(Technical Report 2001-02)

Multi-layer Fault Localization Using Probabilistic Inference in Bipartite Dependency Graphs

Małgorzata Steinder, Adarshpal S. Sethi
Computer and Information Sciences Department
University of Delaware, Newark, DE

***Abstract*– For the purpose of fault diagnosis, communication systems are frequently modeled in a layered fashion imitating the layered architecture of the modeled system. The layered model represents relationships between services, protocols, and functions offered between neighboring protocol layers. In a given layer, an end-to-end service between two hosts may be provided using multiple hop-to-hop services offered in this layer between two hosts on the end-to-end path. When an end-to-end service fails or experiences performance problems it is critical to efficiently find the responsible hop-to-hop services. Dependencies between end-to-end and hop-to-hop services form a bipartite graph whose structure depends on the network topology in the corresponding protocol layer. To represent the uncertainty in the dependency graph, probabilities are assigned to its nodes and links. Finding the most probable explanation (MPE) of the observed symptoms in the probabilistic dependency graph is NP-hard. We transform the bipartite dependency graph to a belief network and investigate several algorithms for computing MPE such as bucket tree elimination and two approximations based on Pearl's iterative algorithms. We also introduce a novel algorithm using an iterative hypothesis update. These algorithms are implemented in Java and their performance and accuracy are evaluated through extensive simulation study.**

1 Introduction

Fault diagnosis is a central aspect of network fault management. Since faults are unavoidable in communication systems, their quick detection and isolation is essential for system robustness, reliability and accessibility. Traditionally, fault diagnosis has been performed manually by an expert or a group of experts experienced in managing communication networks. However, as systems grew larger and more complex, automated fault diagnosis techniques became critical.

Fault localization (also event correlation or root cause diagnosis) [28, 34, 61], an important stage of the fault diagnosis process, isolates the most probable set of faults based on their external manifestations called symptoms or alarms. The most probable set of faults proposed in the fault localization stage constitutes a symptom explanation hypothesis, which is later tested to pinpoint the actual faults. Fault localization aims at locating root causes of the observed symptoms, i.e., faults that may not be further explained.

In the past, fault diagnosis efforts concentrated mostly on detecting, isolating, and correcting faults related to network connectivity [17, 34, 59, 61]. The diagnosis focused on lower layers of the protocol stack (typically physical and data-link layers) [47, 61], and its major goal was to isolate faults related to the availability of network resources, such as broken cable, inactive interface, etc. Since these types of problems are relatively rare, most event correlation techniques existing today assume that only one fault may exist in the system at any time. Also, many existing techniques [34, 61] rely on time-windows to collect a set of symptoms to explain. The time windows are typically in order of several minutes. Usage of time-windows postpones the initialization of the fault localization process. It also lacks in flexibility since time-window lengths may be different for different systems and different types of faults and symptoms. We believe that, when symptom latencies are not easy to predict, fault localization is better realized in an iterative and incremental fashion, i.e., the solution is updated after every symptom observation. Iterative fault localization also allows the actual alarm correlation to be interleaved with additional testing procedures, thereby improving the overall performance and accuracy.

The demands of the modern enterprise services such as e-commerce, telecommuting, virtual private networks [55], and application service provision [10] change the requirements imposed on the fault localization process. E-business customers increasingly demand support for quality of service (QoS) guarantees. QoS parameters are negotiated between a customer and e-business as part of Service Level Agreements [25] (SLAs), which also specify pricing rules for the offered services and a penalty schema to be used if the quality of the offered service violates the agreed upon SLA contract. Various techniques have been investigated to supervise execution of the SLA contract [2, 23], and to notify the management application about any QoS violations. In addition to dealing with resource availability problems, fault diagnosis has to isolate the causes of these performance/QoS related notifications. In such an e-commerce environment, diagnosis may no longer be constrained to the lowest layers of the protocol stack. On the contrary, fault diagnosis has to reach through the transport and application layers into the service layer. Since upper layers heavily depend on lower layers, the fault management system has to integrate fault diagnosis across multiple protocol layers. Performance related problems are more frequent than availability related ones. In large systems, it is likely for two or more unrelated performance problems to occur simultaneously. Therefore, fault diagnosis has to be able to isolate multiple unrelated root causes.

Network connectivity is frequently achieved through a sequence of intermediate nodes invisible to the layers above. For example, in the data-link layer, end-to-end connectivity is provided by a network of bridges; in the network layer, end-to-end connectivity is realized by a network of routers. Similar scenarios exist in the application layer. We say that the end-to-end service provided by a given layer is realized by a set of hop-to-hop services in that layer. When an end-to-end service fails, one needs to locate hop-to-hop services responsible for the end-to-end service failure. Diagnosing end-to-end service failures, both availability and performance related ones, is a difficult problem in complex network topologies. This paper considers end-to-end service failure diagnosis to be a crucial step towards multi-layer fault localization in an enterprise environment. We present probabilistic iterative fault localization techniques capable of isolating multiple-simultaneous root problems responsible for end-to-end service failures in a given layer. The proposed solutions allow the management system to perform fault localization iteratively in real-time.

In Section 2, we describe the layered dependency graph model for multi-layer fault diagnosis refined to expose the end-to-end service model and to allow non-deterministic reasoning about both availability and performance related problems. In Section 3, we describe some graph and belief networks concepts used in this paper. Section 4 introduces the mapping of the layered dependency graph into a belief network, which for end-to-end service failure diagnosis forms a bipartite graph. In Section 5, we describe five algorithms for finding the best symptoms' explanation using a bipartite belief network, which include bucket tree elimination [14] and two approximations based on Pearl's iterative algorithms [49]. We also introduce a novel algorithm using an iterative hypothesis update. These algorithms were implemented in Java and their performance and accuracy were evaluated through extensive simulation study described in Section 6. A comparison of our solutions with other event correlation techniques is presented in Section 7.

2 Layered Model for Alarm Correlation

For the purpose of fault diagnosis, communication systems are frequently modeled in a layered fashion imitating the layered architecture of the modeled system [21, 43, 61]. This approach provides a natural abstraction of the modeled system's entities, reusability of the model's modules, and ability to divide the fault management task into separate, simpler subtasks. The main purpose of the model is to represent information about events, i.e., state changes of a communication system's entities, and their impact on the state of other entities. The ability of a fault in one entity to change the state of other entities, referred to as fault propagation, is an inherent feature of communicating systems. Because of fault propagation, the effects of abnormal operation of functions or services provided by lower layers may be observed in higher layers. While propagating up the protocol stack, the failures change their semantics thereby losing information important for their localization. For example, a failure at the data link layer to successfully transmit a packet across a link may be observed in a higher layer as an inability to *ping* the IP host to which packets are transmitted using the failed link. Similarly, a router failure in the network layer may be observable in the transport layer as an inability to establish the TCP connection with a TCP host to which IP datagrams are routed using the failed router. In order to find explanations of higher-layer problems, it is useful to create a fault propagation model. Fault management systems model fault propagation by representing either causal relationships between events [9, 22, 61] or dependencies between communication system entities [21, 32, 34, 54].

2.1 Layered model template

In the layered fault model, the definition of entity dependencies is based on real-life relationships between layers on a single host and between network nodes communicating within a single protocol layer. The fault model components may be generally divided into *services*, *protocols*, and *functions* [21]. A service offered by protocol layer L between nodes **a** and **b** (*Service$_L$(a,b)*) is implemented in terms of layer L functions on hosts **a** and **b** (*Network Functions$_L$(a)* and *Network Functions$_L$(b)*), and the layer L protocols through which hosts **a** and **b** communicate. The layer L protocols running between hosts **a** and **b** use layer $L-1$ functions on hosts **a** and **b**, and services that layer $L-1$ offers between hosts **a** and **b**. Layer L functions on node **a** depend on layer $L-1$ functions on node **a**. The recursive dependencies between services, protocols and functions constitute a dependency graph as described in [21]. In this paper, we find it useful to eliminate the protocol nodes. This model simplification is justified, since it may be assumed that the protocols are implemented correctly; under this assumption, protocols cannot contribute explanations to service failures. Figure 1 shows the resultant general dependency graph for a layered network, in which *Service$_L$(a,c)* directly depends on *Service$_{L-1}$(a,c)*.

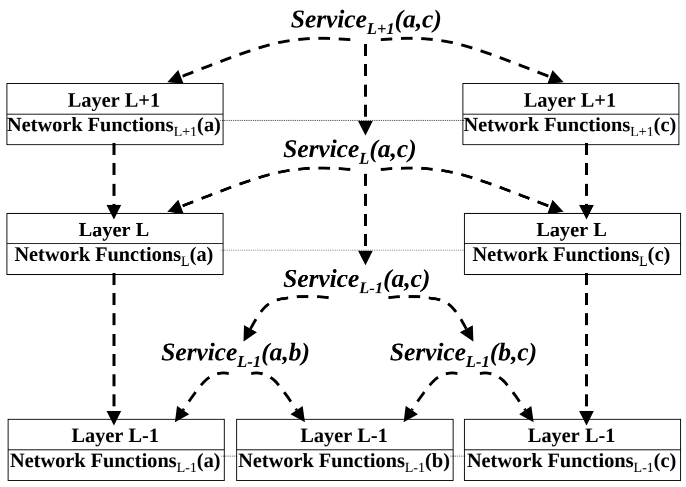


Figure 1: Layered network dependency model

The general dependency graph template obtained from services, protocols and functions in different layers provides a macro-view of the relationships that exist in the system. It may be argued that fault localization should be performed starting from a macro-view to select a potential spot of the problem, and then it should focus on the micro-view of the chosen spot [48]. To incorporate the micro-view of the relationships within particular model components, the layered model should be further refined to include possibly complex relationships within services, protocols and functions in the same layer. Thus, *Network Functions$_L$(a)* should be represented as a graph of multiple layer L functions on node **a** implementing *Network Functions$_L$(a)*. Similarly, *Service$_L$(a,b)* could be extended into a subgraph containing multiple layer L sub-services used to provide *Service$_L$(a,b)*. In particular, an end-to-end service offered by layer L between hosts **a** and **c** is implemented in terms of multiple hop-to-hop services offered by layer L between subsequent hops on the path of the layer L packet from node **a** to node **c** (such as *Service$_{L-1}$(a,c)* in Figure 1). The ability to reason about failures observed in an end-to-end service, i.e., symptoms, and trace them down to particular host-to-host service failures, i.e., faults, is critical in order to perform fault diagnosis in complex network topologies and is the primary focus of the presented research.

In this paper, besides the elimination of the protocol nodes, the model presented in [21] is refined as follows. With every dependency graph node we associate multiple failure modes $F_1, \ldots, F_k$, which represent availability and performance problems pertaining to the service or function represented by the dependency graph node. In real-life systems, the following conditions are typically monitored for and considered a service/function failure:

- F_1 – service/function ceases to exist (e.g., the cable connection is broken),
- F_2 – service/function introduces unacceptable delay (e.g., one of the hop-to-hop links in network layer is congested),
- F_3 – service/function produces erroneous output (e.g., bit errors are introduced in a serial link between routers),
- F_4 – service/function occasionally does not produce output (e.g., packets are lost due to buffer overflow).

The knowledge of communication protocols makes it possible to predict which of these conditions will occur in a higher-level service/function if any of these conditions occur in one or more lower-layer services/functions.

2.2 *Non-determinism and its representation in the layered model*

The fault management application monitors the communication system to detect abnormal service and function state changes. The fault localization component uses the fault model and the observations (symptoms) to determine the cause of the observed abnormal behavior. The analysis is complicated by the fact that most observations may have multiple explanations and the same fault may cause multiple symptoms. Various techniques have been proposed to make the process of fault localization accurate and efficient [22, 33, 47, 61]. The common feature of these approaches is that their fault model is deterministic, i.e., the dependency link from **a** to **b** implies that if **a** fails, then **b** also fails. The deterministic model is typically sufficient to represent faults in lower layers of the protocol stack related to the availability of services offered by these layers. However, these fault localization techniques are rather difficult to apply when faults are Byzantine [50], e.g., related to service performance. In the transport and application layers, frequent reconfigurations of service dependencies make it impossible to keep such a deterministic model up-to-date. The following are some possible scenarios, in which the deterministic model is inadequate:

- *Service*$_{L-1}$*(a,b)* in Figure 1 fails by rejecting some data it is supposed to process (condition F_4). Luckily, none of these data are related to *Service*$_{L-1}$*(a,c)*; therefore, *Service*$_{L-1}$*(a,c)* is not affected.
- *Service*$_{L-1}$*(a,b)* fails by rejecting some data related to *Service*$_{L-1}$*(a,c)* it is supposed to process. However, the data loss is not sufficient to cause the observable degradation of *Service*$_{L-1}$*(a,c)*.
- *Service*$_{L-1}$*(a,b)* fails by delaying processing of data related to *Service*$_{L-1}$*(a,c)* (condition F_2). However, *Service*$_{L-1}$*(b,c)* processes data fast and makes up for the delay so that no degradation of *Service*$_{L-1}$*(a,c)* is observed.
- *Service*$_{L-1}$*(a,c)* fails (conditions F_1, F_2, F_3, or F_4) but the dependent *Service*$_L$*(a,c)* is not currently in use; therefore, no failure of *Service*$_L$*(a,c)* will be observed.
- *Service*$_{L-1}$*(a,b)* fails (conditions F_2, F_3, or F_4) and affects *Service*$_L$*(a,c)*, however, the failure detection mechanism is not sensitive enough to detect the failure of *Service*$_L$*(a,c)*, therefore no symptom related to the failure of *Service*$_L$*(a,c)* is generated.

Uncertainty about dependencies between communication system entities is represented by assigning probabilities to the links in the dependency or causality graph [34, 36]. Some commonly accepted assumptions in this context are that (1) given fault **a**, the occurrences of faults **b** and **c** that may be caused by **a** are independent, (2) given the occurrences of faults **a** and **b** that may cause event **c**, whether **a** actually causes **c** is independent of whether **b** causes **c** (OR relationship between alternative causes of the same event), and (3) faults (root causes) are independent of one another. We take advantage of these approximating assumptions throughout the paper.

