varphi(A)$ is obtained from ψ_A by replacing its leaves labeled with P and U by the respective non-local cd-PDAGs $\varphi(P)$ and $\varphi(U)$. $\varphi(U)$ is obtained in turn from ψ_U by replacing the leaves labeled with T and R by $\varphi(T)$ and $\varphi(R)$, respectively. This process continues recursively until all the leaf-node labels represent Boolean variables of atomic components.
For the reliability analysis we assume the following success probabilities assigned to components:

$$r(p_1) = 0.6,\ r(p_2) = 0.7,\ r(t_1) = r(t_3) = 0.8,\ r(t_2) = 0.9,\ r(r_1) = r(r_2) = 0.9.$$

Given these component reliabilities and the global cd-PDAG $\varphi(A)$, the calculation of the overall reliability of the alp-transit system happens by computing the probability of $\varphi(A)$:

$$Rel(\Sigma_{alptransit}) = P(A = 1) = P(\varphi(A)) = 0.9529.$$

The computation is illustrated in Fig. 10, where the (exact) probability values are restricted to four decimal places.

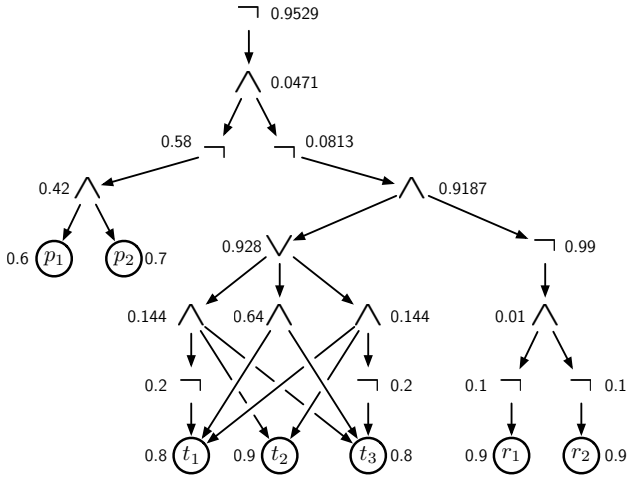


Figure 10: The cd-PDAG $\varphi(A)$ of the entire alp-transit system and the computation of its probability.

3.3 Network Reliability

Based on the formal description of reliability networks presented in Subsection 2.4, we discuss in the following the evaluation of network reliability. As for modular systems, it is based on a PDAG representation of the structure function which shall be called *reliability pdag* in the sequel. This subsection is divided into two major parts: Subsection 3.3.1 treats the problem of efficiently generating the reliability pdag for a previously specified connectivity measure. In Subsection 3.3.2 we explain how the resulting reliability pdag must be transformed to assure that the subsequent probability (reliability) computation becomes efficient. It will be of primary concern to compute the s,t-connectivity measure, for whose structure function the appropriate reliability pdag needs to be found. The methods we propose here create the reliability pdag directly from the network without relying on intermediate representations. For this, we consider two distinct approaches. The first method generates the reliability pdags for all possible source-terminal pairs in the network. The second provides a more general representation with a specified source node. From this, the reliability pdags for different connectivity measures can be derived. Finally, we present a modified version of the second method specifically designed for the case of acyclic networks.

3.3.1 Generating the Structure Function

Algebraic Preliminaries. To establish the necessary formal ground for the subsequent methods, let us introduce the class of Boolean functions of arity at most $n \geq 0$, which are defined as follows:

$$f : \{0,1\}^{\leq n} \to \{0,1\}.$$

The set of all such functions is denoted by $\mathcal{B}_{\leq n}$. Note that the 0-ary (constant) Boolean functions 1 and 0 are also included in this set. Further assume two binary operations $\vee$ and $\wedge$ over the set $\mathcal{B}_{\leq n}$. When interpreting 1 and 0 as logical truth values *true* and *false*, respectively, we can view $\vee$ as logical disjunction and $\wedge$ as logical conjunction. Then the set $\mathcal{B}_{\leq n}$ forms a semiring with addition $\vee$ and multiplication $\wedge$, as well as zero element 0 and unit element 1. We refer to the resulting structure $S_{\mathcal{B}} = (\mathcal{B}_{\leq n}, \vee, \wedge, 1, 0)$ as the *semiring of Boolean functions*[20].

Consider further a network $N = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes such that $|\mathcal{V}| = n$ and $\mathcal{E}$ is the set of all directed edges. With each $e_{ij} \in \mathcal{E}$ we associate a corresponding Boolean variable x_{ij} such that $x_{ij} = 1$ if edge e_{ij} is operative and $x_{ij} = 0$ if e_{ij} is down. Given the network N, its *connectivity matrix* is defined by matrix $A = (f_{ij})_{n \times n}$ over the semiring $S_{\mathcal{B}}$, i.e. with entries f_{ij} in $\mathcal{B}_{\leq n}$. Initially, $f_{ij} = x_{ij}$ if the corresponding edge exists, i.e. if $e_{ij} \in \mathcal{E}$, and $f_{ij} = 0$ otherwise. By appropriate manipulation of the connectivity matrix, the following algorithms generate the structure function with respect to the desired connectivity measure. The reliability pdag relative to this structure function will be usually denoted by φ.

Connectivity of All Pairs. In some situations the connectivity between all node pairs of a network can be of interest. This may be the case in the context of routing problems for instance, where messages must be routed reliably between all pairs of network sites. The following method makes use of dynamic programming and generates efficiently the reliability pdags for the s,t-connectivity — simultaneously for all possible source-terminal pairs in the network.

[20] This semiring satisfies a number of properties which can be easily verified: it is commutative, idempotent, bounded (1 being the maximum and 0 the minimum), and ordered (its natural order is monotonic with respect to both $\wedge$ and $\vee$).

Algorithm 1: WFK Algorithm for All-Pairs Connectivity.

Input: connectivity matrix $A = (f_{ij})_{n\times n}$, $f_{ij} \in \mathcal{B}_{\leq n}$.
Output: $A^{(n)}$, the matrix of structure functions $f_{ij}^{(n)}$ for $conn_{i,j}$ for all $1 \leq i, j \leq n$.

```
1  begin
2      A^(0) ← A
3      for k from 1 to n do
4          foreach i, j ∈ {1, ..., n} do
5              if i, j ≠ k then
6                  [A^(k)]_ij ← [A^(k−1)]_ij ∨ ( [A^(k−1)]_ik ∧ [A^(k−1)]_kj )
7              end
8          end
9      end
10     return A^(n)
11 end
```

By traversing successively each node $v_k \in \mathcal{V}$ of the network (the outer loop, line 3), the procedure shown in Alg. 1 constructs all possible "paths" going through this node by considering all transitive connections between nodes v_i and v_j via v_k, the first two being different from v_k (the inner loops, lines 4-6). $A^{(k)}$ denotes the matrix A after the k-th iteration of the outer loop, and $[A^{(k)}]_{ij}$ denotes its i, j-entry. In view of the reliability computation discussed in the next subsection, we assume at each stage k of the algorithm the structure functions $f_{ij}^{(k)} \in \mathcal{B}_{\leq n}$ to be represented by corresponding reliability pdags $\varphi_{ij}^{(k)}$, since all reliability computations will be carried out later as PDAG-operations. Furthermore, by manipulating reliability pdags rather than other structures, we ensure the space requirement of the algorithm to remain in moderate bounds. This applies of course to all algorithms presented in this subsection. The following theorem confirms that Alg. 1 indeed computes the desired structure functions:

Theorem 3.1. At the end of Algorithm 1, each entry $f_{ij}^{(n)}$ of the matrix $A^{(n)}$ corresponds to the structure function for the v_i, v_j-connectivity measure for all source-terminal pairs v_i and v_j in the network. Proof: see appendix.

The algorithm is inspired by various similar solution schemes, starting with Kleene's proof that any regular language can be represented by a regular expression, Warshall's algorithm for computing the transitive closure of a Boolean matrix [38], and Floyd's adaptation for finding shortest distance paths [39]. Since these algorithms all follow the same general scheme,

they can be viewed in fact as special cases of a single, more general algorithm, which is sometimes refered to as *generalized* WFK (Warshall-Floyd-Kleene) algorithm [40]. Several authors have recognized this generic pattern and proposed axiomatized frameworks for its description, see the comment at the end of this subsection. Also, the Gauss-Jordan method from linear algebra can be adapted accordingly to this more general form.

Example 7. To illustrate the WFK algorithm for all-pairs connectivity, consider the network $N = (\mathcal{V}, \mathcal{E})$ from Fig. 6a, where $\mathcal{V} = \{v_1, v_2, v_3, v_4\}$ and $\mathcal{E} = \{a, b, c, d, e, f\}$. The application of Alg. 1 to the network's connectivity matrix $A = A^{(0)}$ results in the sequence of matrices $A^{(1)}$ to $A^{(4)}$ shown below. For a more compact illustration, the Boolean functions $f_{ij}^{(k)}$ for $k = 1, \ldots, 4$ are represented by nested Boolean formulae, conjunction $x \wedge y$ is realized by juxtaposition xy, and disjunction $x \vee y$ is abbreviated by means of a comma: x, y.

$$A^{(0)} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & a & e & 0 \\ 0 & 0 & c & b \\ 0 & d & 0 & f \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \quad A^{(2)} = \begin{bmatrix} 0 & a & e, ac & ab \\ 0 & 0 & c & b \\ 0 & d & cd & f, bd \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A^{(1)} = \begin{bmatrix} 0 & a & e & 0 \\ 0 & 0 & c & b \\ 0 & d & 0 & f \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad A^{(3)} = \begin{bmatrix} 0 & a, [d(e, ac)] & e, ac & ab, [(e, ac)(f, bd)] \\ 0 & cd & c & b, [c(f, bd)] \\ 0 & d & cd & f, bd \\ 0 & 0 & 0 & 0 \end{bmatrix} = A^{(4)}$$

For example, inspection of the $1, 4$-entry of matrix $A^{(4)}$ reveals the structure function representation for the v_1, v_4-connectivity. By expanding the corresponding Boolean formula using the distributivity law and applying appropriate simplifications[21], we obtain exactly the four minpaths listed in Example 4: ab, ef, acf, and edb. But here our goal is to retain the structure of the nested formula by representing the Boolean function $f_{1,4}^{(4)}$ by a corresponding reliability pdag φ_{v_1,v_4}. A possible PDAG is shown in Fig. 11b; the meaning of the grey nodes is later explained in Subsection 3.3.2. In

[21] The partial order of the semiring $S_{\mathcal{B}}$ (induced by the idempotency axiom) causes every BF g to be absorbed by a BF f which is "greater" w.r.t. this order: $g \preccurlyeq f \iff g \wedge f = g \iff g \vee f = f$.

the next subsection we present an alternative method for generating an equivalent reliability pdag.

