Reliability Engineering and System Safety 156 (2016) 134–147

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Algorithms for Bayesian network modeling and reliability assessment of infrastructure systems

Iris Tien [a,*], Armen Der Kiureghian [b,c]

[a] School of Civil and Environmental Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0355, USA
[b] Civil Engineering Emeritus, Department of Civil and Environmental Engineering, University of California, Berkeley, CA 94720-1710, USA
[c] American University of Armenia, Yerevan, Armenia

ARTICLE INFO

Article history:
Received 9 February 2015
Received in revised form
19 July 2016
Accepted 26 July 2016
Available online 2 August 2016

Keywords:
Bayesian networks
Systems modeling
Algorithms
Reliability assessment
Risk analysis
Infrastructure systems

ABSTRACT

Novel algorithms are developed to enable the modeling of large, complex infrastructure systems as Bayesian networks (BNs). These include a compression algorithm that significantly reduces the memory storage required to construct the BN model, and an updating algorithm that performs inference on compressed matrices. These algorithms address one of the major obstacles to widespread use of BNs for system reliability assessment, namely the exponentially increasing amount of information that needs to be stored as the number of components in the system increases. The proposed compression and inference algorithms are described and applied to example systems to investigate their performance compared to that of existing algorithms. Orders of magnitude savings in memory storage requirement are demonstrated using the new algorithms, enabling BN modeling and reliability analysis of larger infrastructure systems.

1. Introduction

Infrastructure systems are essential for a functioning society, from distributing the water we drink, to delivering the electricity we consume, to enabling transport of people and goods from source to destination points. Our nation's infrastructure, however, is aging and becoming increasingly unreliable with potentially severe consequences. Given a complex infrastructure network comprised of many interconnected components, system reliability analysis is required to identify the critical components and make decisions regarding inspection, repair, and replacement to minimize the risk of system failure.

The Bayesian network (BN) is a useful probabilistic tool for system reliability assessment. It is a graphical tool that offers a transparent modeling scheme, allowing easy checking of the model even by non-experts in systems analysis and probabilistic methods. In an environment where information about a system is evolving in time and is subject to uncertainty, BNs are able to update the reliability state of the system as new information, e.g., from observations, inspections, or repair actions, becomes available. Infrastructure systems are subject to high degrees of uncertainty, including discrepancies between initial design and construction, uncertain degradation of system components over time, and exposure to stochastic hazards. BNs provide the proper probabilistic framework to handle such information for engineering decision making.

A major obstacle to widespread use of BNs for system reliability assessment, however, is the limited size of the system that can be tractably modeled as a BN. This is due to the exponentially increasing amount of information that needs to be stored as the number of components in the system increases. This paper proposes a method to address this limitation.

The main contributions of this paper are novel compression and inference algorithms that enable the modeling of larger systems as BNs than has been previously possible. The paper is organized as follows: Section 2 provides a brief background on BNs, including the advantages of using BNs for system reliability analysis and the current limitations in BN modeling of large systems. Section 3 introduces the proposed compression algorithm for constructing and storing the conditional probability tables (CPTs) required by the BN. Section 4 describes the inference algorithm for system reliability analysis, which uses the compressed CPTs without decompressing them. Section 5 demonstrates the proposed algorithms through application to a test system. Results for memory storage and computation time are presented.

* Corresponding author.
E-mail address: itien@ce.gatech.edu (I. Tien).

http://dx.doi.org/10.1016/j.ress.2016.07.022
0951-8320/

2. Background

2.1. Methods for system reliability assessment, including Bayesian networks (BNs)

Over the years, many methods have been developed to assess system reliability. While not intended to be an exhaustive list, these include reliability block diagrams (RBDs), fault trees (FTs), event trees (ETs), binary decision diagrams (BDDs), and Bayesian networks (BNs). RBDs, FTs, and ETs are symbolic models showing the logical relationships between component states and system outcomes. These can be extended into BDDs, graphical models representing Boolean relationships between variables. General relationships between random variables can be modeled graphically using BNs.