Contrary to other publications on this subject [34], in this paper, the dependency graph nodes have multiple failure modes. Therefore, instead of a single probability value, we assign probability matrices to the dependency links. Let $\mathcal{F}_X$ denote a set of failure modes related to service or function X, and $\mathcal{F}_Y$ denote a set of failure modes related to the dependent service or function Y. The label assigned to dependency link $Y \rightarrow X$ is a two-dimensional matrix $|\mathcal{F}_Y| \times |\mathcal{F}_X|$, $\mathcal{P}$, such that $\mathcal{P}(F_j, F_i) = P\{$*service/function* Y *is in failure mode* $F_j|$ *service/function* X *is in failure mode* $F_i\}$, where $F_j \in \mathcal{F}_Y$ and $F_i \in \mathcal{F}_X$.

2.3 *Obtaining the dependency graph*

The dependency graph described in Section 2.1 records two types of dependencies between services and functions in neighboring protocol layers: *static* and *dynamic* dependencies. As opposed to static dependencies, dynamic dependencies may change during the system runtime. Static dependencies result from, e.g., standardized definition of functions provided by different layers of the protocol stack, or from static network topologies. While static dependencies are considered an easier case, building them manually in large systems is frequently impossible. Therefore, automated techniques of obtaining static dependencies have been investigated. The network topology may be obtained automatically through various network topology detection mechanisms [46, 51], which are built into some commercially available network management systems [58]. Automated detection of static dependencies within software components on a single machine was investigated in [31].

Dynamic dependencies result from, e.g., run-time addition and deletion of services (such as establishment and termination of TCP sessions). To determine the existence of such dynamic services, some popular software utilities may be used, e.g., all active TCP connections may be retrieved using the *netstat* application [56]. Other techniques have been proposed in [6, 19, 51]. Another source of dynamic dependencies is the usage of routing protocols (such as the Spanning Tree Protocol [50] in the data-link layer or any dynamic routing protocol in the network layer), or dynamic configuration changes. Because of the dynamic routing protocols, an end-to-end service may depend on different sets of host-to-host services at different times. In order to reason about the causes of the end-to-end service failures, we need to determine the currently used set of host-to-host services. Network management protocols such as SNMP [7] provide the means to determine dependencies established using configuration or real-time algorithms. The following list presents several examples and specifies how the

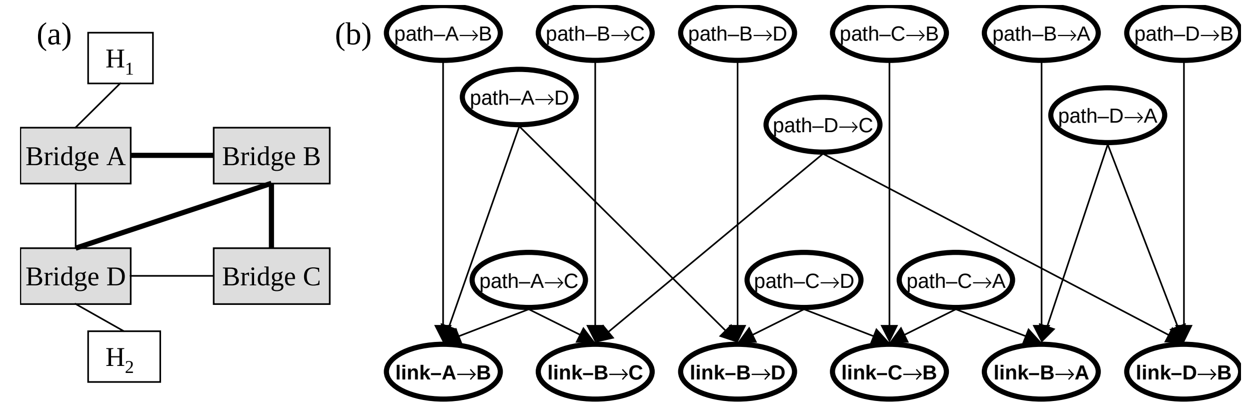


Figure 2: (a) Example bridge topology with the current spanning tree marked in bold; (b) Dependency graph built based on the spanning tree in (a)

dependencies may be obtained using the well-known network management mechanisms.

1 In many networks the IP datagram forwarding service in the network layer is provided through a data link layer forwarding service between bridges or switches. When, for reliability concerns, multiple paths are provided between two bridges, the Spanning Tree Protocol [50] ensures that only one path at a time is possible between any two network nodes. In the data link layer, the links that belong to the spanning tree are the ones currently used to provide the network layer service. The management system may obtain the spanning tree from SNMP agents on the bridges/switches using the data contained in `dot1dBase` Group of *Bridge MIB* [16]. Updates of the spanning tree may be triggered by `newRoot` and `topologyChange` traps [16].

2 In IEEE 802.5 Token Ring [57] networks with source routing bridges, the lost of links in the data link layer used for communication between two end-stations may be obtained from the management agent on the source end-station using data stored by its agent in `dot5SrRouteTable` of *Station Source Routing MIB* [40].

3 The list of IP hop-to-hop links used to provide the end-to-end communication in the network layer may be obtained from the management agents on IP routers using data stored in `ipRoutingTable` of *TCP/IP MIB-II* [41]. While `ipRoutingTable` provides only the identifier of the next-hop host to the particular address, the full route may be computed by following the next-hop identifiers from the routing table on the source host towards the destination host. The alternative method of acquiring the route is by using the program *traceroute* [56].

4 Some protocols may offer proprietary methods of obtaining routing information. This is particularly straightforward for source routing protocols because the routing information is included in every transmitted packet. Two examples of real-life source routing protocols that would enable this type of route retrieval are Source-Directed Relay of the military protocol MIL-STD 188-220 [18] and Dynamic Source Routing [29] proposed for wireless mobile networks.

In Figure 2 we present a dependency graph for data link layer services in the simple network topology composed of four learning bridges [50]. The current spanning tree obtained as described above is marked in bold lines. In the dependency graph, we distinguish between *links*, which provide bridge-to-bridge delivery service, and *paths*, which provide packet delivery service from the first to the last bridge on the packet route from the source node to the destination node. The delivery service provided by paths is built of delivery services provided by links. We find it reasonable to consider unidirectional communication between two hosts a service, although in many circumstances this would not be necessary. However, it is sometimes possible for a communication between two hosts to fail only in one direction, while in the opposite direction it remains intact. By distinguishing between opposite directions, it becomes possible to detect these situations.

The dependency graph presented in Figure 2-(b) constitutes a part of a bigger layered fault model. At the higher level, the model contains IP-source-to-IP-destination delivery service implemented in terms of IP-source-to-bridge, bridge-to-IP-destination packet delivery services and the service provided by the inter-bridge path between IP-source and IP-destination. For example, the delivery service between hosts H_1 and H_2 is provided using packet delivery services between host H_1 and *Bridge A*, *Bridge A* and *Bridge D*, and *Bridge D* and host H_2. When the service between hosts H_1 and H_2 experiences one of the failures F_1, F_2, F_3, or F_4 presented in Section 2, the failure may be blamed on one or more services the H_1–H_2 service depends on. When, either through testing or an additional symptom analysis, it is determined that the problem was caused by the failure of the *path–A→D* service, the fault localization function has to determine the faulty link service.

In the non-deterministic fault-model, locating a faulty link service when the path service fails may be rather complex. In large networks, testing all link services is time consuming even if it is technically possible. Therefore, before any tests are scheduled, the link services that are the most likely to have caused the path problem should be determined based on the analysis of the observed symptoms, i.e., path service failures. To build such a fault hypothesis, in the following sections, we present the application of several algorithms for computing the most probable explanation in *belief networks* [49] as well as a combinatorial algorithm suggested in [34]. We also introduce a novel algorithm using an iterative hypothesis update.

3 Graph Concepts

We now present the basic concepts of graph and belief network theory used in the next sections of this paper.

A *directed graph* is a pair $G = (V, E)$, where V is a set of nodes and $E = \{(V_i, V_j) \mid V_i, V_j \in V\}$ is a set of edges. We will denote by N the number of nodes in graph G, i.e., $N = |V|$. If edge (V_i, V_j) belongs to graph G, we say that V_i is a *parent* of V_j, and V_j is a *child* of V_i. If both (V_i, V_j) and (V_j, V_i) belong to E we say that G contains an undirected edge between V_i and V_j; we also say that V_i and V_j are *neighbors*. A *directed acyclic graph* (DAG) is a directed graph with no directed cycles. An *undirected graph* is a graph that contains only undirected edges.

An *ordered graph* is a pair (G, o), where G is an undirected graph and $o = V_1, V_2, \ldots, V_N$ is an ordering of the nodes. In the ordered graph, the number of neighbors of node V_i that precede it in the ordering is called the *width of node* V_i. A *modified ordered graph* (G_M, o) of an ordered graph (G, o) is created as follows: (1) the nodes of graph G are processed according to ordering o from last to first; (2) while processing node V_i, all its neighbors that precede it in the order are connected to one another using undirected links. The *width of an ordered graph* (G, o), $w*(o)$, is the maximum node width in the modified ordered graph (G_M, o). The *induced width of the directed graph* G, $w*$, is the minimum width of (G, o) computed over all orderings o [14].

The *moral graph* of a directed graph G is obtained by introducing additional undirected edges between any two nodes with a common child and then converting all directed edges into undirected ones [12]. Consider a directed graph obtained from the dependency graph in Figure 2-(b) by reversing all its edges, which will be used in the following sections to represent causal relationships between the end-to-end service failures and hop-to-hop service failures. The inverted graph is moralized by introducing undirected edges between *link–A→B* and *link–B→C*, *link–A→B* and *link–B→D*, *link–B→C* and *link–D→B*, *link–B→D* and *link–C→B*, *link–C→B* and *link–B→A*, as well as *link–B→A* and *link–D→B*. Then, the arrows are removed. The moralization of the inverted graph in Figure 2-(b) leads to the creation of cliques, e.g., a subgraph containing nodes *link–C→B*, *link–B→A*, and *path–C→A*. Each of the cliques contains one *path* node and multiple *link* nodes. One can observe that processing nodes as described in the previous paragraph according to the ordering in which all *link* nodes are given the priority, i.e., they are processed last, would result in the creation of cliques identical to the cliques in the moralized graph. The width of the inverted graph in Figure 2-(b) ordered as described above is thus equal to the maximum clique size in the moralized graph minus one. One can show that this ordering induces the minimum width of the ordered graph. The maximum clique size in the moralized graph is bound by the maximum path length in the original network graph (i.e., spanning tree in Figure 2-(a)). Thus, for an n-bridge/router network the maximum clique of the moralized inverted dependency graph contains n nodes. This lets us conclude that the induced width of the inverted directed graphs of the shape presented in Figure 2-(b) created for n-node networks has an upper bound of $n - 1$.

A *belief network* [14, 49] is a directed acyclic graph [14] (DAG), in which each node represents a random variable over a multivalued domain. We will use terms "node" and "random variable" interchangeably, and denote them by V_i. The set of all nodes is denoted by V. The domain of random variable V_i will be denoted by symbol D_i. The set of directed edges E denotes an existence of causal relationships between the variables and the strengths of these influences are specified by conditional probabilities. Formally, a belief network is a pair (G, P), where G is a DAG, $P = \{P_i\}$, and P_i is the conditional probability matrix associated with a random variable V_i. Let $Par(V_i) = \{V_{i_1}, \ldots, V_{i_n}\}$ be the set of all parents of V_i. P_i is a $(|Par(V_i)|+1)$-dimensional matrix of size $|D_i|\times|D_{i_1}|\times\ldots\times|D_{i_n}|$, where $P_i(v_i, v_{i_1}, \ldots, v_{i_n}) = P(V_i{=}v_i | V_{i_1}{=}v_{i_1}, \ldots, V_{i_n}{=}v_{i_n})$. We will denote by $\mathcal{A}=\{V_1{=}v_1, \ldots, V_n{=}v_n\}$ an assignment of values to variables in set V where each $v_j \in D_j$. We will use $v_j^{\mathcal{A}}$ to denote the value of variable $V_j \in V$ in alignment $\mathcal{A}$. Given a subset of random variables $U_k{=}\{V_{k_1}, \ldots, V_{k_m}\} \subseteq V$, we will denote by $U_k^{\mathcal{A}}{=}\{V_{k_1}{=}v_{k_1}^{\mathcal{A}}, \ldots, V_{k_m}{=}v_{k_m}^{\mathcal{A}}\}$ an assignment of values to variables in set U_k that is consistent with assignment $\mathcal{A}$. An evidence set e is an assignment $U_o^{\mathcal{A}}$, where $U_o \subseteq V$ is a set of variables whose values are known, and for each $V_{o_j} \in U_o$, $v_{o_j}^{\mathcal{A}}$ is its observed value.

Belief networks are used to make four basic queries given evidence set e: belief assessment, most probable explanation, maximum a posteriori hypothesis, and maximum expected utility [14]. The first two queries are of particular interest in the presented research. The *belief assessment* task is to compute $bel(V_i{=}v_i){=}P(V_i{=}v_i|e)$ for one or more variables V_i. The *most probable explanation* (MPE) task is to find an assignment $\mathcal{A}_{\max}$ that best explains the observed evidence e, i.e., $P(\mathcal{A}_{\max}){=}\max_{\mathcal{A}} \Pi_{i=1}^{n} P(V_i{=}v_i^{\mathcal{A}}|Par(V_i)^{\mathcal{A}})$ [14]. It is known that these tasks are NP-hard in general belief networks [11]. A belief updating algorithm, polynomial with respect to $|V|$, is available for *polytrees*, i.e., directed graphs without undirected cycles [49]. However, in unconstrained polytrees, the propagation algorithm still has an exponential bound with respect to the number of node's neighbors.

Since exact inference in belief networks is NP-hard, various approximation techniques have been investigated [15, 49, 52]. To the best of our knowledge, no approximation has been proposed that works well for all types of networks. Moreover, some approximation schemas have been proven to be NP-hard [13].