Remark To adapt the procedure from Alg. 1 to a generic form, we basically have to replace the $\vee$ and $\wedge$ operations by general semiring addition $+$ and multiplication $\times$, respectively, and to consider the initial matrix A of values over an arbitrary closed semiring [40]. By an appropriate choice of the operations $+$ and $\times$, a large variety of path problems can be solved, including the calculation of the transitive closure of a binary relation in a graph [41, 38], shortest paths, maximum capacity paths, maximum reliability paths, and many others. Hence, Algorithm 1 clearly fits into this general solution scheme using the semiring of Boolean functions. The calculation of the matrix A^* of such optimal values (sometimes called the *closure* of A) is commonly refered to as the *algebraic path problem* in the literature. Several authors have established corresponding algebraic frameworks with varying properties for the solution of path problems in graphs and networks [40, 42, 43]. Possible applications of the resulting general solution schemes go even beyond path problems and include the computation of the inverse of a real matrix, or obtaining the regular language accepted by a finite automaton. See [42] for a survey of interesting applications. A similar algebraic approach has been developed by [15] and applied to the computation of network reliability.

Generic Single-Source Connectivity. Instead of specifying from the beginning the desired source and terminal nodes as in the above method, we delay the choice of the terminal nodes to a later moment and fix at first only the source node. The idea is to generate a special structure function representation that contains enough information to allow every node to be a potential terminal node. This enables us afterwards to determine several terminal nodes at the same time. This is one of the advantages of this method: it allows more flexibility with respect to conventional methods that start with a fixed choice of terminal nodes.
Inspired by various elimination algorithms from linear algebra like Gauss and Gauss-Jordan elimination, we set up an algorithm which successively eliminates all network nodes except one that has been dedicated as source node at the beginning of the procedure. The result is a generic expression that can be instantiated accordingly to produce the structure function with respect to the source-to-any-terminal connectivity. The structure function for the s,t-connectivity is obtained as a special case.
The connectivity matrix A of a given network is defined as before, except that we consider additionally for each node v_i a Boolean variable λ_{v_i} called

terminal selector. The value of the terminal selector determines whether the corresponding node belongs to the terminal set or not: $v_i \in K$ if $\lambda_{v_i} = 1$, and $v_i \notin K$ if $\lambda_{v_i} = 0$. In summary, the initial matrix $A = (f_{ij})_{n \times n}$ is newly defined as follows:

$$f_{ij} = \begin{cases} x_{ij} & \text{if } e_{ij} \in \mathcal{E},\ i \neq j, \\ \lambda_{v_i} & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

The role of the terminal selectors becomes more interesting when we later look at the instantiation of the structure function representation obtained by Alg. 2.

The following algorithm successively eliminates nodes from the network similar to the Gauss algorithm. The initial elimination order is not important for the correctness of the algorithm, but may affect the shape of the reliability pdag resulting at the end. In this respect it turns out to be advantageous to order the nodes in the connectivity matrix in a manner such that the non-zero entries are concentrated preferably in the upper right half. This is achieved by arranging the network nodes in increasing distance (number of intermediate edges) from the dedicated source such that their corresponding matrix entries have increasing indices in the matrix. In this way the source node's entry has always the lowest matrix indices, and ties are resolved randomly. The reverse order serves then as elimination sequence.[22]

[22]This ordering heuristic is called Largest Distance First (LDF) in [36]. In an acyclic network it leads to an upper triangular connectivity matrix.

Algorithm 2: Elimination Algorithm for Generic Single-Source Connectivity.

Input: connectivity matrix $A = (f_{ij})_{n\times n}$, $f_{ij} \in \mathcal{B}_{\leq n}$.
Output: $[A^{(1)}]_{11}$, generic structure function for single-source connectivity.

```
1  begin
2      A^(n) ← A
3      for k from n downto 2 do
4          for i from k − 1 downto 1 do
5              if [A^(k)]_ik ≠ 0 then
6                  [A^(k−1)]_ii ← [A^(k)]_ii ∨ ( [A^(k)]_ik ∧ [A^(k)]_kk )
7                  for j from 1 to k − 1 do
8                      if [A^(k)]_kj ≠ 0 and j ≠ i then
9                          [A^(k−1)]_ij ← [A^(k)]_ij ∨ ( [A^(k)]_ik ∧ [A^(k)]_kj )
10                     end
11                 end
12             end
13         end
14     end
15     return [A^(1)]_11
16 end
```

To see how Alg. 2 works, let us view the elimination of a node from the connectivity matrix as the removal of the corresponding vertex from the network. When a vertex is eliminated from the network, we must make up somehow for the paths that are affected by this removal. This is done in two ways: First, when node v_k is eliminated its reachability information $[A^{(*)}]_{kk}$ is passed to all nodes v_i having outgoing edges e_{ik}, thus updating their respective reachability information $[A^{(*)}]_{ii}$ (line 6). In Example 8, through the elimination of v_4 two entries are updated: λ_2 becomes $\lambda_2 \vee (b \wedge \lambda_4)$, and λ_3 becomes $\lambda_3 \vee (f \wedge \lambda_4)$. Second, when v_k has outgoing edges e_{kj} in the current connectivity matrix, we must make up for all transitive connections from nodes v_i to v_j via v_k (cf. Alg. 1). This means that every pair of edges e_{ik} and e_{kj} with respective labels f_{ik} and f_{kj} is replaced by a new single edge e_{ij} with label $f_{ik} \wedge f_{kj}$. This information is added to the old label of edge e_{ij}, if there was such previously in the network (line 9). Consider again Example 8 and the elimination of v_3: after this, the $1,2$-entry of the matrix gets the new label $a \vee (d \wedge f)$. As in Alg. 1, all matrix entries f_{ij} are represented at any time by corresponding reliability pdags φ_{ij}.

At the end of Alg. 2 we obtain what we call a generic reliability pdag, which

does not represent yet a real structure function. We denote this reliability pdag by φ_s where $s = v_1$ is the chosen source node. This is where the terminal selectors come into play: their values determine the exact meaning of the result. To get the structure function for a specific problem with respect to a given set $K = \{t_1, \ldots, t_k\} \subseteq \mathcal{V} \setminus \{s\}$ of terminal nodes, we must instantiate the terminal selectors within φ_s accordingly. In the PDAG context this amounts to *conditioning* φ_s with respect to the terminal selectors as follows: all variables λ_{t_i} for all $t_i \in K$ are set to 1, whereas variables λ_{v_i} for all $v_i \in \mathcal{V} \setminus K$ are set to 0. Note that this is a linear time operation in the size of φ_s. The resulting reliability pdag represents the structure function for $conn_{\exists K}$ and is denoted by $\varphi_{s,\exists K}$. For $K = \{t\}$ we get the reliability pdag $\varphi_{s,t}$ for $conn_{s,t}$ as special case. The following theorem confirms this result:

Theorem 3.2. Let be φ_{v_1} the reliability pdag resulting from Alg. 2, i.e. after eliminating variables $v_2, \ldots, v_n$. Further assume the following instantiation of the terminal selectors: $\lambda_{v_n} = 1$ and $\lambda_{v_i} = 0$ for $1 \leq i \leq n-1$. After appropriately conditioning φ_{v_1} with respect to this instantiation, the resulting pdag φ_{v_1,v_n} represents the structure function for $conn_{v_1,v_n}$. Proof: see appendix.

To obtain the structure function for $conn_{\forall K}$, we simply conjoin the reliability pdags of the respective $conn_{\{s,t_i\}}$ problems for all terminal nodes $t_i \in K$. Hence, after appropriate instantiation of φ_s we can construct the corresponding reliability pdag $\varphi_{s,\forall K} = \varphi_{s,t_1} \wedge \ldots \wedge \varphi_{s,t_k}$. Similarly, for $K = \mathcal{V} \setminus \{s\}$ we obtain the reliability pdag $\varphi_{s,\mathcal{V}}$ for $conn_{\mathcal{V}}$.