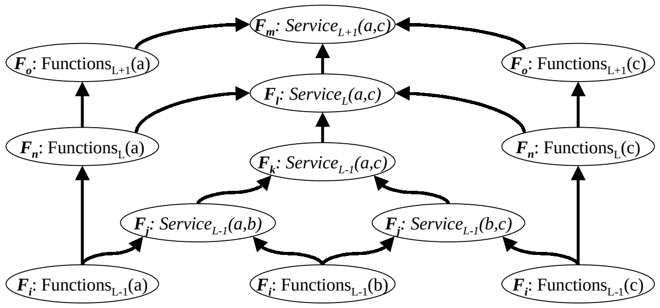
In this paper, we focus on a class of belief networks representing a simplified model of conditional probabilities called *noisy-OR gates* [49] (or QMR networks [12]). The simplified model contains binary-valued random variables. The noisy-OR model associates an inhibitory factor with every cause of a single effect. The effect is absent only if all inhibitors corresponding to the present causes are activated. The model assumes that all inhibitory mechanisms are independent [24, 49]. The usage of this model is justified by assumptions we made in Section 2. This simplification helps avoid exponential time and memory otherwise needed to process and store conditional probability matrices associated with random variables in the belief network. Furthermore, belief assessment in polytrees with the noisy-OR model has polynomial complexity, which makes it attractive to use with our problem as an approximation schema.

4 Mapping Layered Model into Belief Network

We build a belief network based on the layered dependency graph described in Section 2 as follows.

- For every node of the layered dependency graph and for every failure mode associated with this node, we create a random variable, whose domain is $\{$*true*, *false*$\}$. Let V_i be a belief network node created for failure mode F_j of the dependency graph node representing *Service*$_L$*(a,b)* or *Network Function*$_L$*(a)*. Assignment V_i =*true* indicates that *Service*$_L$*(a,b)* or *Network Function*$_L$*(a)* is in condition F_j. Assignment V_i =*false* that *Service*$_L$*(a,b)* or *Network Function*$_L$*(a)* is NOT in condition F_j.
- For every dependency graph edge X→Y and for every failure mode of node Y, F_i, determine F_j, the failure mode of node X that results from condition F_i in node Y. This determination may be performed based on the knowledge of communication protocols. For example, knowing that layer L protocol implements an error detection mechanism, one can predict that erroneous output produced by *Service*$_{L-1}$*(a,b)* (condition F_3) results in data loss in *Service*$_L$*(a,b)* (condition F_4). When layer L does not implement an error detection mechanism, condition F_3 in *Service*$_{L-1}$*(a,b)* results in condition F_3 in *Service*$_L$*(a,b)*. Let V_i be the belief network node corresponding to dependency graph node Y and failure mode F_i. Let V_j be the belief network node corresponding to dependency graph node X and failure mode F_j. Add a belief network edge pointing from V_i to V_j.
- Let $\mathcal{P}$ be the probability matrix associated with dependency link X→Y. The probability matrix P_j associated with node V_j represents the following conditional probability distribution.
  $P(V_j=\textit{false} \mid V_i=\textit{false}) = 1$
  $P(V_j=\textit{false} \mid V_i=\textit{true}) = 1 - \mathcal{P}(F_i, F_j)$
  $P(V_j=\textit{true} \mid V_i=\textit{false}) = 0$
  $P(V_j=\textit{true} \mid V_i=\textit{true}) = \mathcal{P}(F_i, F_j)$

The belief network resulting from the mapping of the dependency graph presented in Figure 1 consists of one or more, possibly overlapping, belief networks of the shape presented in Figure 3.



F_m: Service$_{L+1}$(a,c)
F_o: Functions$_{L+1}$(a)
F_o: Functions$_{L+1}$(c)
F_l: Service$_L$(a,c)
F_n: Functions$_L$(a)
F_n: Functions$_L$(c)
F_k: Service$_{L-1}$(a,c)
F_j: Service$_{L-1}$(a,b)
F_j: Service$_{L-1}$(b,c)
F_i: Functions$_{L-1}$(a)
F_i: Functions$_{L-1}$(b)
F_i: Functions$_{L-1}$(c)

Figure 3: Belief network built for the dependency graph in Figure 1; $F_i, F_j, F_k, F_l, F_m, F_n, F_o \in \{F_1, F_2, F_3, F_4\}$.

To complete the mapping of the fault localization problem into the problem of computing queries in belief networks, we need to define the interpretation of faults and symptoms in the domain of belief networks. A symptom is defined as an observation that a dependency graph node X, which typically corresponds to a higher-level service, is in condition F_j (*negative* symptom), or is NOT in condition F_j (*positive* symptom). We will denote by $\mathcal{S}$ the set of all possible symptoms. If V_i is the belief network node corresponding to the dependency graph node X and its failure mode F_i, then the negative symptom is interpreted as an instantiation of V_i with value *true*, and the positive symptom is interpreted as an instantiation of V_i with value *false*. Thus, as a result of this mapping, the set of all observed symptoms, which will be denoted by $\mathcal{S}_o \subseteq \mathcal{S}$, becomes the evidence set e. The dependency graph node X, which corresponds to a lower-level service or function, is at fault if it is in any of the conditions $F_1, \ldots, F_4$, say condition F_i. The set of all possible faults is denoted by $\mathcal{F}$. The fact that the service or function corresponding to X is in failure mode F_i is represented by value *true* in the domain of the random variable V_i. The problem of finding the set of faults, $\mathcal{F}_c \subseteq \mathcal{F}$ that best explains the set of observed symptoms $\mathcal{S}_o$ is equivalent to computing the MPE query based on the evidence set e.

5 Algorithms

In this section, five algorithms are presented to find the best symptom explanation with causal dependencies between events represented by graphs described in Section 4. We start from a *combinatorial algorithm* [5] used as an optimal algorithm in [34]. Then, three algorithms based on belief networks are presented: *bucket-tree elimination* [14] and adaptations of two algorithms for polytrees [49]: *iterative belief propagation in polytrees*, and *iterative MPE in polytrees*. Finally, we introduce a novel algorithm based on *iterative hypothesis update*. We will use n to denote the number of nodes (bridges, switches, or routers) in the managed system.

5.1 *Combinatorial algorithm*

The combinatorial algorithm presented in this section assumes a naive approach by evaluating all possible combinations of faults for their ability to explain the observed symptoms. For a given combination of faults $\mathcal{F}_i$ and a set of observed symptoms $\mathcal{S}_o$, the measure of goodness $g(\mathcal{F}_i, \mathcal{S}_o)$ is computed as follows.

$$\begin{aligned} g(\mathcal{F}_i, \mathcal{S}_o) &= P\{\text{all faults in } \mathcal{F}_i \text{ occurred}\} \cdot \\ &\quad P\{\text{each symptom in } \mathcal{S}_o \text{ is caused by at least} \\ &\quad \text{one fault in } \mathcal{F}_i\} \\ &= \prod_{f \in \mathcal{F}_i} P(f) \cdot \prod_{s \in \mathcal{S}_o} \left(1 - \prod_{f \in \mathcal{F}_i} \left(1 - P(s \mid f)\right)\right) \end{aligned}$$

While correlating real-life symptoms, it is frequently assumed that the number of faults that occurred is small. This suggests that in the combinatorial algorithm we should start evaluating fault combinations from those that contain the fewest faults and terminate the search as soon as an explanation of all symptoms is known. This leads to the following combinatorial algorithm.

Algorithm 1 (Combinatorial Algorithm)

```
for $i = 1$ until $i < |\mathcal{F}|$ do
    for all $i$-fault combinations from $\mathcal{F}$, $\mathcal{F}_i$
        compute $g(\mathcal{F}_i, \mathcal{S}_o)$
    if at least one $\mathcal{F}_i$ is found such that $g(\mathcal{F}_i, \mathcal{S}_o) > 0$
```

return $\mathcal{F}_i$ such that $g(\mathcal{F}_i, \mathcal{S}_o)$ is maximum

It may be easily calculated that Algorithm 1 performs $\sum_{i=1}^{|\mathcal{F}|}\binom{|\mathcal{F}|}{i} \cdot i \cdot |\mathcal{S}_o| = \mathcal{O}(2^n)$ operations. However, when multiple concurrent faults are unlikely the algorithm's practical complexity may be polynomial. In our simulation study we will determine if the exponential bound is a significant factor in practical applications and if implementation of other, more complicated algorithms is justified.

5.2 Most probable explanation through bucket elimination

Bucket elimination [14] (Algorithm 2) is one of the most popular algorithmic frameworks for computing queries listed in Section 3 in belief networks. In this section we present computation of the most probable explanation (MPE) query.

Algorithm 2 (Bucket elimination MPE)

```
    initialize buckets $B_1, \ldots, B_{|V|}$ for variables $V_1, \ldots, V_{|V|}$
Backward phase:
    for $p = |V|$ downto $1$ do
        if $(V_p = v_p) \in e$ then
            for each $h_j(V_{j_1}, \ldots, V_{j_m}) \in B_p$ do
                let $j_k$ be an index of $V_p$ in the parameter list of $h_j$
                let $h'_j(V_{j_1}, \ldots, V_{j_{k-1}}, V_{j_{k+1}}, \ldots, V_{j_m}) =$
                    $h_j(V_{j_1}, \ldots, V_{j_{k-1}}, v_p, V_{j_{k+1}}, \ldots, V_{j_m})$
                put $h'_j$ in the bucket of variable $V_{j_l} \in var(h_j)$
                  that has the highest number in ordering $o$
        else let $U_p = \bigcup_{i=1}^{|B_p|} var(h_i)$, $h_i \in B_p$
                let $i_k$ be an index of $V_p$ in the parameter list of $h_i$
                for all $\mathcal{A}_p$ compute
                  $h_p(\mathcal{A}_p) = \max_{v_p} \prod_{i=1}^{|B_p|} h_i(v_{i_1}^{\mathcal{A}_p}, \ldots, v_{i_{k-1}}^{\mathcal{A}_p}, v_p,$
                                                  $v_{i_{k+1}}^{\mathcal{A}_p}, \ldots, v_{i_m}^{\mathcal{A}_p})$
                  $v_p^{opt}(\mathcal{A}_p) = argmax_{V_p} h_p(\mathcal{A}_p)$
Forward phase:
    for $p = 1$ upto $|V|$ do $\mathcal{A}_p^{opt} = \mathcal{A}_{p-1}^{opt} \cup \{v_p^{opt}(\mathcal{A}_{p-1}^{opt})\}$
```

Algorithm 2 works by creating a set of buckets, one for every random variable in the belief network. Given ordering o, the random variables are numbered consecutively. Initially, the bucket for variable V_i, B_i, contains all functions $h_j(V_{j_1}, \ldots, V_{j_m}) = P_{j_1}$ such that none of the variables $V_{j_1}, \ldots, V_{j_m}$ is higher in ordering o than V_i; P_{j_1} is an m-dimensional conditional probability matrix associated with node V_{j_1} and random variables $V_{j_2}, \ldots, V_{j_m}$ are all parents of random variable V_{j_1}. The buckets are then eliminated starting from the last according to ordering o. Eliminating a bucket removes its corresponding variable from all functions in this bucket. If the bucket's corresponding variable has been observed, then the elimination of the bucket is performed by assigning the observed value in each m-parameter function in the bucket and placing thus created $m-1$-parameter function in the bucket corresponding to its highest numbered variable. A bucket corresponding to an unobserved variable is eliminated by converting all its functions into one function using maximization as the elimination operator. The elimination performed in bucket B_i computes, for all possible value assignments to variables mentioned in B_i excluding V_i, the value of variable V_i that maximizes the product of all functions in B_i.

In the second phase, the algorithm proceeds from the lowest numbered variable to the highest numbered one and collects the values of their most probable assignments. Initially, the assignment contains only the optimal value for V_1. In the i-th step, the partial assignment $\mathcal{A}_{i-1}^{opt} = (v_1, \ldots, v_{i-1})$ is extended with the optimal value for variable V_i computed in the backward phase for partial assignment $\mathcal{A}_{i-1}^{opt}$.

The following notation is used in the formal presentation of the algorithm. (1) If h_i belongs to bucket B_p, $var(h_i)$ denotes the set of all variables mentioned in h_i excluding V_p. (2) $\mathcal{A}_p$ is an assignment of values to variables in U_p, where $U_p = \{V_1, \ldots, V_p\}$. (3) For $V_i \in U_p$, $v_i^{\mathcal{A}_p}$ is a value of variable V_i in assignment $\mathcal{A}_p$.

The *bucket elimination* algorithm for computing MPE is exact and always outputs a solution. We consider it the optimal algorithm for computing the explanation of the observed symptoms. The computational complexity of the algorithm is determined by the number of variables in every bucket and is bound by $\mathcal{O}(|V| exp(w^*(o)))$, where $w^*(o)$ is the width of the graph induced by ordering o, defined in Section 3. For bipartite graphs such as the one in Figure 2, the complexity is $\mathcal{O}(n^2 exp(n))$ assuming that (1) the optimal ordering is applied as described in Section 3 and (2) the belief network contains all possible *path* nodes (there are $\mathcal{O}(n^2)$ such nodes possible). Sections 5.3, 5.4, and 5.5 present three algorithms of polynomial complexity.

5.3 Iterative inference in Bayesian polytrees

Recall from Section 3 that in singly-connected networks (polytrees) representing the noisy-OR-gate model of conditional probability distribution, Bayesian inference (belief updating) may be computed in polynomial time using the algorithm presented in [49]. The graph in Figure 2-(b) is not a polytree because it contains an undirected loop, *path–A→D — link–A→B — path–A→C — link–B→C — path–D→C — link–D→B — path–D→A — link–B→A — path–C→A — link–C→B — path–C→D — link–B→D — path–A→D.*

Networks with loops violate certain independence assumptions based on which the local computation equations were derived for polytrees. As suggested in [49], the iterative algorithm in loopy networks may or may not converge. Nevertheless, successful applications of the iterative algorithm have been reported. The most famous of them are Turbo-Codes [4] that offer near Shannon limit correcting coding and decoding. The Turbo-Codes decoding algorithm was shown to be an instance of iterative belief propagation in polytrees applied to loopy networks [42]. Other compound codes were also formulated as a problem of belief propagation in graphical models [37]. Previous applications of a deterministic decoding schema to fault localization in deterministic fault models [61] inspire the application of probabilistic decoding to fault localization in non-deterministic fault models.