Example 8. Consider again the four-node network from Fig. 6a and suppose v_1 has been chosen as source node. Alg. 2 sequentially eliminates nodes v_4, v_3, v_2 from the network (in this order), which is illustrated by the sequence of matrices $A = A^{(4)}, A^{(3)}, A^{(2)}, A^{(1)}$ shown below. The matrices are depicted in decreasing size, since at each step k of the algorithm (outer loop), only entries with indices $\leq k$ are considered. Note that the notational conventions from Example 7 apply here as well, and the λ_{v_i} are

abbreviated by λ_i.

$$A^{(4)} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} \lambda_1 & a & e & 0 \\ 0 & \lambda_2 & c & b \\ 0 & d & \lambda_3 & f \\ 0 & 0 & 0 & \lambda_4 \end{bmatrix} \end{array} \quad A^{(3)} = \begin{bmatrix} \lambda_1 & a & e \\ 0 & \lambda_2, b\lambda_4 & c \\ 0 & d & \lambda_3, f\lambda_4 \end{bmatrix}$$

$$A^{(2)} = \begin{bmatrix} \lambda_1, [e(\lambda_3, f\lambda_4)] & a, ed \\ 0 & (\lambda_2, b\lambda_4), c(\lambda_3, f\lambda_4) \end{bmatrix}$$

$$A^{(1)} = \Big[[\lambda_1, e(\lambda_3, f\lambda_4)], [(a, ed)\big((\lambda_2, b\lambda_4), c(\lambda_3, f\lambda_4)\big)] \Big]$$

At the end of the algorithm we are interested in the upper left entry of matrix $A^{(1)}$: the generic structure function representation with source node v_1. The reliability pdag φ_{v_1} representing this function is depicted in Fig. 11a. In order to obtain the structure function representation for the v_1, v_4-connectivity, we instantiate the terminal selectors appropriately within the Boolean formula of $f_{11}^{(1)}$ and get as result $ef, [(a, ed)(b, cf)]$ (cf. for this Example 7). In terms of φ_{v_1} this amounts to setting the terminal selectors within φ_{v_1} as follows: $\lambda_4 \equiv 1$ and $\lambda_1 \equiv \lambda_2 \equiv \lambda_3 \equiv 0$. After all possible simplifications, this results in the instantiated reliability pdag φ_{v_1,v_4} shown in Fig. 11b. Note that the depicted reliability pdag is not unique; several logically equivalent PDAGs are possible. For instance, the reliability pdag for the structure function obtained in Example 7 would have another shape.

Acyclic Networks. In practice, networks are often cycle-free and relatively sparse, resulting in correspondingly few edges. It would be thus desirable to take account of this observation by devising an algorithm which relies solely on the number of edges as critical parameter. In the particular case of a directed, acyclic network there is always an ordering of nodes (which may be not unique) such that the network's connectivity matrix becomes upper triangular, i.e. if $e_{ij} \in \mathcal{E}$, then $i < j$.[23] Taking a closer look at Alg. 2, the whole third loop can be left out in this case. Furthermore, the first two loops can be merged to a single loop which iterates through edges only. The edge set is supposed to be ordered by decreasing column index (first priority), and by decreasing

[23] Such node ordering is sometimes also called *topological order*.

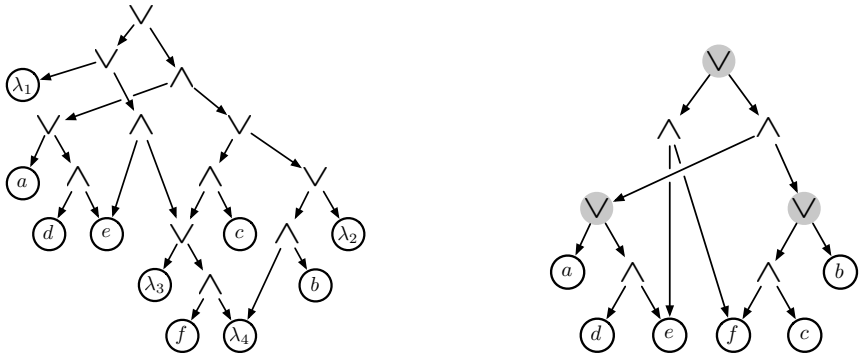
row index (second priority). We denote this order by π and the set of edges ordered according to π is refered to as $\mathcal{E}_\pi$. The adapted procedure results in Algorithm 3 shown below and takes besides the connectivity matrix A, still defined by (15), also the set $\mathcal{E}_\pi$ as input and loops through it in the specified order, so that the elimination of nodes happens the same way as it does in Alg. 2. Note that the essential part of Alg. 3 on line 3 corresponds exactly to line 6 in Alg. 2. Since this new algorithm is a special case of Alg. 2, the assertion of Theorem 3.2 applies here as well.

Algorithm 3: Elimination Algorithm for Generic Single-Source Connectivity (Acyclic Version).

Input: 1. upper triangular connectivity matrix $A = (f_{ij})_{n\times n}$, $f_{ij} \in \mathcal{B}_{\leq n}$, and 2. $\mathcal{E}_\pi$.

Output: $[\,A\,]_{11}$, generic structure function for single-source connectivity.

```
1 begin
2 |   foreach e_ij ∈ E_π in order π do
3 |   |   [A]_ii ← [A]_ii ∨ ( [A]_ij ∧ [A]_jj )
4 |   end
5 |   return [A]_11
6 end
```



(a) (b)

Figure 11: (a) The generic reliability pdag φ_{v_1} from Example 8 for the network in Fig. 6a. (b) The specific reliability pdag φ_{v_1,v_4} after instantiating the terminal selectors within φ_{v_1}: $\lambda_4 \equiv 1$, $\lambda_1 \equiv \lambda_2 \equiv \lambda_3 \equiv 0$. The shaded $\vee$-nodes are not deterministic.

Notes on Complexity. Assuming that the basic operations $\vee$ and $\wedge$ require constant time, the WFK algorithm for all-pairs connectivity (Alg. 1)

has a complexity of $\Theta(n^3)$, which is an asymptotic tight bound. A non-trivial lower bound is not known for this case. The algorithm can be designed more efficiently by means of parallelization (e.g. by distributing the computations over a mesh of connected processors), see [44], and by using further dynamic programming techniques [45]. In any case, the current literature on these topics is rich and offers many possibilities for optimization.
The algorithm for generic single-source connectivity (Alg. 2) has an asymptotic (not tight) upper bound complexity of $O(n^3)$. It requires at most $n(n-1)(2n-1)\,/\,6$ $\vee$-operations and the same number of $\wedge$-operations. However, this is under the assumption that maximum possible fill-in occurs, i.e. the connectivity matrix is assumed to be complete. In practice, the effective number of operations is much smaller since they are only performed in case of non-zero entries. So the theoretical worst-case complexity $O(n^3)$ does not accurately reflect the relative efficiency of the algorithm when applied to real problems.
In the case of acyclic (and possibly sparse) networks Algorithm 3 exploits the fact that the number of edges reflects the network size much more accurately than the number of nodes does. The running time is then $\Theta(m)$, thus linear in the number of edges m. In terms of the number of nodes, this corresponds to the upper bound complexity $O(n^2)$.

3.3.2 Computing Network Reliability

Assume we have generated the reliability pdag φ for one of the above described network reliability problems. Based on this reliability pdag, we can tackle the computation of network reliability. For this, recall that φ must meet the decomposability and determinism properties discussed in Subsection 3.1.2 to allow efficient probability computation. Unfortunately, this condition is not satisfied in general at this stage. Thus prior to reliability computation, we must transform the reliability pdag φ into a logically equivalent cd-PDAG. The whole process can be split into three consecutive steps:

1. Transformation of the reliability pdag φ into a logically equivalent deterministic PDAG (d-PDAG), denoted by φ^{d}. This operation is linear in the size of the PDAG.

2. Transformation of the d-PDAG φ^{d} resulting from the previous operation into a logically equivalent cd-PDAG denoted by φ^{cd}. This operation is computationally hard in general and may result in an expo-

nentially larger representation of the new reliability pdag relative to φ^{d}.

3. Exact probability computation based on the transformed reliability pdag φ^{cd}. This operation is linear in the size of the cd-PDAG.

Before describing in more detail the above-mentioned stages, let us remark that we can perform at this point alternatively an approximate reliability computation by directly sampling from φ, making any further transformations needless. Such a Monte-Carlo based approximation may be suitable in case of very large networks, where the main issue is performance rather than accuracy of the result.[24]

Step 1: Making the PDAG deterministic. To make the reliability pdag φ deterministic, we traverse φ and replace each $\vee$-node $\varphi_1 \vee \varphi_2$ by $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$, where φ_1 and φ_2 are arbitrary sub-PDAGs. With this, the determinism problem is solved since all $\vee$-nodes vanish, but the new $\wedge$-nodes are still not necessarily decomposable (this issue is approached in the next step). For the following algorithms, we assume to dispose of the predefined (self-explanatory) predicates with respect to an arbitrary PDAG α: isPdagLeaf(α), isPdagOr(α), and isPdagAnd(α). The transformation can be now cast into the following recursive procedure:

Algorithm 4: PDAG to d-PDAG Transformation.

```
1  procedure makeDeterministic(φ)
2  begin
3      if isPdagLeaf(φ) then
4          break
5      else
6          if isPdagOr(φ) then
7              φ ← ¬(⋀_{i=1}^{m} ¬ψ_i) where {ψ_1, ..., ψ_m} = children(φ)
8          end
9          foreach ψ_i ∈ children(φ) do
10             makeDeterministic(ψ_i)
11         end
12     end
13 end
```

[24] This strategy is not specifically related to networks and may be applied in any case in which the structure function is represented by a PDAG.

This algorithm runs in $\Theta(\#(\varphi)+|\varphi|)$ time, assuming that the node replacement requires constant time. As an example, consider the reliability pdag φ_{v_1,v_4} depicted in Fig. 11b from Example 8: it has three $\vee$-nodes which are not deterministic (indicated with shaded circles). Applying Alg. 4 to φ_{v_1,v_4} results in the deterministic PDAG $\varphi^{\mathrm{d}}_{v_1,v_4}$ depicted in Fig. 12a.

Step 2: Making the d-PDAG decomposable. To make the d-PDAG φ^{d} resulting from Alg. 4 decomposable, we traverse this time φ^{d} and check for each $\wedge$-node $\varphi_1 \wedge \varphi_2$ whether decomposability holds. Whenever φ_1 and φ_2 have a common sub-PDAG ψ, i.e. if $vars(\varphi_1) \cap vars(\varphi_2) \neq \emptyset$, decomposability is not satisfied. In such a case, we decompose $\varphi_1 \wedge \varphi_2$ by means of Shannon's expansion with respect to some variable $x \in vars(\psi)$, which includes conditioning $\varphi_1 \wedge \varphi_2$ on x and $\neg x$.[25] This procedure must be recursively applied to each newly created $\wedge$-node until decomposability holds for all $\wedge$-nodes, resulting in the cd-PDAG φ^{cd}. Computationally, this is the hardest task of the entire transformation process and requires possibly exponential time. The pseudo-code in Algorithm 5 shows a recursive procedure which takes a deterministic PDAG as input and transforms it into a logically equivalent cd-PDAG.