The effectiveness of iterative propagation in loopy networks and its near-optimal accuracy came as a surprise to the research community [53]. While there is no theoretical explanation to these results, some empirical research has been performed to determine which properties of graphs make it more likely to achieve high accuracy while applying iterative propagation, and when no convergence can be achieved. In [45] the performance of iterative propagation in various types of graphs is investigated including bipartite graphs like the one in Figure 2-(b). It is concluded that, while iterative propagation may offer close-to-optimal accuracy for many types of networks, there are properties of conditional probability distributions that make their corresponding Bayesian networks prone to oscillations when the iterative algorithm is applied. Low prior probabilities (i.e., values in probability matrices of parent-less nodes) and small conditional probabilities associated with the causal links seem to be contributing factors affecting the lack of the iterative algorithm's convergence [45].

The impact of the low prior probabilities on the applicability of iterative propagation to loopy networks is discouraging because prior probabilities in the fault localization task, which correspond to independent fault occurrence probabilities, are very small. In spite of that, our research investigates the iterative propagation technique in bipartite fault graphs. The reason for this is that we are not interested in the precise value of the belief metric; as long as the relative values are preserved we can still hope to achieve a good solution. In Section 6 we show the encouraging results of this investigation.

Recall from Section 4 that the problem of fault localization may be translated into the most probable explanation (MPE) query in belief networks. The iterative algorithms for polytrees proposed in [49] include the algorithm for calculating MPE. Nevertheless, we start presenting iterative algorithms from the description of belief updating, which is conceptually simpler. We also present an adaptation of belief updating to estimating the MPE.

Iterative belief propagation utilizes a message passing schema in which the belief network nodes exchange λ and π messages (Figure 4). Message $\lambda_X(v_j)$ that node X sends to its parent V_j for every valid V_j's value v_j, denotes a posterior probability of the entire body of evidence in the sub-graph obtained by removing link $V_j \rightarrow X$ that contains X, given that $V_j = v_j$. Message $\pi_{U_i}(x)$ that node X sends to its child U_i for every valid value of X, x, denotes a probability that $X = x$ given the entire body of evidence in the subgraph containing U_i created by removing edge $X \rightarrow U_i$. In this section, we present a summary of the iterative algorithm for polytrees and its application to the fault localization problem. The complete description of the iterative algorithm for polytrees along with some illustrative examples may be found in [49]. Based on the messages received from its parents and children, node X computes $\lambda(x)$, $\pi(x)$, and $bel(x)$ as follows [49]:

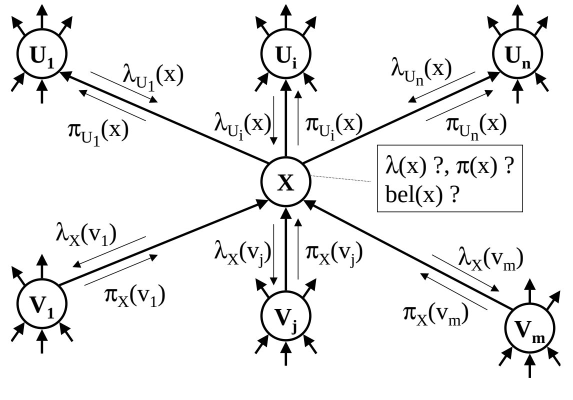


Figure 4: Message passing in Pearl's belief propagation

$$\lambda(x) = \prod_{i=1}^{n} \lambda_{U_i}(x)$$
$$\pi(x) = \begin{cases} \alpha \prod_{j=1}^{m}(1 - c_{V_j X}\pi_{jX}) & \text{if } x = 1 \\ \alpha(1 - \prod_{j=1}^{m}(1 - c_{V_j X}\pi_{jX})) & \text{if } x = 0 \end{cases}$$
$$bel(x) = \alpha\lambda(x)\pi(x)$$

In the above equations, $\pi_{jX} = \pi_X(v_j)$ for $v_j = 1$, α is a normalizing constant, and β is any constant. In a noisy-OR polytree, let us denote by q_{XU_i} the probability of activating the inhibitor controlling link $X \rightarrow U_i$. Every random variable assumes values from $\{0, 1\}$, where 1 denotes occurrence of the corresponding event and 0 means that the event did not occur. The probability that U_i occurs given X occurs is $c_{XU_i} = 1 - q_{XU_i}$. The messages $\lambda_X(v_j)$ and $\pi_{U_i}(x)$ are computed using the following equations [49]:

$$\lambda_X(v_j) = \beta\Big(\lambda(1) - q_{V_j X}^{v_j}(\lambda(1) - \lambda(0)) \prod_{k \neq j}(1 - c_{V_k X}\pi_{kX})\Big)$$
$$\pi_{U_i}(x) = \alpha \prod_{k \neq i} \lambda_{U_k}(x)\pi(x)$$

In the initialization phase, for all observed nodes X, $\lambda(x)$ is set to 1 if x is the observed value of X. For other values of x, $\lambda(x)$ is set to 0. For all unobserved nodes $\lambda(x)$ is set to 1 for all values of x. Parentless nodes have their $\pi(x)$ set to the prior probabilities. The belief propagation algorithm in polytrees starts from the evidence node and propagates the changed belief along the graph edges by computing $bel(x)$, $\lambda_X(v_i)$'s and $\pi_X(u_i)$'s in every visited node. In loopy graphs, several iterations are performed in which the entire graph is searched according to some pre-defined ordering.

This paper adapts the iterative belief propagation algorithm to the problem of fault localization with fault models represented by bipartite graphs as in Figure 2-(b). In this application, we perform one traversal of the entire graph for every observed symptom. For every symptom we define a different ordering that is equivalent to the breadth-first order started in the node representing the observed symptom.

Algorithm 3 (MPE through iterative belief updating)

```
Inference iteration starting from node $Y_i$:
    let $o$ be the breadth-first order starting from $Y_i$
    for all nodes $X$ along ordering $o$ do
        if $X$ is not an unobserved path node then
            compute $\lambda_X(v_j)$ for all $X$'s parents, $V_j$,
                and for all $v_j \in \{0,1\}$
            compute $\pi_{U_i}(x)$ for all $X$'s children, $U_i$,
                and for all $x \in \{0,1\}$
Symptom analysis phase:
    for every symptom $S_i \in \mathcal{S}_O$ do
        run inference iteration starting from $S_i$
    compute $bel(v_i)$ for every node $V_i$, $v_i \in \{0,1\}$
Fault selection phase:
    while $\exists$ link node $V_j$ for which $bel(1)>0.5$ and $\mathcal{S}_O \neq \emptyset$ do
        take $V_j$ with the greatest $bel(1)$
        mark $V_j$ as observed to have value of 1
        remove all symptoms explained by $V_j$ from $\mathcal{S}_O$
        run inference iteration starting from $V_j$
```

compute $bel(v_i)$ for every node V_i, $v_i \in \{0,1\}$

It may be noticed that in the unobserved *path* nodes, because their $\lambda(x) = 1$ for any value of x, $\lambda_X(v_j) = 1$ regardless of the values of other messages in the expression. Since *path* nodes have no children, there is no need to compute π messages. This allows us to avoid the calculations in the unobserved *path* nodes and thereby significantly improve performance.

The computations described above allow us to obtain the marginal posterior distribution resulting from the observation of the evidence (symptoms). Based on this distribution we need to choose faults that explain the evidence. We choose a *link* node with the highest posterior probability, place the corresponding fault in the MPE hypothesis, mark the node as observed with value 1, and perform one iteration of the belief propagation starting from the chosen *link* node. This step is repeated until (1) the posterior distribution contains *link* nodes whose probability is greater than 0.5, and (2) unexplained symptoms remain.

Local computations in *path* nodes require $\mathcal{O}(k)$ operations, where k is the number of links that constitute the path. Since in an n-node network, a path may be composed of at most n links, local computations in *path* nodes require $\mathcal{O}(n)$ operations. Thus, in a single iteration processing *path* nodes requires $\mathcal{O}(n(|\mathcal{S}|) \subseteq \mathcal{O}(n^3)$ operations. Local computations in *link* nodes require $\mathcal{O}(k)$ steps, where k is a number of node's children. Thus, processing all *link* nodes is $\mathcal{O}(\sum k)$. Observe that $\sum k =$ the number of all causal links in the bipartite graph, i.e., n^3 because there are at most n^2 *path* nodes and every path may be composed of at most n links. Therefore, processing all *link* nodes requires $\mathcal{O}(n^3)$ operations. We may conclude that a single iteration of the algorithm is $\mathcal{O}(n^3)$, and the complexity of the entire algorithm is $\mathcal{O}(|\mathcal{S}_o|n^3) \subseteq \mathcal{O}(n^5)$.

5.4 *Iterative most probable explanation in Bayesian polytrees*

In this section, we present the application of the iterative MPE algorithm for polytrees [49] to networks with undirected loops. The iterative belief updating algorithm presented in Section 5.3 computes the marginal posterior probability of the Bayesian network variable values given the observed evidence. In Algorithm 3, we used this distribution to select the most probable explanation. The MPE algorithm in every iteration produces the most probable value assignment to the belief network nodes. This allows us to eliminate the *fault selection* phase in Algorithm 3, which contributes to the complexity and is an additional source of the inaccuracy.

Similarly to belief updating, the MPE computation algorithm proceeds from the evidence nodes by passing messages λ^* and π^* along the belief network edges. Message $\lambda^*_X(v_j)$ sent by node X to its parent V_j represents the conditional probability of the most probable prognosis for the values of nodes located in the subgraph containing X resulting from the removal of the link $V_i \to X$, given the proposition $V_i = v_j$. Message $\pi^*_{U_i}(x)$ sent by node X to its child U_i represents the probability of the most probable values of the nodes located in the subgraph containing X resulting from the removal of link $X \to U_i$, which include the proposition $X = x$. The belief metric $bel^*(x)$ stands for the probability of the most probable explanation of evidence e that is consistent with the proposition $X = x$. Messages $\lambda^*_X(v_j)$ and $\pi^*_{U_i}(x)$, and belief metric $bel^*(x)$ are computed using the following equations [49]:

$$\lambda^*_X(v_j) = \max_{x,\{v_k \neq v_j\}} \Big(\prod_i \lambda^*_{U_i}(x) P(x|v_1,\ldots,v_m) \prod_{k\neq j} \pi^*_X(v_k) \Big)$$
$$\pi^*_{U_i}(x) = \prod_{k\neq i} \lambda^*_{U_k}(x) \max_{\{v_k\}} \Big(P(x|v_1,\ldots,v_m) \prod_k \pi^*_X(v_k) \Big)$$
$$bel^*(x) = \beta \prod_k \lambda^*_{U_k}(x) \max_{\{v_k\}} \Big(P(x|v_1,\ldots,v_m) \prod_k \pi^*_X(v_k) \Big)$$

The calculation of $\max_{\{v_k\}}(P(x|v_1,\ldots,v_m)\prod_k \pi^*_X(v_k))$ is the primary difficulty in obtaining λ^*s, π^*s, and bel^*s. Using notation from Section 5.3, the maximization may be expressed as follows:

$$\max_{\{v_k\}} \Big(P(x|v_1,\ldots,v_m) \prod_k \pi^*_X(v_k) \Big) =$$
$$\begin{cases} \max_{\{v_k\}}\Big((\prod_{V_k|v_k=1} q_{V_k X}) \prod_k \pi^*_X(v_k) \Big) & \text{if } x = 0 \\ \max_{\{v_k\}}\Big((1 - \prod_{V_k|v_k=1} q_{V_k X}) \prod_k \pi^*_X(v_k) \Big) & \text{if } x = 1 \end{cases}$$

While for x=0 the expression may be simplified to $\prod_k \max(q_{V_k X}\pi^*_X(v_k{=}1), \pi^*_X(v_k{=}0))$, the exact computation of the maximization for $x = 1$ requires enumerating all possible combinations of value assignments to the parents of X, and choosing a combination that maximizes the value of the expression. Clearly, listing all combinations is computationally infeasible.

In this paper, we propose an approximation that allows to compute the maximization expression in polynomial time. Let $E(v_1,\ldots,v_m)$= $P(x{=}1|v_1,\ldots,v_m)\prod_k \pi^*_X(v_k)$. First, note that $P(x{=}1|v_1,\ldots,v_m)$=$\Big(1 - \prod_{V_j=1} q_{V_j X}\Big)$. A combination that maximizes the value of expression $E(v_1,\ldots,v_m)$ must contain at least one assignment $V_j = 1$. Otherwise, $E(v_1,\ldots,v_m)$ would be equal to zero. The approximation presented in this paper is based on the fact that the observation $X = 1$ is more likely to have been caused by an activation of a single parent of X, than by simultaneous activations of two or more parents of X. The calculation presented below aims at finding the set of all parents of X, π_1, that should be assigned to one in the combination that best explains the observation $X = 1$. The best choices for V_j's $\in \pi_1$ are those parents of X for which $\pi^*_X(V_j{=}0)$=0, because all combinations in which such V_j=0 result in $E(v_1,\ldots,v_m)$=0. If no V_js exist such that $\pi^*_X(V_j = 0) = 0$, then we pick one V_j for which $c_{V_j X}\pi^*_X(1)/\pi^*_X(0)$ is maximum. In this expression, $c_{V_j X}\pi^*_X(1)$ and $\pi^*_X(0)$ represent an estimate of V_j's contribution to $E(v_1,\ldots,v_m)$ with $v_j = 1$ and $v_j = 0$, respectively. Expression $c_{V_j X}\pi^*_X(1)/\pi^*_X(0)$ approximates

$$\frac{E(v_1,\ldots,v_{j-1},1,v_{j+1},\ldots,v_m)}{E(v_1,\ldots,v_{j-1},0,v_{j+1},\ldots,v_m)},$$

i.e., the benefit resulting from changing V_j's value from 0 to 1 in the parameter list of $E(v_1,\ldots,v_m)$. V_j's for which neither $\pi^*_X(V_j{=}0)$=0 nor $c_{V_j X}\pi^*_X(1)/\pi^*_X(0)$ is maximum are assigned to one if their $\pi^*_X(v_j{=}0) \le \pi^*_X(v_j{=}1)$. This condition ensures that V_j has bigger contribution to the value of $E(v_1,\ldots,v_m)$ when it is assigned to 1 rather than 0, which is proven by the following inequation:

$$\frac{E(v_1,\ldots,v_{j-1},1,v_{j+1},\ldots,v_m)}{E(v_1,\ldots,v_{j-1},0,v_{j+1},\ldots,v_m)} = \frac{1-q_{V_jX}\prod_{V_k\neq V_j|v_k=1}q_{V_kX}}{1-\prod_{V_k\neq V_j|v_k=1}q_{V_kX}}\cdot\frac{\pi^*_X(v_j=1)}{\pi^*_X(v_j=0)} \geq 1$$

V_js that do not meet any of the conditions described above are assigned value 0.