To illustrate this transformation by an example, consider the d-PDAG $\varphi^{\mathrm{d}}_{v_1,v_4}$ from Fig. 12a obtained after step 1. The shaded $\wedge$-node is not decomposable since its children share the common variables e and f. By choosing for instance e as pivot variable, we replace this $\wedge$-node by Shannon's expansion with respect to e which yields the new PDAG shown in Fig. 12b. This one is still not decomposable because the children of the new shaded $\wedge$-node have the variable f in common. After performing a similar replacement relative to f, we obtain the decomposable PDAG $\varphi^{\mathrm{cd}}_{v_1,v_4}$ shown in Fig. 12c which allows efficient probability calculation.

[25] The choice of the variable x is arbitrary and can be taken according to some heuristic. In fact, the expansion can be also performed with respect to the whole sub-PDAG ψ.

Algorithm 5: d-PDAG to cd-PDAG Transformation.

```
1  procedure makeDecomposable(φ)   // φ is assumed to be deterministic
2  begin
3      if isPdagLeaf(φ) then
4          break
5      else
6          if isPdagAnd(φ) then
7              I ← ⋂_{ψ∈children(φ)} vars(ψ)
8              if I ≠ ∅ then
9                  choose x ∈ I according to some heuristic
10                 φ_x ← φ | x        // replace each occurence of x in φ by 1
11                 φ_¬x ← φ | ¬x     // replace each occurence of x in φ by 0
12                 makeDecomposable(φ_x)
13                 makeDecomposable(φ_¬x)
14                 φ ← (φ_x ∧ x) ∨ (φ_¬x ∧ ¬x)    // Shannon's expansion w.r.t. x
15             else
16                 foreach ψ ∈ children(φ) do
17                     makeDecomposable(ψ)
18                 end
19             end
20         else
21             foreach ψ ∈ children(φ) do
22                 makeDecomposable(ψ)
23             end
24         end
25     end
26 end
```

Several aspects are left out in Algorithm 5 which are relevant though in a real implementation. For instance, one would assume for each sub-PDAG ψ of φ its variable set $vars(\psi)$ to be stored in a hash table to make the test on line 8 as efficient as possible. Furthermore, it would be probably more convenient to perform both steps 1 and 2 together in one traversal of the PDAG. But for the sake of clarity of the algorithms and to underline the fact that to guarantee decomposability is the hard part, we have decided to consider these transformations separately.

Step 3: Exact reliability computation. The last step of the evaluation process consists in computing the probability of the reliability pdag φ^{cd} resulting from step 2. This computation is exactly the same as for modular systems or hybrid systems which we have seen before. At this point we are

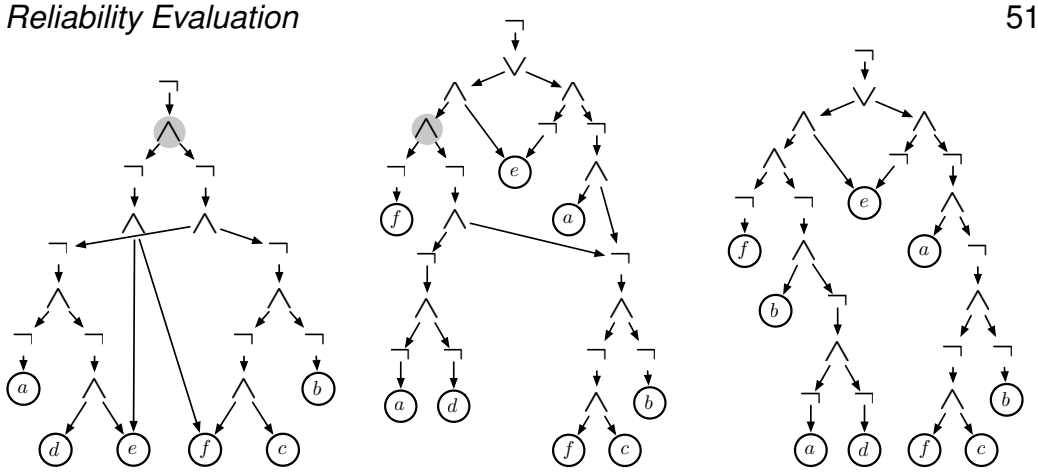


Figure 12: (a) The d-PDAG $\varphi^{\mathrm{d}}_{v_1,v_4}$ after making φ_{v_1,v_4} deterministic. The shaded $\wedge$-node is not decomposable. (b) The transformed d-PDAG $\varphi^{\mathrm{d}}_{v_1,v_4}$ after expanding the shaded $\wedge$-node from (a) relative to e. The new shaded $\wedge$-node is not decomposable. (c) The cd-PDAG $\varphi^{\mathrm{cd}}_{v_1,v_4}$ obtained from (b) after expanding its shaded $\wedge$-node relative to f.

still free to assign arbitrary probability values to the edges, or to perform sensitivity analysis if required.

To illustrate this last step, we consider once more our sample network example used so far. For the computation of the v_1, v_4-connectivity, we need to calculate the probability of the cd-PDAG $\varphi^{\mathrm{cd}}_{v_1,v_4}$ depicted in Fig. 12c. To simplify the calculation, suppose that the success probabilities are equal for all edges: $r(a) = r(b) = r(c) = r(d) = r(e) = r(f) = 0.9$. Now we can compute the probability of the given cd-PDAG (analogously to the computation shown in Fig. 10) by recursively propagating the intermediate probabilities in $\varphi^{\mathrm{cd}}_{v_1,v_4}$ up to the root at which the final probability is obtained. With the above edge reliabilities, we get the overall network reliability $Rel(N[\, conn_{v_1,v_4}\,]) = P(\varphi^{\mathrm{cd}}_{v_1,v_4}) = 0.97848$.

3.4 Hybrid System Reliability

The representation of structure functions by means of PDAGs allowed in the previous subsections to evaluate system reliability in a convenient and efficient way. Since modules and networks are the building blocks of hybrid systems, the methods used so far can be transferred to hybrid networks and hybrid systems.

Recall first the formal system description of hybrid networks and let the structure functions f_{V_i} and f_{E_j} associated with vertices and edges be rep-

resented by the corresponding cd-PDAGs $\vartheta(V_i)$ and $\varepsilon(E_j)$ which are specified as follows:

$$\vartheta(V_i) := \begin{cases} \text{leaf node labeled with } y_i & \text{if } V_i = v_i \text{ (atomic node)}, \\ \varphi(\Sigma_i) & \text{if } V_i = \Sigma_i \text{ (module)}, \\ \varphi(N_i[\,\mathscr{Q}_i\,]\ ) & \text{if } V_i = N_i[\,\mathscr{Q}_i\,] \text{ (network)}, \end{cases}$$

and similarly

$$\varepsilon(E_j) := \begin{cases} \text{leaf node labeled with } z_j & \text{if } E_j = e_j \text{ (atomic edge)}, \\ \varphi(\Sigma_j) & \text{if } E_j = \Sigma_j \text{ (module)}, \\ \varphi(N_j[\,\mathscr{Q}_j\,]\ ) & \text{if } E_j = N_j[\,\mathscr{Q}_j\,] \text{ (network)}, \end{cases}$$

Basically, the procedure to compute the reliability of a hybrid network is the same as for usual reliability networks as discussed in Subsection 3.3, except that here we must deal possibly with modular vertices and/or modular edges. Given a network $N[\,\mathscr{Q}\,] = (\mathcal{V}, \mathcal{E}, g)$ with a specified problem $\mathscr{Q}$, the goal is first to represent its structure function g by a reliability pdag $\varphi(N[\,\mathscr{Q}\,])$ where $\mathscr{Q}$ indicates the imposed reliability measure. If we assume only edges to fail, the leaf nodes of $\varphi(N[\,\mathscr{Q}\,])$ consist then of PDAGs $\varepsilon(E_j)$ only. In case of a simple (i.e. non-hybrid) network, we are left exclusively with leaf nodes labeled with the corresponding Boolean variables z_j. In the hybrid case however, we replace every PDAG $\varepsilon(E_j)$ which is associated with a modular edge by the corresponding cd-PDAG representation $\varphi(\Sigma_j)$ of the non-local structure function of the underlying modular (or hybrid) system Σ_j. To obtain this cd-PDAG we apply the strategy from Subsection 3.2. Such a generic scenario is shown schematically in Fig. 13. In either case, to compute the reliability of the network based on the reliability pdag $\varphi(N[\,\mathscr{Q}\,])$, we must first transform $\varphi(N[\,\mathscr{Q}\,])$ into the corresponding reliability cd-pdag $\varphi^{\mathrm{cd}}(N[\,\mathscr{Q}\,])$ as discussed in Subsection 3.3, and then evaluate it by calculating its probability $P(\varphi^{\mathrm{cd}}(N[\,\mathscr{Q}\,]))$.

The approach in case of a hybrid system $\Sigma = (\{\Sigma_0, \Sigma_1, \ldots, \Sigma_m\}, f)$ is a straightforward extension of the modular case from Subsection 3.2. We already know from the context of modular systems that the local structure functions associated with subsystems Σ_i are represented by local cd-PDAGs ψ_i. In case Σ_i comes in form of a network N_i, the discussion from the previous paragraph applies. Note that the reliability pdag $\varphi(N[\,\mathscr{Q}\,])$ must be then transformed into the cd-PDAG $\varphi^{\mathrm{cd}}(N[\,\mathscr{Q}\,])$. So the non-local

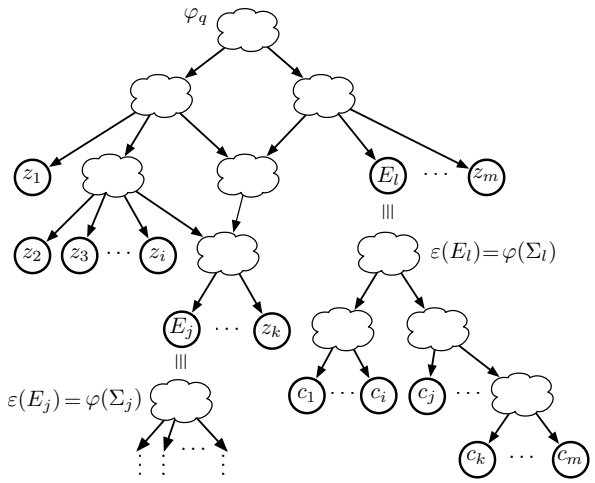


Figure 13: The reliability pdag $\varphi(N[\mathscr{Q}])$ of a hybrid network obtained for reliability measure $\mathscr{Q}$. The nodes labeled with modular edges E_j and E_l are replaced by the corresponding cd-PDAGs $\varphi(\Sigma_j)$ and $\varphi(\Sigma_l)$ associated with the respective underlying systems.

cd-PDAG φ associated with every subsystem Σ_i is now as follows:

$$\varphi(\Sigma_i) = \begin{cases} \text{leaf node labeled with } x_i & \text{if } \Sigma_i \text{ is a component } c_i, \\ \varphi^{\mathrm{cd}}(N_i[\,\mathscr{Q}_i\,]) & \text{if } \Sigma_i \text{ is a network } N_i[\,\mathscr{Q}_i\,] \ (i \neq 0), \\ \psi_i(\varphi(\Sigma_{i_1}), \ldots, \varphi(\Sigma_{i_k})) & \text{if } \Sigma_i \text{ is a module.} \end{cases} \tag{16}$$

Again, the reliability of the hybrid system — given prior probabilities of all atomic components, vertices, and edges — is then obtained by computing $Rel(\Sigma) = P(\varphi(\Sigma_0))$.

4 Diagnostics

The analysis presented in the previous sections corresponds to the classical approach of reliability: given the probabilities of operation of its components, compute the probability of operation of the whole system. In this section we tackle the system analysis from another direction: given that the system or some of its parts are observed to be down, find out which elements are most likely responsible for this defect. This task is fully in the spirit of the more general problem of diagnostics: based on observations about the state of operation of the entire system or some of its parts, the goal is to find the most probable diagnoses which explain the observed behaviour.

4.1 The Basic Setting

To solve this problem, we follow in this paper the approach of *Bayesian diagnostics*: we compute the posterior (or conditional) success probabilities of modules, components, and other subsystems given the failure or operation of the system or a subsystem thereof. The posterior probabilities indicate which subsystems are most likely the cause of the observed behaviour. For this we can use the results of reliability computations from the previous section. Repeating this process in an iterative manner allows to refine the diagnoses: by performing further tests specifically on the suspected modules or components, we can update their posterior probabilities. In this way we may identify the causes of the observed failures with an increasing certainty.

To demonstrate our approach on a more concrete basis, let us start with a simple setting. Consider for this a system $\Sigma = (\{\Sigma_0, c_1, \ldots, c_m\}, f)$ with top-level module Σ_0 and atomic components $c_1, \ldots, c_m$. Now assume that we observe a system failure, i.e. $f(\mathbf{x}) = 0$. The occurence of this event changes the *prior* success probabilities $P(x_i = 1)$ of the components c_i into corresponding *posterior* probabilities $P(x_i = 1 \,|\, f(\mathbf{x}) = 0)$. These conditional probabilities can be computed according to Bayes' theorem:

$$\begin{aligned} P(x_i = 1 \,|\, f(\mathbf{x}) = 0) &= \frac{P(\{x_i = 1\} \cap \{f(\mathbf{x}) = 0\})}{P(f(\mathbf{x}) = 0)} \\ &= \frac{P(\{x_i = 1\} \cap \{f(\mathbf{x}) = 0\})}{1 - P(f(\mathbf{x}) = 1).} \end{aligned} \tag{17}$$

The denumerator of this formula is almost given since the reliability computation is assumed to be accomplished, hence the unreliability is easily

obtained. Now the numerator remains to be calculated, which is in general more complicated. We show later in this section how to perform this computation in terms of PDAG manipulations. But let us first put the problem statement of diagnostics in a more general form. Consider for this an arbitrary system $\Sigma = (\{\Sigma_0, \Sigma_1, \ldots, \Sigma_m\}, r, f)$ as subject of diagnostics. We assume the structure function f to be represented by the global cd-PDAG $\varphi(\Sigma_0)$. According to the organizing tree of the system the cd-PDAG $\varphi(\Sigma_i)$ associated with each subsystem Σ_i is contained as a sub-PDAG in $\varphi(\Sigma_0)$. To represent the negation of $\varphi(\Sigma_i)$, i.e. the $\neg$-node with child $\varphi(\Sigma_i)$, we write $\neg\varphi(\Sigma_i)$.
In this framework we allow for an arbitray number of observations. For this consider the event $h_i(\mathbf{x}_i) = 0$ which represents the observation that subsystem Σ_i (component, module, network, etc.) is down, and similarly $h_i(\mathbf{x}_i) = 1$ stands for the event that Σ_i is up. So for any subsystem Σ_i we may possibly observe $ob_i \in \{h_i(\mathbf{x}_i) = 0, h_i(\mathbf{x}_i) = 1\}$, namely a subsystem failure or operation. Moreover, one may be interested in several queries qr_j at the same time. This amounts to computing the posterior probabilities of several subsystems at once. In fact we may ask for posterior probabilities of success or failure, which means that for any subsystem under investigation Σ_j such that $j \neq i$, we have $qr_j \in \{h_j(\mathbf{x}_j) = 0, h_j(\mathbf{x}_j) = 1\}$. So consider a set $O = \{\Sigma_{i_1}, \ldots, \Sigma_{i_k}\}$ of observed subsystems and a set $Q = \{\Sigma_{j_1}, \ldots, \Sigma_{j_t}\}$ of query subsystems such that $O \cap Q = \emptyset$. The conditional probability of all the queries given the observations is given by

$$P(qr_{j_1}, \ldots, qr_{j_t} \mid ob_{i_1}, \ldots, ob_{i_k}) = \frac{P(qr_{j_1}, \ldots, qr_{j_t}, ob_{i_1}, \ldots, ob_{i_k})}{P(ob_{i_1}, \ldots, ob_{i_k})}. \qquad (18)$$

In the next subsection, we explain how to compute such posterior probabilities.

4.2 Computing Posterior Probabilities

For the subsequent discussion on how to compute the numerator in (18), we restrict ourselves on a single query $qr \in \{h_j(\mathbf{x}_j) = 0, h_j(\mathbf{x}_j) = 1\}$ and and a single observation $ob \in \{h_i(\mathbf{x}_i) = 0, h_i(\mathbf{x}_i) = 1\}$. This is only to keep the computations illustrative and implies no restrictions on the conceptual level. The observed subsystem shall be denoted by $\Sigma_{ob} = \Sigma_i$ and the system under investigation by $\Sigma_{qr} = \Sigma_j$. Under these assumptions, we may express the conditional probability (18) now in terms of cd-PDAGs as

follows:

$$P(qr \,|\, ob) = \frac{P(qr, ob)}{P(ob)} = \frac{P(\,\varphi(\Sigma_{qr}) \wedge \varphi(\Sigma_{ob})\,)}{P(\,\varphi(\Sigma_{ob})\,)}, \tag{19}$$

where $\varphi(\Sigma_{ob})$ and $\varphi(\Sigma_{qr})$ are the respective cd-PDAGs relative to the subsystems Σ_{ob} and Σ_{qr}. They are both sub-PDAGs of the global cd-PDAG $\varphi(\Sigma_0)$. The conjunction $\varphi(\Sigma_{ob}) \wedge \varphi(\Sigma_{qr})$ in the numerator is obtained by joining $\varphi(\Sigma_{ob})$ with $\varphi(\Sigma_{qr})$ by a $\wedge$-node, resulting in a new PDAG which is denoted by $\varphi_{ob\wedge qr}$ in the sequel. The problem now is that this newly created PDAG is not necessarily decomposable, i.e. the children of its root node may now have common variables. This implies that efficient probability computation is no longer assured since $\varphi_{ob\wedge qr}$ is not a cd-PDAG. Such case is illustrated in Fig. 14 with $\varphi(\Sigma_{ob}) = \neg\varphi(A)$ and $\varphi(\Sigma_{qr}) = \varphi(R)$, see Example 9. Remember that making an arbitrary PDAG decomposable is a computationally hard task in the general case.
Here, the strategy to make $\varphi_{ob\wedge qr}$ decomposable depends on the positions of the involved subsystems with respect to each other within the organizing tree, and hence within the global PDAG. So we are basically faced with two possible situations:

1. Σ_{ob} and Σ_{qr} are in distinct sub-trees, or
2. Σ_{ob} and Σ_{qr} are not in distinct sub-trees, i.e. either
   (a) Σ_{qr} is a sub-tree of Σ_{ob}, or
   (b) Σ_{ob} is a sub-tree of Σ_{qr}.