Let $\pi_1{=}\{V_j|\pi^*_X(v_j{=}0){=}0\}$ if at least one such V_j exists. Otherwise, $\pi_1{=}\{V_B\}$ where $V_B{=}\text{argmax}_{V_j}(c_{V_jX}\pi^*_X(1)\,/\pi^*_X(0))$. Let $\pi_0{=}\{V_j|\pi^*_X(v_j{=}1){=}0\}$. The following equations summarize our approximation technique of the maximization for $x=1$.

$$\max_{\{v_k\}}\Big(P(x|v_1,\ldots,v_m)\prod_k\pi^*_X(v_k)\Big)\simeq(1-q_X)\pi_X$$

$$q_X=\prod_{V_j\in\pi_1}q_{V_jX}\prod_{V_j|\pi^*_X(v_j=0)\leq\pi^*_X(v_j=1)}q_{V_jX}$$

$$\pi_X=\prod_{V_j\in\pi_1}\pi^*_X(1)\prod_{V_j\in\pi_0}\pi^*_X(0)\prod_{V_j\notin\pi_0\cup\pi_1}\max(\pi^*_X(1),\pi^*_X(0))$$

Thus, the complete expression for the maximization is as follows:

$$\max_{\{v_k\}}\Big(P(x|v_1,\ldots,v_m)\prod_k\pi^*_X(v_k)\Big)\simeq\begin{cases}(1-q_X)\pi_X & \text{if } x=1\\ \prod_{V_k}\max(q_{V_kX}\pi^*_X(1),\pi^*_X(0)) & \text{if } x=0\end{cases}$$

The above expression is then substituted instead of the maximization in the computation of $bel^*(x)$ and $\pi^*_{U_i}(x)$, to compute their approximations $\tilde{bel}^*(x)$ and $\tilde{\pi}^*_{U_i}(x)$, respectively. The approximation for $\lambda^*_X(v_j)$ follows the same reasoning with two modifications: (1) the maximization does not include V_j, which we address by replacing q_X and π_X with $q^{(j)}_X$ and $\pi^{(j)}_X$ presented below, respectively; (2) for $v_j=1$, $V_j\in\pi_1$, which makes the search for other π_1 members unnecessary; therefore we use $q^{(j)}_{X1}$ presented below instead of $q^{(j)}_X$. We approximate $\lambda^*_X(v_j)$ as follows:

$$\tilde{\lambda}^*_X(v_j)=\begin{cases}\max\Big(\prod_i\lambda^*_{U_i}(0)\prod_{k\neq j}\max(q_{V_kX}\pi^*_X(1),\pi^*_X(0)),\\ \qquad\prod_i\lambda^*_{U_i}(1)\,(1-q^{(j)}_X)\pi^{(j)}_X\Big) & \text{if } v_j=0\\ \max\Big(q_{V_jX}\prod_i\lambda^*_{U_i}(0)\prod_{k\neq j}\max(q_{V_kX}\pi^*_X(1),\pi^*_X(0)),\\ \qquad\prod_i\lambda^*_{U_i}(1)\,(1-q_{V_jX}q^{(j)}_{X1})\pi^{(j)}_X\Big) & \text{if } v_j=1\end{cases}$$

In the expression for $\tilde{\lambda}^*_X(v_j)$, $q^{(j)}_X$, $q^{(j)}_{X1}$, and $\pi^{(j)}_X$ are defined using the following expressions:

$$q^{(j)}_X=\prod_{V_k\neq j\in\pi_1}q_{V_kX}\prod_{V_k\neq j|\pi^*_X(v_k=0)\leq\pi^*_X(v_k=1)}q_{V_kX}$$

$$q^{(j)}_{X1}=\prod_{V_k\neq j|\pi^*_X(v_k=0)\leq\pi^*_X(v_k=1)}q_{V_kX}$$

$$\pi^{(j)}_X=\prod_{V_k\neq j\in\pi_1}\pi^*_X(1)\prod_{V_k\neq j\in\pi_0}\pi^*_X(0)\prod_{V_k\neq j\notin\pi_0\cup\pi_1}\max(\pi^*_X(1),\pi^*_X(0))$$

The boundary conditions for the childless and parentless nodes are as follows. (1) A childless unobserved node is assumed to receive message $\lambda^*_{U_0}=\{1,1\}$. (2) A childless node observed as 1 or 0 receives $\lambda^*_{U_0}=\{0,1\}$ or $\lambda^*_{U_0}=\{1,0\}$, respectively. (3) For a parentless node $q_X{=}0$, $\pi_X{=}P(X{=}1)$, and $\prod_{k\neq j}\max(q_{V_kX}\pi^*_X(1),\pi^*_X(0)){=}P(X{=}0)$.

The algorithm for computing MPE calculates $\tilde{\lambda}^*$ and $\tilde{\pi}^*$ for every network node traversing the graph starting from the observed symptom in the breadth-first order. A single traversal is repeated for every observed symptom. At the end, $\tilde{bel}^*$ values are computed for all network nodes. The MPE contains all *link* nodes with $\tilde{bel}^*(x=1)>\tilde{bel}^*(x=0)$.

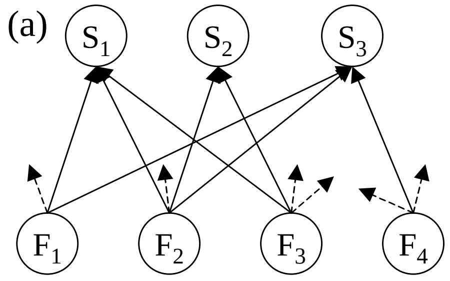
Algorithm 4 (Iterative MPE)

```
Inference iteration starting from node $Y_i$:
    let $o$ be the breadth-first ordering starting from $Y_i$
    for all nodes $X$ along ordering $o$ do
        compute $\tilde{\lambda}^*_X(v_j)$ for all $X$'s parents, $V_j$,
                  and for all $v_j \in \{0,1\}$
        compute $\tilde{\pi}^*_{U_i}(x)$ for all $X$'s children, $U_i$,
                  and for all $x \in \{0,1\}$
Symptom analysis phase:
    for every symptom $S_i \in \mathcal{S}_O$ do
        run inference iteration starting from $S_i$
    compute $\tilde{bel}^*(v_i)$ for every node $V_i$, $v_i \in \{0,1\}$
Fault selection phase:
    choose all link nodes with $\tilde{bel}^*(X=1) > \tilde{bel}^*(X=0)$
```

Local computations in *path* nodes require $\mathcal{O}(n^2)$ operations, where n is the maximum path length. This bound could be decreased to $\mathcal{O}(n)$ at the expense of significant complication of algorithm implementation and introducing a big constant making the performance gain difficult to observe in reasonably sized networks. Thus, in a single iteration, processing *path* nodes requires $\mathcal{O}(n^2(|\mathcal{S}|)\subseteq\mathcal{O}(n^4)$ operations. Similarly to belief updating, local computations in all *link* nodes require $\mathcal{O}(n^3)$ operations. We may conclude that a single iteration of the algorithm is $\mathcal{O}(n^4)$, and the complexity of the entire algorithm is $\mathcal{O}(|\mathcal{S}_o|n^4)\subseteq\mathcal{O}(n^6)$.

5.5 *Iterative hypothesis update*

In this section, we introduce a novel event correlation algorithm for calculating the best explanation of the observed symptoms. Contrary to the algorithms presented in Sections 5.1- 5.4, the technique we describe in this section creates a number of alternative fault hypotheses ranked using a belief metric. The algorithm proceeds iteratively and after every symptom observation it is able to output the set of the most probable hypotheses. The iteration triggered by the ith symptom, S_i, creates the set of hypotheses, $\mathcal{H}_i$, based on the set of hypotheses resulting from the previous iteration, $\mathcal{H}_{i-1}$, and the information about causal relationships between faults and symptoms stored in the belief network. Every hypothesis $h_j\in\mathcal{H}_i$ is a subset of $\mathcal{F}$, and is able to explain all symptoms in $\{S_1,\ldots,S_i\}$. We define H_{S_i} as set $\{F_k\in\mathcal{F}\}$ such that



(b) S_1: $H_{S_1} = \{ F_1, F_2, F_3 \}$

$\rightarrow \boldsymbol{H_1} = \{\{F_1\}, \{F_2\}, \{F_3\}\}$

S_3: $H_{S_3} = \{ F_1, F_2, F_4 \}$

$\rightarrow \boldsymbol{H_2} = \{\{F_1\}, \{F_2\}, \{F_3, F_4\}\}$

S_2: $H_{S_2} = \{ F_2, F_3 \}$

$\rightarrow \boldsymbol{H_3} = \{\{F_1, F_3\}, \{F_2\}, \{F_3, F_4\}\}$

Figure 5: Example of iterative hypothesis updating: (a) Example belief network; (b) Sets of hypotheses created after observing symptoms S_1, S_3, and S_2.

F_k may cause S_i, i.e., the belief network contains a directed path from F_k to S_i. The set of hypotheses $\mathcal{H}_i$ is created from $\mathcal{H}_{i-1}$ by incorporating the explanation, H_{S_i}, of the last observed symptom, S_i. Every hypothesis $h_j \in \mathcal{H}_i$ is minimal, i.e., if any fault $F_l \in h_j$ is removed from h_j, hypothesis h_j is no longer able to explain all the observed symptoms.

With every hypothesis $h_j \in \mathcal{H}_i$ we associate belief metric b_i, which similarly to the measure of goodness $g()$ in Algorithm 1 (Section 5.1) represents the probability that all faults belonging to h_j have occurred and that every observed symptom $S_k \in \{S_1, \ldots, S_i\}$ is explained by at least one of the faults in h_j. Formally, we define $b_i(h_j)$ as follows:

$$b_i(h_j) = \prod_{F_l \in h_j} P(F_l) \cdot \prod_{S_k \in \{S_1,\ldots,S_i\}} \Big(1 - \prod_{F_l \in h_j} P(1 - P(S_k|F_l))\Big)$$

To incorporate the explanation of symptom S_i into the set of fault hypotheses, in the i-th iteration of the algorithm, we analyze every hypothesis $h_j \in \mathcal{H}_{i-1}$. If h_j is able to explain symptom S_i, we put it into $\mathcal{H}_i$. The hypotheses in $\mathcal{H}_{i-1}$ that do not explain S_i have to be extended by adding to each of them a fault from H_{S_i}. One possible way to do that is to create a new hypothesis for every fault $F_l \in H_{S_i}$ and every hypothesis $h_j \in \mathcal{H}_{i-1}$ that does not explain S_i, by adding F_l to h_j. Unfortunately, this would result it the very fast growth of $\mathcal{H}_i$ and, in consequence, make the computational complexity of the algorithm unacceptable. Instead, we adopt the following heuristics. Fault $F_l \in H_{S_i}$ may be added to $h_j \in \mathcal{H}_{i-1}$ only if the size of h_j, $|h_j|$, is smaller than the size of any hypothesis in $\mathcal{H}_{i-1}$ that contains F_l and explains symptom S_i. The usage of this heuristics is derived from the fact that the probability of multiple simultaneous faults is small. Therefore, of any two hypotheses containing F_l the hypothesis that contains the fewest faults is more likely to constitute the optimal symptom explanation. Thus, since it is not efficient to keep all possible hypotheses, we remove the hypotheses that are bigger in size. In the following Algorithm 5, $\mu(F_l)$ denotes the minimum size of a hypothesis that contains fault F_l calculated over all hypotheses in the current hypotheses set.

Algorithm 5 (Iterative Hypothesis Update)

```
let $\mathcal{H}_0 = \{\emptyset\}$ and $b_0(\emptyset) = 1$
for every observed symptom $S_i$:
    let $\mathcal{H}_i = \emptyset$
    for all $F_l \in \mathcal{F}$ let $\mu(F_l) = |\mathcal{F}|$
    for all $h_j \in \mathcal{H}_{i-1}$ do
        for all $F_l \in h_j$ such that $F_l \in H_{S_i}$ do
            $\mu(F_l) = \min(\mu(F_l), |h_j|)$
        add $h_j$ to $\mathcal{H}_i$ and calculate $b_i(h_j)$
    for all $h_j \in \mathcal{H}_{i-1} - \mathcal{H}_i$ do
        for all $F_l \in \mathcal{F} \cap H_{S_i}$ such that $\mu(F_l) > |h_j|$ do
            add $h_j \cup \{F_l\}$ to $\mathcal{H}_i$ and compute $b_i(h_j \cup \{F_l\})$
choose $h_j \in \mathcal{H}_{|\mathcal{S}_o|}$ such that $b_{|\mathcal{S}_o|}(h_j)$ is maximum
```

We illustrate the algorithm with the following example. The fault model in Figure 5-(a) presents causal relationships between faults F_1, F_2, F_3, and F_4 and symptoms S_1, S_2, and S_3. Suppose the symptoms are observed in order S_1, S_3 and S_2. Figure 5-(b) presents the iterative creation of the hypothesis sets after every symptom observation. Initially, the only available hypothesis is $\emptyset$, which indicates that given no symptom observations we conclude that no faults occurred. Then, symptom S_1 arrives, whose explanation is $H_{S_1} = \{F_1, F_2, F_3\}$. We create extensions of the only available hypothesis, $\emptyset$, which does not explain S_1, for every fault in H_{S_1}. As a result, we obtain $\mathcal{H}_1 = \{\{F_1\}, \{F_2\}, \{F_3\}\}$. The explanation for symptom S_3 is $H_{S_3} = \{F_1, F_2, F_4\}$. Since, F_1 and F_2 belong to hypotheses $\{F_1\}$ and $\{F_2\}$ respectively, $\{F_1\}$ and $\{F_2\}$ are placed in $\mathcal{H}_2$ and both $\mu(F_1)$ and $\mu(F_2)$ are set to 1. Hypothesis $\{F_3\}$ does not explain S_3, therefore it has to be extended with faults in H_{S_3}. However, we cannot use F_1 and F_2 since their $\mu(.)$'s $\leq |\{F_3\}|$. The only extension possible is $\{F_3, F_4\}$. In the next iteration, after symptom S_2 has been observed, we are allowed to extend $\{F_1\}$ by adding fault F_3 since $\mu(F_3) = |\{F_3, F_4\}| = 2$ while $|\{F_1\}| = 1$, but we are not allowed to extend $\{F_1\}$ by adding fault F_2, because $\mu(F_2) = |\{F_1\}| = 1$.