The first case is straightforward since $\varphi(\Sigma_{ob})$ and $\varphi(\Sigma_{qr})$ share no variables which makes $\varphi_{ob\wedge qr}$ automatically decomposable. Its probability $P(\varphi_{ob\wedge qr})$ is thus easily obtained:

$$P(\varphi_{ob\wedge qr}) = P(\,\varphi(\Sigma_{ob}) \wedge \varphi(\Sigma_{qr})\,) = P(\,\varphi(\Sigma_{ob})\,) \cdot P(\,\varphi(\Sigma_{qr})\,), \tag{20}$$

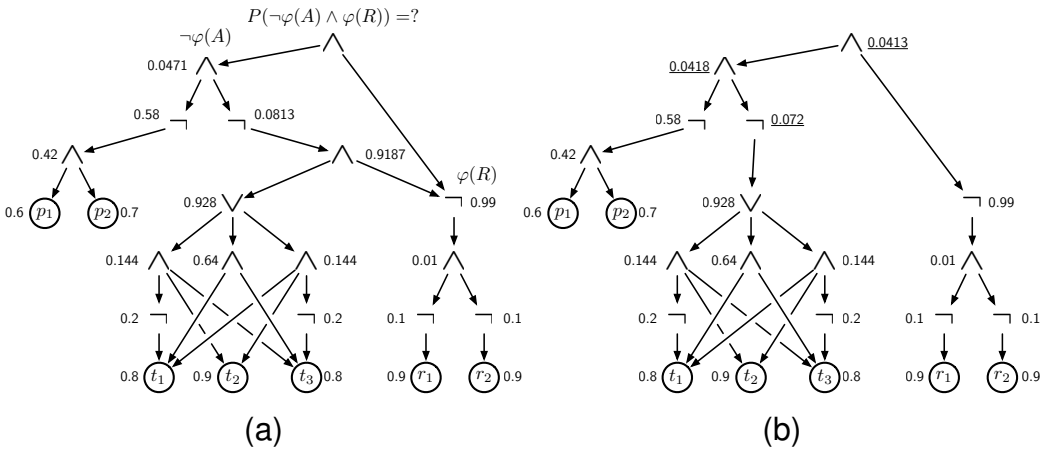
where $P(\varphi(\Sigma_{ob}))$ and $P(\varphi(\Sigma_{qr}))$ are calculated according to (13) in Subsection 3.1. The product (20) implies that $P(qr \,|\, ob) = P(qr) = P(\,\varphi(\Sigma_{qr})\,)$, which means that the query (i.e. the operation of the investigated subsystem) is not affected by the observation.
The second case is more complicated since $\varphi_{ob\wedge qr}$ is not decomposable. To show how to overcome this difficulty, let us first assume the case (a), i.e. Σ_{qr} is a sub-tree of Σ_{ob}. The usual procedure is then to generate from $\varphi_{ob\wedge qr}$ the new PDAG

$$\big(\,\varphi_{ob\wedge qr} \,|\, \varphi(\Sigma_{qr}) \wedge \varphi(\Sigma_{qr})\,\big) \vee \big(\,\varphi_{ob\wedge qr} \,|\, \neg\varphi(\Sigma_{qr}) \wedge \neg\varphi(\Sigma_{qr})\,\big) \tag{21}$$

which corresponds to Shannon's expansion with respect to $\varphi(\Sigma_{qr})$ and involves *conditioning* $\varphi_{ob\wedge qr}$ on $\varphi(\Sigma_{qr})$ and $\neg\varphi(\Sigma_{qr})$, a technique we came across in Subsection 3.3 already. Here, conditioning $\varphi_{ob\wedge qr}$ on $\neg\varphi(\Sigma_{qr})$ yields the contradiction, so we are left with the expression $\varphi_{ob\wedge qr} \,|\, \varphi(\Sigma_{qr}) \wedge \varphi(\Sigma_{qr})$. In the latter, conditioning $\varphi_{ob\wedge qr}$ on $\varphi(\Sigma_{qr})$ results in $\varphi(\Sigma_{ob}) \,|\, \varphi(\Sigma_{qr})$, so that the whole expression (21) can be simplified to $\varphi(\Sigma_{ob}) \,|\, \varphi(\Sigma_{qr}) \wedge \varphi(\Sigma_{qr})$ which satisfies determinism and now also decomposability. This transformation is analogue for the case (b), except that there the roles of Σ_{qr} and Σ_{ob} are interchanged. The conditional probability (19) can be thus rewritten as follows:

$$\begin{aligned} P(qr \,|\, ob) &= \frac{P(\,\varphi(\Sigma_{ob}) \,|\, \varphi(\Sigma_{qr}) \wedge \varphi(\Sigma_{qr})\,)}{P(\,\varphi(\Sigma_{ob})\,)} \\ &= \frac{P(\,\varphi(\Sigma_{ob}) \,|\, \varphi(\Sigma_{qr})\,) \cdot P(\,\varphi(\Sigma_{qr})\,)}{P(\,\varphi(\Sigma_{ob})\,)}. \end{aligned} \tag{22}$$

The formula (22) now allows to efficiently compute the posterior probability of any subsystem Σ_{qr} given some observation about another subsystem Σ_{ob}. This computation can be generalized for the case of multiple queries and multiple observations. In any case, conditioning is involved which is always an efficient operation, i.e. polynomial in the size of the PDAG.



(a) (b)

Figure 14: (a) The PDAG $\neg\varphi(A)\wedge\varphi(R)$, and (b) the cd-PDAG $\neg\varphi(A)|\varphi(R)\wedge\varphi(R)$ with the updated probabilities.

Example 9. Recall our alp-transit system from Example 3 and suppose we observe the whole transalpine traffic to be completely stuck, i.e. the system is down. Given this fact, we may ask now what is the (conditional)

probability that the railway system is working correctly? In relation to the previous discussion, this corresponds to the observed event $ob = \{A = 0\}$ and the query event $qr = \{R = 1\}$. Based on the system cd-PDAG $\varphi(A)$, this leads to the new PDAG $\neg\varphi(A) \wedge \varphi(R)$ which is shown in Fig. 14a. This corresponds to case 2.(a) above since $\varphi(R)$ is a sub-PDAG of $\neg\varphi(A)$, which means that the new PDAG is not decomposable. After appropriate conditioning however, we obtain the new cd-PDAG $\neg\varphi(A)|\varphi(R) \wedge \varphi(R)$ which is depicted in Fig. 14b. This cd-PDAG has three new nodes with respect to the previous PDAG, for which new probabilities have to be computed (they are underlined in the figure). Finally, we are able to compute the posterior probability of the railway system operation given the system failure by

$$P(R = 1 \,|\, A = 0) = \frac{P(\,\neg\varphi(A)|\varphi(R) \wedge \varphi(R)\,)}{P(\,\neg\varphi(A)\,)} \approx \frac{0.0413}{0.0471} \approx 0.877.$$

We can compute exactly in the same way the posterior probabilities of all subsystems. The posterior failure probabilities provide useful information for computing the most probable diagnoses. This allows to identify the modules, components, networks, or any other subsystems which most likely have caused a system failure or some other observed behaviour.

5 Conclusion & Future Research

This paper has explored reliability evaluation of modular systems, reliability networks, and hybrid systems. Hybrid systems result from the combination of modular systems and reliability networks. Further, we have discussed well-known deterministic properties of coherent systems and also some new probabilistic properties. The main contribution of this paper is a common computational framework for reliability and diagnostics, which is based on a compact structure function representation by means of PDAGs. The particular subclass of decomposable and deterministic PDAGs (cd-PDAGs) allows to compute the exact reliability in polynomial time with respect to the size of the underlying cd-PDAG. The superior succinctness properties of cd-PDAGs relative to other well-known Boolean representation languages such as OBDDs and dDNNFs set this Boolean language apart from the rest, hence making it the language of choice for probability computations. Furthermore, it turns out that cd-PDAGs can be used to efficiently compute posterior probabilities of subsystems in a modular system. Results of previous reliability computations can be reused for this purpose. This provides an efficient tool for computing the most probable diagnoses, which underlines once more the duality between reliability and diagnostics. As a conclusion, we can say that our approach constitutes a valuable complement or alternative to existing methods for exact reliability computation, both from a computational and a modeling point of view.

Future research will focus on more general system models, which includes network reliability models with multiple layers, representing structure functions of multistate systems by multistate DAGs (a generalization of PDAGs for multistate variables), and their reliability evaluation. A further issue that is missing in this paper are computational results, which should provide useful insight on how PDAG-based reliability computation compares with competitive methods in practice. There is also still much potential in the context of diagnostics. Until now we have provided the computational key for obtaining posterior probabilities. Based on this, the goal would be to devise clever methods to find the most probable diagnoses.

A Proofs

Proof of Theorem 2.1 To prove the theorem we first prove the following lemma which slightly relaxes the assumptions made in the theorem:

Lemma A.1. Let be $f(x_1, \ldots, x_m)$ a Boolean function which is monotone in the variable x_k, that is $f(\mathbf{x}_{[0_k]}) \leq f(\mathbf{x}_{[1_k]})$ for all vectors

$$\mathbf{x}_{[0_k]} = (x_1, \ldots, x_{k-1}, 0, x_{k+1}, \ldots, x_m),\ \mathbf{x}_{[1_k]} = (x_1, \ldots, x_{k-1}, 1, x_{k+1}, \ldots, x_m).$$

Further let be $P \in \mathbf{P}$ the probability distribution defined in Subsection 2.2.2. Now if we assume $P' \in \mathbf{P}$ such that $P(c_k) \leq P'(c_k)$ and $P(c_i) = P'(c_i)$ for all $i \neq k$, then $P(\{f(\mathbf{x}) = 1\}) \leq P'(\{f(\mathbf{x}) = 1\})$, where $\mathbf{x} \in \Omega^m$.

Proof We can decompose the Boolean space which is induced by the event $E = \{f(\mathbf{x}) = 1\}$ (this space corresponds to the satisfying set of f) into two disjoint spaces induced by the corresponding events E_1 and E_2. The space relative to E_1 contains all vectors $\mathbf{x}_{[0_k]}$ and additionally — due to monotonicity of f with respect to x_k — the corresponding vectors $\mathbf{x}_{[1_k]}$. Then the space relative to E_2 is the space relative to the complement $E \backslash E_1$ which contains the remaining vectors with component $x_k = 1$. Given the disjoint decomposition $E = E_1 \cup E_2$ we can thus write

$$P(E) = P(E_1) + P(E_2). \tag{23}$$

Now we can further expand this formula according to Shannon's decomposition relative to x_k:

$$\begin{aligned} P(E_1) &= P(E_1|x_k = 1) \cdot P(x_k = 1) + P(E_1|x_k = 0) \cdot P(x_k = 0) \\ &= P(E_1|x_k = 1) \end{aligned}$$

since $\{E_1|x_k = 1\} = \{E_1|x_k = 0\}$. A similar observation leads to

$$\begin{aligned} P(E_2) &= P(E_2|x_k = 1) \cdot P(x_k = 1) + P(E_2|x_k = 0) \cdot P(x_k = 0) \\ &= P(E_2|x_k = 1) \cdot P(x_k = 1) \end{aligned}$$

since $P(E_2|x_k = 0) = 0$. Now (23) becomes

$$P(E) = P(E_1|x_k = 1) + P(E_2|x_k = 1) \cdot P(x_k = 1). \tag{24}$$

For the measure P' exactly the same decomposition applies so that

$$P'(E) = P'(E_1|x_k = 1) + P'(E_2|x_k = 1) \cdot P'(x_k = 1). \tag{25}$$

Clearly, $P(E_1|x_k = 1) = P'(E_1|x_k = 1)$ and $P(E_2|x_k = 1) = P'(E_2|x_k = 1)$. And since $P(x_k = 1) \leq P'(x_k = 1)$, it follows that $P(E) \leq P'(E)$. ∎

Corollary A.2. Consider the probability distribution $P \in \mathbf{P}$ from Lemma A.1. Given another distribution $P' \in \mathbf{P}$ such that $P(c_i) \leq P'(c_i)$ for all $i = 1, \ldots, m$ and a monotone Boolean function $f(x_1, \ldots, x_m)$, it holds that $P(\{f(\mathbf{x}) = 1\}) \leq P'(\{f(\mathbf{x}) = 1\})$.

Proof This is a direct consequence of Lemma A.1: the monotonicity property is simply extended with respect to all variables $x_1, \ldots, x_m$. ∎

Corollary A.2 implies thus that if $P(c_i) = a_i$ for all $i = 1, \ldots, m$, then for any other probability distribution $P' \in \mathbf{P}$ it holds that $P(E) \leq P'(E)$ where E is the event defined in Subsection 2.2.2. Thus the lower probability distribution $\underline{P}(E)$ is the smallest distribution with respect to $\mathbf{P}$. This corresponds exactly to assertion (1) of Theorem 2.1 which concludes the proof. The proof of assertion (2) is analogue. ∎

Proof of Theorem 3.1 Since Alg. 1 appears to be a special case of the generalized WFK-algorithm, the proof of correctness can be adopted from [40] with slight modifications. The important observation is that the correctness of the algorithm relies on the fact that the input matrix is defined over a *closed* semiring, which indeed holds in our case. According to [40], a closed semiring S satisfies: $a^* = 1 + a \cdot a^* = 1 + a^* \cdot a$ for all $a \in S$. In our semiring $S_{\mathcal{B}}$, the closure is simple since we have $f^* = 1$ for all $f \in \mathcal{B}_{\leq n}$. This makes $S_{\mathcal{B}}$ a simple semiring, i.e. one that is closed and bounded. ∎

Proof of Theorem 3.2 Due to the properties of the Boolean constants 1 and 0, it does in fact not matter — with respect to the correctness of the method — at which point (i.e. before or after running Alg. 2) the instantiation of the terminal selectors is carried out. Hence we proceed as follows: we fix v_1 as source and v_n as terminal, order the nodes appropriately, and instantiate the terminal selectors already within the initial connectivity matrix A by setting $\lambda_{v_n} = 1$ and $\lambda_{v_i} = 0$ for all $i \in \{1, \ldots, n-1\}$. Now we have to show that the resulting reliability pdag generated by Alg. 2 indeed represents the structure function for v_1, v_n-connectivity.
But first we need to prove two lemmas. The starting point is an instantiated connectivity matrix $A = (f_{ij})_{n \times n}$ (according to above), associated with a complete and directed graph $G = (V, E)$, hence all non-diagonal entries of A are non-zero. Further, $f_{ij}^{(k)}$ denotes the i, j-entry of A at stage k (i.e. after the elimination of k nodes), and φ_{ij} denotes the corresponding specific reliability pdag for the v_i, v_j-connectivity.

Lemma A.3. After $k+1$ steps of Algorithm 2, i.e. after elimination of nodes $v_n, \ldots, v_{n-k}$, each entry $f_{ij}^{(k+1)}$, $i \neq j$, represents the reliability pdag for the v_i, v_j-connectivity, $i, j < n-k$, in the subgraph $G' = (V', E')$ where $V' = \{v_i, v_{n-k}, \ldots, v_n, v_j\}$.

Proof By induction over k. Consider the base case $k = 1$, i.e. v_n has been eliminated. Then we have in matrix $A^{(1)}$:

$$\begin{aligned} f_{ij}^{(1)} &= f_{ij}^{(0)} \vee (f_{in}^{(0)} \wedge f_{nj}^{(0)}) \quad \forall\, i \neq j < n \\ &= e_{ij} \vee (e_{in} \wedge e_{nj}), \end{aligned}$$

which corresponds indeed to the reliability pdag φ_{ij} w.r.t. the subgraph that comprises the node set $\{v_i, v_n, v_j\}$.
Let us assume that the induction hypothesis holds for k, i.e. after eliminating the nodes $v_n, v_{n-1}, \ldots, v_{n-k+1}$. We show now that the assertion holds for $k+1$, i.e. after the elimination of nodes $v_n, \ldots, v_{n-k}$. At stage $k+1$ of the algorithm, the typical i, j-entry of matrix $A^{(k+1)}$ is as follows:

$$f_{ij}^{(k+1)} = \underbrace{f_{ij}^{(k)}}_{(1)} \vee (\underbrace{f_{i,n-k}^{(k)}}_{(2)} \wedge \underbrace{f_{n-k,j}^{(k)}}_{(3)}) \quad \forall\, i \neq j < n-k.$$

According to the induction hypothesis, we have the following:

- $(1) = \varphi_{ij}$ w.r.t. the subgraph with the node set $\{v_i, v_n, \ldots, v_{n-k+1}, v_j\}$
- $(2) = \varphi_{i,n-k}$ w.r.t. the subgraph with the node set $\{v_i, v_n, \ldots, v_{n-k}\}$
- $(3) = \varphi_{n-k,j}$ w.r.t. the subgraph with the node set $\{v_n, \ldots, v_{n-k}, v_j\}$

Let us interpret the above reliability pdags in terms of elementary paths. Then the combined reliability pdag $\varphi_{i,n-k} \wedge \varphi_{n-k,j}$ represents elementary paths from v_i to v_j in the subgraph with the merged node set $\{v_i, v_n, \ldots, v_{n-k}, v_j\}$. The expression φ_{ij} also represents elementary paths from v_i to v_j, but those which do not go through the most recently eliminated node v_{n-k}. Combining this expression with the former term, we obtain

$$\varphi_{ij} \vee (\varphi_{i,n-k} \wedge \varphi_{n-k,j}),$$

which represents all elementary paths from v_i leading to v_j in the subgraph with nodes $v_i, v_n, \ldots, v_{n-k}, v_j$. ∎

Lemma A.4. After $k+1$ steps of Algorithm 2, i.e. after elimination of nodes $v_n, \ldots, v_{n-k}$, each entry $f_{ii}^{(k+1)}$ represents the reliability pdag for the v_i, v_n-connectivity, $1 \leq i \leq n-k-1$, in the subgraph $G' = (V', E')$ where $V' = \{v_i, v_{n-k}, \ldots, v_n\}$.

Proof Let us verify the base case first, i.e. $k = 1$. After the elimination of v_n the diagonal entries of $A^{(1)}$ have the form

$$\begin{aligned} f_{ii}^{(1)} &= f_{ii}^{(0)} \vee (f_{in}^{(0)} \wedge f_{nn}^{(0)}) \quad \forall\, 1 \leq i \leq n-1 \\ &= 0 \vee (f_{in}^{(0)} \wedge 1) = f_{in}^{(0)} = e_{in}, \end{aligned}$$

corresponding to the reliability pdags φ_{in} in the respective subgraphs w.r.t. the node sets $\{v_i, v_n\}$.
Assuming that the assertion holds for k, we proceed by verifying it for $k+1$, i.e. after the elimination of nodes $v_n, \ldots, v_{n-k}$. The diagonal entries of $A^{(k+1)}$ have the following form:

$$f_{jj}^{(k+1)} = \underbrace{f_{ii}^{(k)}}_{(1)} \vee (\underbrace{f_{i,n-k}^{(k)}}_{(2)} \wedge \underbrace{f_{n-k,n-k}^{(k)}}_{(3)}) \quad \forall\, 1 \leq j \leq n-k-1.$$

According to the induction hypothesis, the entries (1) and (3) represent the following reliability pdags:

$(1) = \varphi_{in}$ w.r.t. the subgraph with the node set $\{v_i, v_n, \ldots, v_{n-k+1}\}$, $1 \leq i \leq n-k$,
$(3) = \varphi_{n-k,n}$ w.r.t. the subgraph with the node set $\{v_n, \ldots, v_{n-k}\}$,

Due to Lemma A.3, entry (2) represents the reliability pdag $\varphi_{i,n-k}$ w.r.t. the subgraph with the node set $\{v_i, v_n, \ldots, v_{n-k}\},\ i < n-k$.
Now by similar reasoning as in Lemma A.3, the reliability pdag $\varphi_{i,n-k} \wedge \varphi_{n-k,n}$ represents elementary paths from v_i to v_n in the subgraph with nodes $\{v_i, v_n, \ldots, v_{n-k}\}$, for $i < n-k$. The reliability pdag φ_{in} represents in turn all elementary paths from v_i to v_n in the subgraph of nodes $\{v_i, v_n, \ldots, v_{n-k+1}\}$, for $1 \leq i \leq n-k$. Given these considerations, the combined reliability pdag

$$\varphi_{in} \vee (\varphi_{i,n-k} \wedge \varphi_{n-k,n})$$

represents thus all elementary paths from v_i to v_n in the subgraph with the set of nodes $v_i, v_{n-k}, \ldots, v_n$, $1 \leq i \leq n-k-1$. ∎

Theorem 3.2 follows directly from Lemma A.4, since at the end of Algorithm 2 all nodes except v_1 have been eliminated. Hence after $n-1$ steps, entry $f_{11}^{(n-1)}$ represents the specific reliability pdag for the v_1, v_n-connectivity in the complete graph. ∎

References

[1] B. Anrig and J. Kohlas, “Model-based reliability and diagnostic: A common framework for reliability and diagnostics,” in *DX’02, 13th Interational Workshop on Principles of Diagnosis* (M. Stumptner and F. Wotawa, eds.), (Semmering, Austria), pp. 129–136, 2002.