The last problem to solve is the efficient computation of $b_i(h_j)$. We observe that $b_i(h_j)$ may be calculated iteratively based on $b_{i-1}(h_j)$ as follows:

1. If $h_j \in \mathcal{H}_i$ and h_j explains S_{i+1}

$$b_{i+1}(h_j) = b_i(h_j)\Big(1 - \prod_{F_l \in h_j \cap H_{S_{i+1}}} (1 - P(S_{i+1}|F_l)\Big)$$

2. Otherwise, if F_l explains S_{i+1}

$$b_{i+1}(h_j \cup \{F_l\}) = b_i(h_j) P(F_l) P(S_{i+1}|F_l)$$

To calculate the upper bound for the worst case computational complexity we observe that the calculation of $b_i(h_j)$ is $\mathcal{O}(|h_j \cap H_{S_i}|) \subseteq \mathcal{O}(|H_{S_i}|) \subseteq \mathcal{O}(n)$, since in an n-node net-

Table 1: Comparison of Algorithms 1- 5

Algorithm	*Theoretical bound*	*Detection rate*	*False positive rate*	*Max. network size with localization time <10s*	*Lost and spurious symptoms*	*Is algorithm iterative?*	*Prediction capabilities*
Naive Combinatorial (Alg. 1)	$\exp(n)$	96-99%	1-4%	20	yes	no	no
Bucket Elimination (Alg. 2)	$\exp(n)$	97-100%	0-3%	10	yes	no	yes
Iterative Belief Updating (Alg. 3)	n^5	94-98%	2-12%	50	yes	yes	yes
Iterative MPE (Alg. 4)	n^6	96-100%	0-8%	25	yes	yes	no
Iterative Hypothesis Updating (Alg. 5)	n^4	96-99%	1-3%	100	yes	yes	no

work a path may be composed of at most n links. The calculation of $b_i(h_j \cup \{F_l\})$ is $\mathcal{O}(1)$. The algorithm performs $|\mathcal{S}_o|$ iterations. In every iteration we execute two *for* loops. The first loop updates belief metric of all hypothesis that explain symptom S_i. For every hypothesis, h_j, it first recalculates $\mu(.)$'s of all faults in the hypothesis that could have caused symptom S_i ($\mathcal{O}(|H_{S_i}|)$ operations), and computes $b_i(h_j)$ ($\mathcal{O}(|H_{S_i}|)$ operations). Thus, the first loop requires $\mathcal{O}((\max_i(|\mathcal{H}_i|)|H_{S_i}|)$ steps. The second loop requires $\mathcal{O}(\max_i(|\mathcal{H}_i|)|H_{S_i}| \cdot 1)$ operations. Therefore the complexity of the entire algorithm is $\mathcal{O}(|\mathcal{S}_o| \max_i(|\mathcal{H}_i|)n)$. To get the precise bound we need to determine the bound for $\max_i(|\mathcal{H}_i|)$. It turns out that in rare cases the size of the hypothesis set may grow exponentially. To avoid this problem we set a limit on the number of hypotheses that may be created in each iteration; the least likely hypotheses are rejected when the limit is exceeded. The price we pay for this modification is that the best hypothesis is no longer guaranteed to be minimal. If the limit set on the size of the hypothesis set is $\mathcal{O}(n)$, operations involved in controlling the size of $\mathcal{H}_i$ do not increase the theoretical bound on the complexity of the entire algorithm. In the simulation study described in Section 6, we used the limit of $2n$. Thus, the complexity is $\mathcal{O}(|\mathcal{S}_o|n^2)$, i.e., $\mathcal{O}(|\mathcal{S}_o|n^2)$, and in the worst case it is $\mathcal{O}(n^4)$.

6 Simulation Study and Comparison of Algorithms

The algorithms presented in Section 5 were implemented in Java. We used JavaBayes [1] package to obtain an implementation of Algorithm 2. The algorithms were evaluated through a set of comprehensive experiments. As a real-life application domain, we chose the data link layer in a bridged network in which the path ambiguity is resolved using Spanning Tree Protocol [50]. As a result, the shape of the considered graphs is reduced to trees, thus making random generation of dependencies resembling real-life scenarios easier. We tested the algorithms on randomly generated network topologies, whose size ranged from 5 to 100 nodes for the most efficient algorithm, Algorithm 5. The high computation time of other algorithms made it infeasible to perform sufficient number of experiments with large graphs, not allowing to draw any sound conclusions with regard to the algorithms' accuracy and performance. We had to limit the scope of experiments to graphs of size $\leq$ 10, $\leq$ 20, $\leq$ 25, and $\leq$ 50 in the case of Algorithms 2, 1, 4, and 3, respectively (see Table 1).

For every graph size, we randomly generated spanning tree connections, link failure probabilities, and conditional probabilities on causal links between *link* and *path* nodes. The link failure probabilities were uniformly distributed random values of the order of 10^{-6}, and the conditional probabilities on causal links were uniformly distributed random values in the range $[0.5, 1)$. For every graph size, one hundred different graphs were generated.

For each randomly generated graph, we performed 200 experiments. In every experiment, we randomly generated the set of malfunctioning links, $\mathcal{F}_c$, based on their failure probabilities. Then, based on the conditional probabilities on causal links between *link* and *path* nodes, the set of observed symptoms, $\mathcal{S}_o$, resulting from the faults in $\mathcal{F}_c$ was generated. The observed symptoms were then randomly ordered.

The ordered set $\mathcal{S}_o$ was supplied as an input to the algorithms presented in Section 5. Their output, the set of detected faults, $\mathcal{F}_d$, was compared with $\mathcal{F}_c$. We used the following two metrics to represent the accuracy of the algorithms.

$$\text{detection rate} = \frac{|\mathcal{F}_d \cap \mathcal{F}_c|}{|\mathcal{F}_c|}$$

$$\text{false positive rate} = \frac{|\mathcal{F}_d - \mathcal{F}_c|}{|\mathcal{F}_d|}$$

In the above equations, *detection rate* represents the percentage of faults occurring in the network in a given experiment that were detected by an algorithm. *False positive rate* represents the percentage of faults proposed by an algorithm that were not occurring in the network in a considered experiment, i.e., they were false fault hypotheses. Table 1 shows detection rate and false positive rate intervals of the analyzed algorithms.

The results of the experiments were analyzed as a two-stage nested design [44] with graph size as the first stage, and graph shape and probabilistic distribution as the second stage factors. The analysis using standard F test [44] allowed us to determine that both factors have an impact on the algorithms' accuracy. While the dependency between the graph shape/probability distribution and accuracy is intuitive, that the graph size has an impact on the accuracy may seem surprising. In the following paragraphs, we explain the reasons for this dependency.

Figure 6 presents the relationship between detection rate and graph size. The mean for a particular graph size is an average over the mean detection rates for particular graphs of that size,

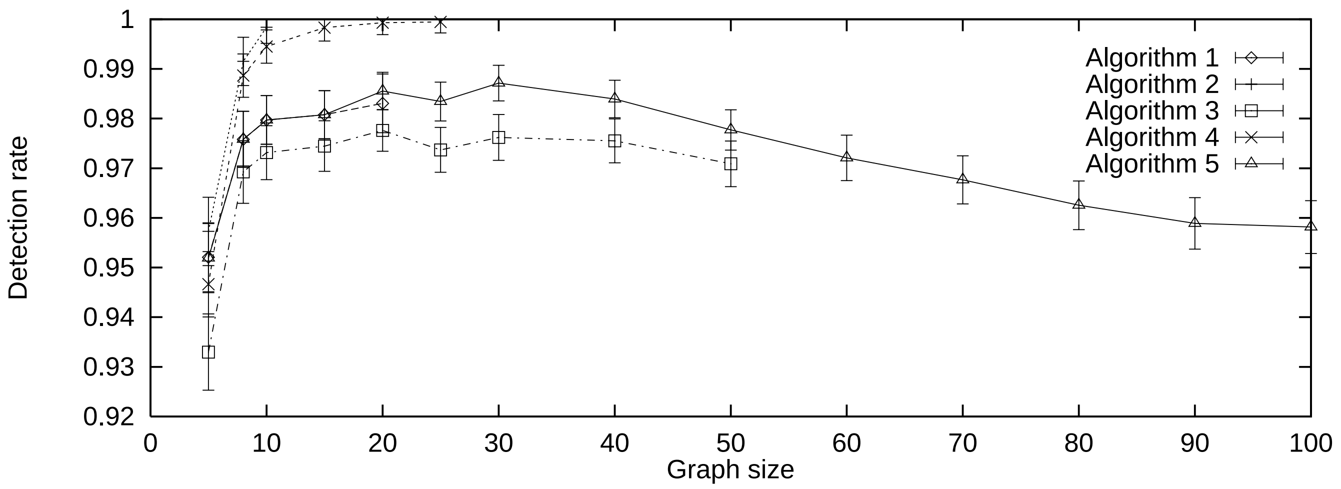


Figure 6: Comparison of accuracies achievable with algorithms presented in Section 5 for different network sizes

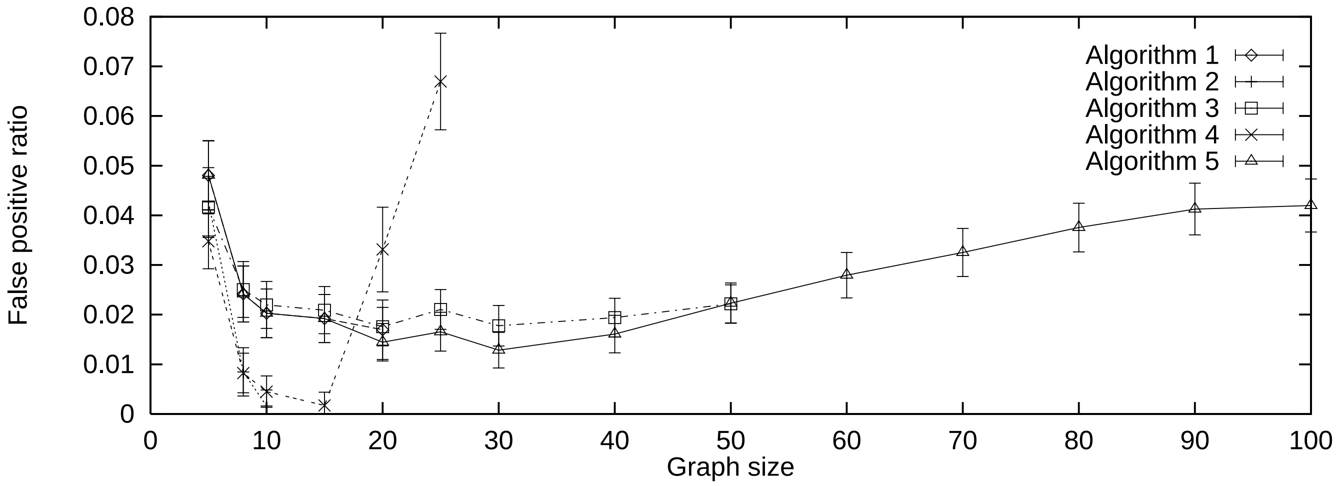


Figure 7: Comparison of false-positive metric values for algorithms presented in Section 5 using different network sizes

within statistically computed confidence intervals. We observe that Algorithms 2 and 4 slightly outperform other algorithms. In the analyzed graph size range, the difference is of the order of 1-2%.

The shape of the graphs in Figure 6 indicates a strong dependency of the detection rate on the graph size. The analysis of particular experiments shows, that for small (5-node) graphs, the number of symptoms observed is typically small (less than 10), which in some cases is not sufficient to precisely pinpoint the actual fault. Since in small graphs the size of $\mathcal{F}_c$ is also small, any mistake in fault detection significantly reduces the detection rate. When the graph gets bigger, the number of observed symptoms increases, thereby increasing the ability to precisely detect the faults. On the other hand, as the graph size grows, the multi-fault scenarios are getting more and more frequent. In multi-fault experiment, it is rather difficult to detect all actual faults, which leads to partially correct solutions. While the contribution of the partially correct solutions to the decrease of the detection rate is smaller in case of multi-fault experiments, the frequency of such partially correct solutions seems to cause the decrease of the detection rate observed in the case of Algorithm 5. Another reason behind the decreasing accuracy is the fact that in large networks the number of possible symptom explanations is bigger; if a sufficient number of symptoms is not observed, the algorithms are likely to choose a very likely, but not correct solution.

The gradual drop of the detection rate observed in the case of Algorithm 5 suggests that this drop may be asymptotic. One can also conclude that all analyzed algorithms have the very satisfactory detection rate of at least 95% (for graphs larger than 5 nodes).