[2] J. Kohlas, B. Anrig, and R. Bissig, “Reliability and diagnostic of modular systems,” *ORiON: The Journal of the Operations Research Society of South Africa*, vol. 16, no. 1, pp. 47–62, 2001.

[3] M. Wachter, R. Haenni, and J. Jonczy, “Reliability and diagnostics of modular systems: a new probabilistic approach,” in *DX’06, 17th International Workshop on Principles of Diagnosis* (C. A. González, T. Escobet, and B. Pulido, eds.), (Peñaranda de Duero, Spain), pp. 273–280, 2006.

[4] A. Rueda and M. Pawlak, “Pioneers of the reliability theories of the past 50 years,” in *RAMS’04, 50th Annual Reliability and Maintainability Symposium*, (Los Angeles, USA), pp. 102–109, 2004.

[5] R. E. Barlow, “Mathematical reliability theory: from the beginning to the present time,” in *MMR’02, 3rd International Conference on Mathematical Methods In Reliability* (B. Lindqvist and K. A. Doksum, eds.), no. 7 in Series on Quality, Reliability and Engineering Statistics, (Trondheim, Norway), pp. 73–79, World Scientific Publishing, 2002.

[6] R. Reiter, “A theory of diagnosis from first principles,” *Artificial Intelligence*, vol. 32, pp. 57–95, 1987.

[7] J. de Kleer and B. C. Williams, “Diagnosing multiple faults,” *Artificial Intelligence*, vol. 32, pp. 97–130, 1987.

[8] J. de Kleer, A. K. Mackworth, and R. Reiter, “Characterizing diagnoses and systems,” *Artificial Intelligence*, vol. 56, no. 2–3, pp. 197–222, 1992.

[9] J. Kohlas, B. Anrig, R. Haenni, and P. A. Monney, “Model-based diagnostics and probabilistic assumption-based reasoning,” *Artificial Intelligence*, vol. 104, pp. 71–106, 1998.

[10] P. Fröhlich, *DRUM–II: Efficient Model–Based Diagnosis of Technical Systems*. PhD thesis, University of Hannover, Germany, 1998.

[11] J. de Kleer, "An assumption-based TMS," *Artificial Intelligence*, vol. 28, pp. 127–162, 1986.

[12] J. Kohlas, *Zuverlässigkeit und Verfügbarkeit*. Teubner, 1987.

[13] R. E. Barlow, F. Proschan, and P. Franken, *Statistische Theorie der Zuverlässigkeit*. Berlin, Germany: Berlin Akademie-Verlag, 1981.

[14] C. J. Colbourn, *The Combinatorics of Network Reliability*. New York, USA: Oxford University Press, 1987.

[15] D. R. Shier, *Network reliability and algebraic structures*. New York, USA: Oxford Clarendon Press, 1991.

[16] M. O. Ball, C. J. Colbourn, and J. S. Provan, "Network reliability," in *Network Models* (M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, eds.), vol. 7 of *Handbooks in Operations Research and Management Science*, pp. 673–762, Elsevier, 1995.

[17] L. G. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979.

[18] M. Wachter and R. Haenni, "Propositional DAGs: a new graph-based language for representing Boolean functions," in *KR'06, 10th International Conference on Principles of Knowledge Representation and Reasoning* (P. Doherty, J. Mylopoulos, and C. Welty, eds.), (Lake District, U.K.), pp. 277–285, AAAI Press, 2006.

[19] M. Wachter and R. Haenni, "Multi-state directed acyclic graphs," in *CanAI'07, 20th Canadian Conference on Artificial Intelligence* (Z. Kobti and D. Wu, eds.), LNAI 4509, (Montréal, Canada), pp. 464–475, 2007.

[20] P. Clote and E. Kranakis, *Boolean Functions and Computation Models*. Springer, 1998.

[21] P. Walley, *Statistical Reasoning with Imprecise Probabilities*. Monographs on Statistics and Applied Probability 42, London, U.K.: Chapman and Hall, 1991.

[22] F. G. Cozman, "Credal networks," *Artificial Intelligence*, vol. 120, no. 2, pp. 199–233, 2000.

[23] I. Levi, *The Enterprise of Knowledge*. Cambridge, USA: The MIT Press, 1980.

[24] R. Haenni, "Climbing the hills of compiled credal networks," in *ISIPTA'07, 5th International Symposium on Imprecise Probabilities and Their Applications* (G. de Cooman, J. Vejnarová, and M. Zaffalon, eds.), (Prague, Czech Republic), pp. 213–222, 2007.

[25] R. M. Sinnamon and J. D. Andrews, "Fault-tree analysis and binary decision diagrams," in *IEEE Annual Reliability and Maintainability Symposium*, (Las Vegas, USA), pp. 215–222, 1996.

[26] X. Zang, D. Wang, H. Sun, and K. S. Trivedi, "A BDD-based algorithm for analysis of multistate systems with multistate components," *IEEE Transactions on Computers*, vol. 52, no. 12, pp. 1608–1618, 2003.

[27] A. Satyanarayana and A. Prabhakar, "New topological formula and rapid algorithm for reliability analysis of complex networks," *IEEE Transactions on Reliability*, vol. R-27, pp. 82–100, 1978.

[28] J. A. Abraham, "An improved algorithm for network reliability," *IEEE Transactions on Reliability*, vol. 28, pp. 58–61, 1979.

[29] K. Heidtmann, "Smaller sums of disjoint products by subproducts inversion," *IEEE Transactions on Reliability*, vol. 38, no. 4, pp. 305–311, 1989.

[30] K. D. Heidtmann, "Statistical comparison of two sum-of-disjoint-product algorithms for reliability and safety evaluation," in *SAFECOMP'02, 21st International Conference on Computer Safety, Reliability and Security*, (Catania, Italy), pp. 70–81, 2002.

[31] R. Bertschy and P. A. Monney, "A generalization of the algorithm of Heidtmann to non-monotone formulas," *Journal of Computational and Applied Mathematics*, vol. 76, pp. 55–76, 1996.

[32] G. Fey and R. Drechsler, "Utilizing BDDs for disjoint SOP minimization," in *MWSCAS'02, 45th IEEE International Midwest Symposium on Circuits and Systems*, (Tulsa, USA), pp. 306–309, 2002.

[33] S. Y. Kuo, S. K. Lu, and F. M. Yeh, "Determining terminal-pair reliability based on edge expansion diagrams using OBDD," *IEEE Transactions on Reliability*, vol. 48, no. 3, pp. 234–246, 1999.

[34] X. Zang, H. Sun, and K. S. Trivedi, "A BDD-based algorithm for reliability graph analysis," tech. rep., Department of Electrical Engineering, Duke University, 2000.

[35] G. Hardy, C. Lucet, and N. Limnios, "K-terminal Network Reliability measures with Binary Decision Diagrams," *IEEE Transactions on Reliability*, vol. 56, pp. 506–515, September 2007.

[36] J. Jonczy and R. Haenni, "Network reliability evaluation with propositional directed acyclic graphs," in *Advances in Mathematical Modeling for Reliability* (T. Bedford, J. Quigley, L. Walls, B. Alkali, A. Daneshkhah, and G. Hardman, eds.), pp. 25–31, IOS Press, 2008.

[37] M. Wachter, *Knowledge Compilation Map – Theory and Application*. PhD thesis, University of Bern, Switzerland, 2008.

[38] S. Warshall, "A theorem on boolean matrices," *Journal of the ACM*, vol. 9, no. 1, pp. 11 – 12, 1962.

[39] R. W. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[40] D. Lehmann, "Algebraic structures for transitive closure," *Theoretical Computer Science*, vol. 4, pp. 59–76, 1977.

[41] B. Roy, "Transitivité et connexité," *C. R. Acad. Sci. Paris*, vol. 249, pp. 216–218, 1959.

[42] G. Rote, "Path problems in graphs," in *Computational Graphs Theory* (G. Tinhofer, E. Mayr, H. Noltemeier, M. M. Syslo, and R. Albrecht, eds.), Computing Supplementum 7, pp. 155–198, Springer, 1990.

[43] B. A. Carré, *Graphs and Networks*. Oxford Clarendon Press, 1979.

[44] R. Wankar, E. Fehr, and N. S. Chaudhari, "An efficient parallel algorithm for the all pairs shortest path problem using processor arrays with reconfigurable bus systems," Tech. Rep. B-13-99, Freie Universität Berlin, Germany, 1999.

[45] L. Huang, "Dynamic programming algorithms in semiring and hypergraph frameworks," tech. rep., University of Pennsylvania, Philadelphia, USA, November 2006.