Figure 7 presents the relationship between false positive rate and the graph size. The false positive rate for a particular graph size is calculated as a mean of average false positive rates for particular graphs of that size. Similarly to the detection rate metric, the false positive rates for Algorithms 2 and 4 in the analyzed graph size range are almost identical and slightly lower (better) than the false positive rates for Algorithms 1, 3, and 5. Interestingly, the false positive rate for Algorithm 4 starts to grow sharply when the graph size reaches 15. This observation, along with the shape of the detection rate curves for this algorithm presented in Figure 6, lets us conclude that Algorithm 4 tends to propose too broad solutions as the explanations of the observed symptoms. This phenomenon, which occurs in large networks, is caused by the increased frequency of multi-fault scenarios.

The false positive rate calculated for Algorithm 5 exhibits the gradual increase with the growth of the graph size. Similarly to the detection rate, the shape of the curve indicates that the growth may be asymptotic. If this was the case we could conclude that the false positive rate for Algorithm 5 does not exceed 4%. The false positive rate for Algorithm 4 reaches 8% in the tested range. Unfortunately, temporal complexity does not allow us to perform meaningful experiments to calculate the false positive rate values for Algorithm 4 in the wider graph size range.

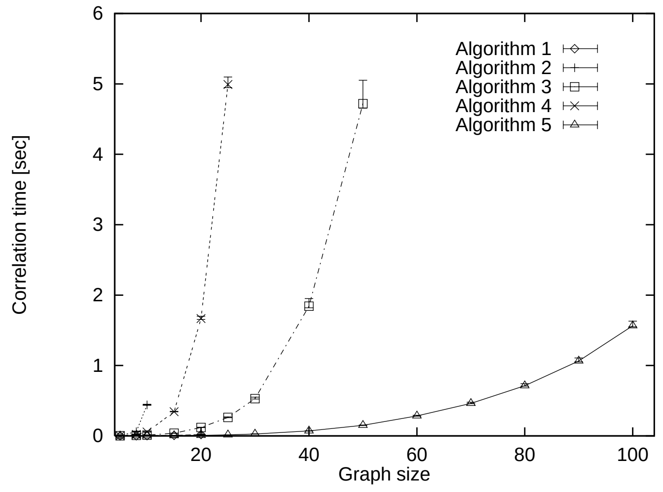


Figure 8: Comparison of single fault detection time for algorithms presented in Section 5 vs. network size

The next sequence of figures, Figures 8, 9, 10, and 11, present the dependency of the correlation time on the graph size in the presence of 1, 2, 3, and 4 network faults, respectively. The figures may be used to order Algorithms 2, 3, 4, and 5 with respect to their performance. Regardless of the number of faults occurring in the system, Algorithm 5 appears to be the most efficient, followed by Algorithm 3, Algorithm 4, and Algorithm 2 as the most time consuming. Algorithm 5 is at least an order of magnitude faster than any of the above algorithms.

Algorithm 1 is very efficient if only one fault occurs in the network. In this case, its correlation time is comparable to the correlation time of Algorithm 5 (Figure 8). However, already with two-fault scenarios (Figure 9) its temporal complexity seems to be closer to that of Algorithm 3 than Algorithm 5. Further performance degradation is observed with three fault scenarios (Figure 10) when the correlation time curve of Algorithm 1 overlaps with the curve of Algorithm 4. With four-fault scenarios, (Figure 11), for graphs of size up to 10, Algorithm 1 has the same correlation time as Algorithm 2. For bigger graphs, in the absence of correlation time results for Algorithm 2, Algorithm 1 is the most time consuming of all algorithms.

Figures 8, 9, 10, and 11 prove that the additional complexity

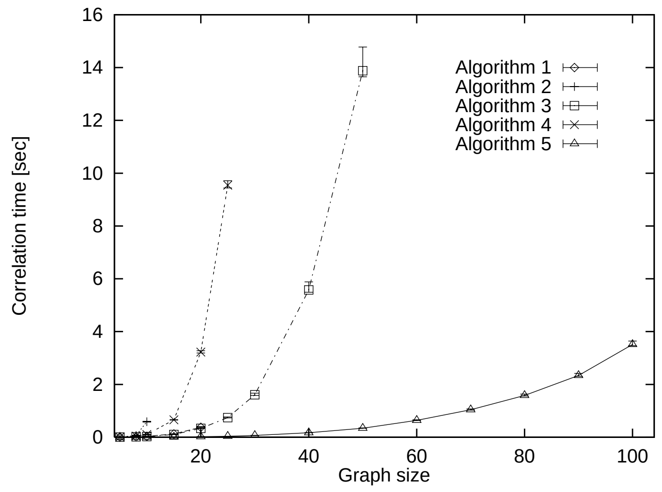


Figure 9: Comparison of correlation time for algorithms presented in Section 5 vs. network size in the presence of two network faults

involved in the design and implementation of Algorithms 3, 4, and 5 is justified by their greater efficiency, unless the multi-fault scenarios occur so rarely that their existence may be neglected. When multi-fault scenarios are likely, the naive combinatorial Algorithm 1 offers rather non-impressive performance.

In the tested graph size range, Algorithm 2 exhibited the best accuracy. However, the difference between the accuracy of Algorithm 2 and that of other algorithms is too small to justify the substantially worsened performance. Algorithm 5 proved to be the most efficient while also preserving very good accuracy. In Figure 12, its correlation time measured over the entire tested graph size range is presented. The correlation time of the order of several seconds even for large networks and multi-fault scenarios is very encouraging.

To make the evaluation of algorithms presented in Section 5 complete, one also needs to compare them with respect to other features. We believe that the following factors should be taken into account in this evaluation: (1) potential for dealing with lost and spurious symptoms, (2) ability to work in the event-driven environment and (3) usability for prediction and test planning. Comparison of algorithms with respect to these factors is presented in Table 1. Although the solution for dealing with lost and spurious symptoms is not described in this paper, all algorithms presented in Section 5 have a potential for working in an environment in which lost and spurious symptoms occur. This additional form of uncertainty may be embedded in the probability distribution and/or the graphical belief network model. Also, all presented algorithms, except Algorithms 1 and 2, are iterative and allow an event-driven building of fault hypotheses. Prediction of network symptoms based on other observed symptoms is possible with algorithms based on belief networks, i.e., Algorithms 2, 3, and 4. In addition, Algorithms 2 and 3 may be used to calculate the utility of tests that would check the existence of unobserved faults or symptoms, to allow optimizing the testing procedure.

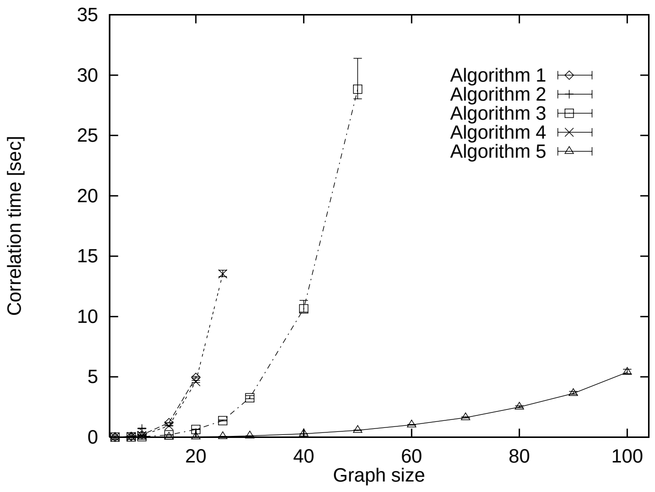


Figure 10: Comparison of correlation time for algorithms presented in Section 5 vs. network size in the presence of three network faults

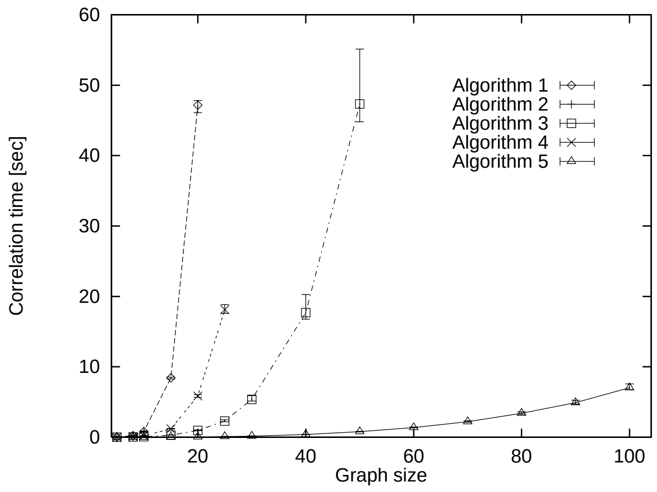


Figure 11: Comparison of correlation time for algorithms presented in Section 5 vs. network size in the presence of four network faults

7 Related Work

In the past, various event correlation techniques were proposed including rule-based systems [39, 60], model-based reasoning systems [28, 47], model traversing techniques [30, 32], case-based systems [38], fault propagation models [22, 34], and the code-book approach [61].

Rule-based systems are composed of rules (productions) of the form **if** *condition* **then** *action*. The condition part is a logical combination of propositions about the current set of received alarms and the system state [39, 60]; the action determines the state of correlation process. The operation of the system is controlled by an inference engine, which in fault management applications typically uses a forward-chaining inference mechanism [39, 47]. Rule-based systems are believed to lack scalability, to be difficult to maintain, and to have difficult to predict outcomes due to unforeseen rule interactions. The most frequently mentioned difficulty in using rule-based systems stems from the necessity of rewriting many rules when system design or implementation changes. Although approaches have been proposed to automatically derive correlation rules based on the observation of statistical data [35], it is still necessary to regenerate the large portion of correlation rules when the system configuration changes. The lack of structure in the system of rules typically makes it very difficult to allow reusability of rules that seems so intuitive in hierarchically built distributed systems.

Another group of approaches incorporate an explicit representation of the structure and function of the system being diagnosed. The representation provides information about dependencies between network components [27, 28, 30, 32, 34] or about cause-effect relationships between network events [22, 47]. The fault isolation process explores the network model to verify correlation between events. Model-based reasoning systems [28, 47] utilize inference engines controlled by a set of correlation rules, which contain model exploration predicates. Model-traversal techniques recursively search the dependency graph towards the failing object in an event-driven fashion starting from the component that symptoms refer to [27, 30, 32]. Fault propagation models [22, 34] provide heuristic symptom explanation algorithms aimed at satisfying some optimality criteria.

Event correlation systems based on a formal representation of network dependencies and structure represent an improvement over early rule-based systems by having the potential to solve novel problems, and by being more expandable. However, the models that they require are difficult to obtain and keep up-to-date. The computational complexity involved in model traversals limits the scalability of the fault isolation process. The approach presented in [3] avoids maintaining an explicit network model by providing scenario templates organized on a hierarchically based network structure, which are instantiated with the data obtained from the arriving event attributes or from the configuration database. In addition, the internal event publishers need not be aware which components consume the events that they forward; therefore, a change to higher-level scenario does not require changes to any of the lower level scenarios. One of the problems that the approach in [3] does not solve is dealing with complex network topologies. The solution shows how to propagate events between layers gradually increasing their level of abstraction. It does not, however, show how the reasoning should be performed within a layer if the network topology in this layer is complex.

The code-book technique [61] uses a network model to derive a code – a set of possible symptom observations for every problem that may occur in the network. This process, called code-book generation, is performed in advance upon the installations particular network topology. Code-book generation eliminates the runtime computational complexity involved in model traversals. Network alarms observed over a certain time window constitute a coded problem to be decoded using the code-book based on the minimum Hamming distance metric. The code-book technique is very efficient and is resilient to the noise in the alarm data. However, it is difficult to apply to the

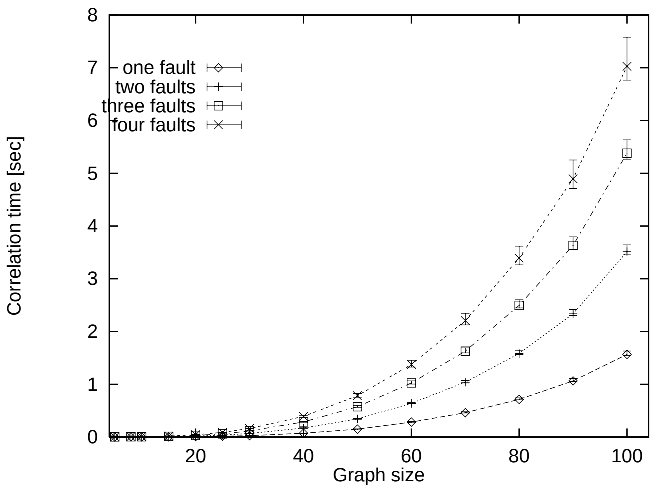


Figure 12: Fault localization time with Algorithm 5

correlation of transport and application layer events since relationship changes between managed objects, which are frequent in higher-layers, require reconfiguration of the code-book. It may be argued that fault-symptom mapping dictionaries are not suitable as a model for fault localization in a large, constantly changing enterprise network [54]. Additionally, multi-layer event correlation using a single correlation window is inadequate as events on different layers may have substantially different temporal relationships. Also, we can not perform tests or access or access information about the already detected failures during correlation.

Case-based systems [38] try to use experience gathered through past problem solving to find a solution for the new problem. The solution for the new problem is adapted form the solution of the closest matching problem solved in the past. Case-based systems are able to learn correlation patterns and are resilient to network configuration changes. They do not have a problem with network model maintenance. However, they do not take advantage of the known knowledge regarding entity behavior, nor do they allow fault isolation to be combined with fault detection. The need to build a substantial case library before the system is able to isolate faults makes case-based systems difficult to apply to an evolving or frequently changing architecture, such as server farms, or in the transport and application layers.

The common feature of most of the above approaches is that their reasoning is deterministic. This paper focuses on nondeterministic event correlation which is unavoidable in fault diagnosis related to quality of service degradation particularly in upper protocol layers. In nondeterministic fault model, alarm correlation aims at finding the most probable explanation of the observed alarms. This is an NP-hard problem [5, 34]. In the past, some research has been performed on finding appropriate heuristics to solve the problem in polynomial time. Katzela et al. [34] proposed an $\mathcal{O}(n^3)$ algorithm that finds the most probable explanation of a set of symptoms in an n-node dependency graph. The approach presented in [34] does not allow lost or spurious symptoms and the correlation may not be performed in event-driven fashion. They base the approach on the dependency graph in which every dependency graph node (terminal object) has only one mode of failure. The approach lacks a description presenting its possible application to real-life fault localization tasks. The solution presented in this paper addresses all the above issues achieving comparable computational complexity.

Kliger et al. [36] proposed a probabilistic model to be used with the codebook approach. Unfortunately, they do not present the non-deterministic decoding schema. We believe that the approach of Algorithms 3 and 4 can be used for this purpose.

The literature on event correlation contains reports of applying belief networks to fault diagnosis. However, the approaches are limited to rather narrow applications. In [17] a polynomial time algorithm for updating belief in a restricted Bayesian network used as a model for fault diagnosis in linear light-wave networks was proposed. In [24] belief networks have been applied to troubleshoot printing services. Other reported applications of Bayesian network theory to fault diagnosis include proactive fault detection [26]. The belief network used here is tree-shaped based on the structure of SNMP [8] MIB [41]. Wang et al. [59] applied Bayesian theory to identifying faulty links in communication networks. The analysis is performed based on connectivity information obtained by the management station through testing. The identification is done using *maximum a posteriori* method. Bipartite belief networks representing fault-symptom causal relationships were used in [9] to develop a hierarchical domain-oriented reasoning mechanism in the delegated management architecture. The proposed technique is able to pinpoint LAN segments suspected of having a particular fault.

Statistical data analysis methods were used for non-deterministic fault diagnosis in bipartite-graphs in [20]. The solution was proposed to detect link failures in wireless and/or battlefield networks.

8 Conclusions and Future Work

In this paper, we presented and evaluated several algorithms to perform fault localization using fault propagation models represented by bipartite graphs. We showed that exact algorithms are not only theoretically unacceptable, because of their exponential complexity bound, but they are also not usable in practice even for relatively small networks. Algorithms based on iterative message propagation (Algorithms 3 and 4) and iterative hypothesis updating (Algorithm 5) allow to find a solution efficiently in an event-driven fashion. In addition, Algorithm 5 builds the explanation incrementally, forming it after every symptom observation from the already existing explanation. All iterative algorithms, as revealed through an extensive simulation study, have very promising accuracy and performance.

There are a number of issues that need to be addressed in future work. So far, we have implicitly assumed that the set of observed symptoms is accurate, i.e., every symptom indicates a failure of the corresponding end-to-end service. In reality, spurious symptoms may occur, which do not indicate any abnormal condition. Moreover, the reasoning was performed based only on the negative information, i.e., observed end-to-end service failures. We did not take into account positive information that some end-to-end services did not fail. Confidence in the failure of a particular hop-to-hop service should be decreased if many of its resultant end-to-end service failures did not occur. Reasoning with positive feedback needs to take into account that some symptoms are not observed because of their loss rather than because there was no failure. It is rather straightforward to incorporate positive, lost, and spurious symptoms in the iterative algorithms presented in this paper. The impact of this additional form of uncertainty on the accuracy of the fault localization process remains to be investigated.

In our simulation study, we considered the case in which the conditional probability distribution represented by a belief network is known accurately. In general, only estimates of probability values may be known. We plan to investigate the impact of inaccuracy within these estimates on the fault localization process.

The algorithms presented in this paper were evaluated on a restricted class of network topologies. While we find no reason to believe that in arbitrary network topologies the performance or accuracy of these algorithms would be substantially different, we think that the algorithms should be evaluated also on arbitrary topologies resembling real-life networks.

In this paper, we considered the situation in which the routing information necessary to build a dependency model for end-to-end services is available. However, to obtain this information may be time consuming and require substantial amount of resources needed to install and run management agents on network devices, which collect the management information, and to regularly transmit the routing information over the network. Obtaining routing information may be particularly difficult if the management information is transmitted over the managed network, because the managed network outages, i.e., situations when there is a need for fault localization, may affect the ability to transmit the routing information. In future research, we would like to investigate diagnosing end-to-end service failures without access to the accurate routing information.

References

[1] http://www.cs.cmu.edu/ javabayes/Home/.

[2] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Océano – SLA-based management of computing utility. In *IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, WA. (to appear).

[3] K. Appleby, G. Goldszmidt, and M. Steinder. Yemanja – a layered event correlation engine for multi-domain server farms. In *IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, WA. (to appear).

[4] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo Codes. In *Proceedings of International Communications Conference*, pp. 1064–1070, 1993.

[5] A. T. Bouloutas, S. Calo, and A. Finkel. Alarm correlation and fault identification in communication networks. *IEEE Transactions on Communications*, 42(2/3/4):523–533, 1994.

[6] A. Brown, G. Kar, and A. Keller. An active approach to characterizing dynamic dependencies for problem determination in a distributed application environment. In *IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, WA. (to appear).

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A Simple Network Management Protocol (SNMP). RFC 1157, 1990.

[8] J. D. Case, K. McCloghrie, M. T. Rose, and S. Waldbusser. Protocol operations for version 2 of the simple network management protocol (SNMPv2). RFC 1905, IETF Network Working Group, 1996.

[9] C. S. Chao, G. L. Yang, and A. C. Liu. An automated fault diagnosis system using hierarchical reasoning and alarm correlation. In *IEEE Workshop on Internet Applications*, 1999.

[10] R. Comerford. The new software paladins. *IEEE Spectrum*, 37(6), Jun. 2000.

[11] G. F. Cooper. Probabilistic inference using belief networks is NP-Hard. Technical Report KSL-87-27, Stanford University, 1988.

[12] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer-Verlag, 1999.

[13] P. Dagum and M. Luby. Approximately probabilistic reasoning in Bayesian belief networks is NP-hard. *Artificial Intelligence*, pp. 141–153, 1993.

[14] R. Dechter. Bucket Elimination: A unifying framework for probabilistic inference. In E. Horvitz and F. V. Jensen, eds, *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, Portland, Oregon, Aug. 1996. Morgan Kaufmann Publishers.

[15] R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In D. Geiger and P. Shenoy, eds, *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, Brown University, Providence, Rhode Island, Aug. 1997. Morgan Kaufmann Publishers.

[16] E. Decker, P. Langille, A. Rijsinghani, and K. McCloghrie. Definition of managed objects for bridges. RFC 1493, 1993.

[17] R. H. Deng, A. A. Lazar, and W. Wang. A probabilistic approach to fault diagnosis in linear lightwave networks. In H. G. Hegering and Y. Yemini, eds, *Integrated Network Management III*, pp. 697–708. North-Holland, Apr. 1993.

[18] DoD. *Military Standard—Interoperability Standard for Digital Message Device Subsystems (MIL-STD 188-220B)*, Jan. 1998.

[19] C. Ensel. Automated generation of dependency models for service management. In *Workshop of the OpenView University Association (OVUA)*, Bologna, Italy, Jun. 1999.

[20] M. Fecko and M. Steinder. Combinatorial designs in multiple faults localization for battlefield networks. In *US Army Research Lab/ATIRP FedLab Symposium*, College Park, MD, Mar. 2001.

[21] R. Gopal. Layered model for supporting fault isolation and recovery. In *Proceedings of IEEE Network Operation and Management Symposium*, Honolulu, Hawaii, 2000.

[22] M. Hasan, B. Sugla, and R. Viswanathan. A conceptual framework for network management event correlation and filtering systems. In M. Sloman, S. Mazumdar, and E. Lupu, eds, *Integrated Network Management VI*, pp. 233–246. IEEE Publishing, May 1999.

[23] R. Hauck and H. Reiser. Monitoring of service level agreements with flexible and extensible agents. In *Workshop of the OpenView University Association (OVUA)*, Bologna, Italy, Jun. 1999.

[24] D. Heckerman and M. P. Wellman. Bayesian networks. *Communications of the ACM*, 38(3):27–30, Mar. 1995.

[25] A. Hiles. *Service Level Agreements : Managing Cost and Quality in Service Relationships*. Chapman and Hall, 1993.

[26] C. S. Hood and C. Ji. Proactive network management. In *Proceedings of IEEE INFOCOM*, Kobe, Japan, 1997.

[27] K. Houck, S. Calo, and A. Finkel. Towards a practical alarm correlation system. In A. S. Sethi, F. Faure-Vincent, and Y. Raynaud, eds, *Integrated Network Management IV*, pp. 226–237. Chapman and Hall, May 1995.

[28] G. Jakobson and M. D. Weissman. Alarm correlation. *IEEE Network*, pp. 52–59, Nov. 1993.

[29] D. Johnson and D. Maltz. *Dynamic Source Routing in Ad Hoc Wireless Networks*, chapter 5, pp. 153–181. Kluwer Academic Publishers, 1996.

[30] J. F. Jordaan and M. E. Paterok. Event correlation in heterogeneous networks using the OSI management framework. In H. G. Hegering and Y. Yemini, eds, *Integrated Network Management III*, pp. 683–695. North-Holland, Apr. 1993.

[31] G. Kar, A. Keller, and S. Calo. Managing application services over service provider networks. In *Proceedings of IEEE Network Operation and Management Symposium*, Honolulu, Hawaii, 2000.

[32] S. Kätker. A modeling framework for integrated distributed systems fault management. In C. Popien, ed, *Proceeding of the IFIP/IEEE International Conference on Distributed Platforms*, pp. 187–198, Dresden, Germany, 1996.

[33] S. Kätker and M. Paterok. Fault isolation and event correlation for integrated fault management. In A. Lazar, R. Saracco, and R. Stadler, eds, *Integrated Network Management V*, pp. 583–596. Chapman and Hall, May 1997.

[34] I. Katzela and M. Schwartz. Schemes for fault identification in communication networks. *IEEE Transactions on Networking*, 3(6), 1995.

[35] M. Klemettinen, H. Mannila, and H. Toivonen. Rule discovery in telecommunication alarm data. *Journal of Network and Systems Management*, 7(4):395–423, Dec. 1999.

[36] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo. A coding approach to event correlation. In A. S. Sethi, F. Faure-Vincent, and Y. Raynaud, eds, *Integrated Network Management IV*, pp. 266–277. Chapman and Hall, May 1995.

[37] F. R. Kschischang and B. J. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *Journal on Selected Areas in Communications*, 1998.

[38] L. Lewis. A case-based reasoning approach to the resolution of faults

in communications networks. In H. G. Hegering and Y. Yemini, eds, *Integrated Network Management III*, pp. 671–681. North-Holland, Apr. 1993.

[39] G. Liu, A. K. Mok, and E. J. Yang. Composite events for network event correlation. In M. Sloman, S. Mazumdar, and E. Lupu, eds, *Integrated Network Management VI*, pp. 247–260. IEEE Publishing, May 1999.

[40] K. McCloghrie, F. Baker, and E. Decker. IEEE 802.5 Station Source Routing MIB using SMIv2. RFC 1749, 1994.

[41] K. McCloghrie and M. Rose. Management information base for network mangement of tcp/ip-based internets: Mib-ii. RFC 1213, 1991.

[42] R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng. Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *Journal on Selected Areas in Communications*, 16(2):140–151, Feb. 1998.

[43] D. M. Meira and J. M. S.Nogueira. Modelling a telecommunication network for fault management applications. In *Proceedings of IEEE Network Operation and Management Symposium*, pp. 723–732, New Orleans, Louisiana, 1998.

[44] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 5th edition, 2000.

[45] F. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In K. B. Laskey and H. Prade, eds, *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, Stockholm, Sweden, Jul. 1999. Morgan Kaufmann Publishers.

[46] M. Novaes. Beacon: A hierarchical network topology monitoring system based in IP multicast. In A. Ambler, S. B. Calo, and G. Car, eds, *Services Management in Intelligent Networks*, no. 1960 in Lecture Notes in Computer Science, pp. 169–180. Springer-Verlag, 2000.

[47] Y. A. Nygate. Event correlation using rule and object based techniques. In A. S. Sethi, F. Faure-Vincent, and Y. Raynaud, eds, *Integrated Network Management IV*, pp. 278–289. Chapman and Hall, May 1995.

[48] K. Ohta, T. Mori, N. Kato, H. Sone, G. MAnsfield, and Y. Nemoto. Divide and conquer technique for network fault management. In A. Lazar, R. Saracco, and R. Stadler, eds, *Integrated Network Management V*, pp. 675–687. Chapman and Hall, May 1997.

[49] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.

[50] R. Perlman. *Interconnections, Second Edition: Bridges, Routers, Switches, and Internetworking Protocols*. Addison Wesley, 1999.

[51] S. Ramanathan, D. Caswell, and S. Neal. Auto-discovery capabilities for service management: An ISP case study. *Journal of Network and Systems Management*, 8(4):457–482, 2000.

[52] I. Rish and R. Dechter. On the impact of causal independence. Technical report, Dept. of Information and Computer Science, University of California, Irvine, 1998.

[53] I. Rish and M. Singh. A tutorial on infrence and learning in Bayesian networks. IBM Watson Research Center, August 2000.

[54] S. H. Schwartz and D. Zager. Value-oriented network management. In *Proceedings of IEEE Network Operation and Management Symposium*, Honolulu, Hawaii, 2000.

[55] C. Scott, P. Wolfe, and M. Erwin. *Virtual private networks*. O'Reilly, 2nd edition, 1999.

[56] W. R. Stevens. *TCP/IP Illustrated*, vol. I. Addison Wesley, 1st edition, 1995.

[57] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 2nd edition, 1996.

[58] Tivoli. *Netview for Unix: Administrator's Guide, Version 6.0*, Jan. 2000.

[59] C. Wang and M. Schwartz. Fault detection with multiple observers. *IEEE Transactions on Networking*, 1(1):48–55, Feb. 1993.

[60] P. Wu, R. Bhatnagar, L. Epshtein, M. Bhandaru, and Z. Shi. Alarm correlation engine (ACE). In *Proceedings of IEEE Network Operation and Management Symposium*, pp. 733–742, New Orleans, Louisiana, 1998.

[61] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High speed and robust event correlation. *IEEE Communications Magazine*, 34(5):82–90, 1996.