RBDs are useful to show the components of a system and their relationships; however, they are not efficient for reliability analysis of complex infrastructure systems [17,20]. FTs have been used extensively in the nuclear industry [33]. They are constructed for a particular undesired system outcome; hence, a single FT cannot model all possible modes or causes of system failure [4]. ETs trace forward through a causal chain to assess the probability of occurrence of different system outcomes. The size of an event tree, however, can grow exponentially with the number of sequential events [22]. BDDs are useful for modeling Boolean functions, as they occur in system reliability analysis [1,7]. The number of nodes and paths in a BDD is exponential with the number of variables in the domain of the Boolean function [19]. For reliability analysis of an infrastructure system, this implies a BDD of exponentially increasing size as the number of components in the system increases.

A Bayesian network (BN) is a graphical model comprised of nodes and links. Each node represents a random variable and each link describes the probabilistic dependency between two variables. Each BN node is assigned a set of mutually exclusive and collectively exhaustive states. In our application, the nodes represent the states of the system components and the overall system performance, and the links describe the probabilistic dependencies between component and system performance. The reader is referred to texts such as [13] for further information on BNs.

As stated earlier, the capability of BNs for updating and handling of uncertain information and their graphical modeling representation makes them particularly well suited for reliability assessment of infrastructure systems under evolving states of information [30]. There are both exact and approximate methods for inference in BNs. These methods are applicable given a BN structure, to update probability assessments over the network in light of new information. For the case where system topology changes, as can occur in post-disaster scenarios, first a restructuring of the BN, then performing inference over the new BN is necessary. Approximate inference methods are generally sampling based, including importance sampling [23,35] and Markov chain Monte Carlo [11]. In theory, these methods converge to the exact solution for a sufficiently large number of samples. In practice, however, the rate of convergence is unknown and can be slow [27]. This is especially true when simulating events that are a priori unlikely. Exact inference methods are, therefore, preferred. The algorithm described in Section 4 is for exact inference.

2.2. Current limitations in BN modeling of large systems

The use of BNs for system reliability assessment has been limited by the size and complexity of the system that can be tractably modeled. Systems analyzed in previous studies have been small, typically comprised of 5–10 components. This includes studies on generating BNs from conventional system modeling methods, e.g., RBDs [14,32] and FTs [5]. Mahadevan et al. [16] demonstrate the ability of the BN to use system-level test data to update information at the component level. They note that the computational effort increases significantly with the number of system components. They introduce an approach characterized as "branch and bound," whereby events of relatively low probability are ignored, to apply the BN to larger systems. The example given, however, is for a system consisting of only 8 components, and the willful discarding of available information, leading to a subsequent loss of accuracy in the result, is not ideal.

Boudali and Dugan [6] use BNs to model the reliability of slightly larger systems, including a system of 16 components. However, the authors state that this "large number" of components makes it "practically impossible" to solve the network without resorting to simplifying assumptions or approximations. Clearly, even a system of 16 components is not enough to create a full model of many real-world infrastructures. Nielsen et al. [19] propose a method utilizing Reduced Ordered Binary Decision Diagrams (ROBDDs) to efficiently perform inference in BNs representing large systems with binary components. However, a troubleshooting model is considered, which includes a major assumption of single-fault failures, i.e., the malfunction of exactly one component causes the system to be at fault. In general, the number of paths in the ROBDD is exponentially increasing with the number of components. It is the single-fault assumption that bounds the size of the ROBDD. For general systems, including infrastructure systems, this single-fault assumption cannot be guaranteed. Therefore, the gains from using the ROBDD may not be applicable.

Finally, a topology optimization algorithm is proposed in Bensi et al. [3] to address the inefficiency of a converging BN structure as shown in Fig. 1. The authors develop a discrete optimization program to create a more efficient, chain-like BN model of the system based on survival- or failure-path sequences. The proposed optimization program, however, must consider the permutation of all component indices and, therefore, may become intractably large for large systems.

2.3. Conditional probability tables in construction of BN

The system size limitation arises due to the conditional probability tables (CPTs) that must be associated with each node in the BN. In the BN terminology, the CPT of a child node provides the probability mass function of the variable represented by that node given each of the mutually exclusive combinations of the states of the parent nodes. For an infrastructure system, the state of the system is dependent on the states of each of its constituent components, as shown in Fig. 1. The BN can include parent nodes of the components, as indicated by the dashed arrows, representing common hazards, characteristics, or demands among components. The focus of this study is on the system description

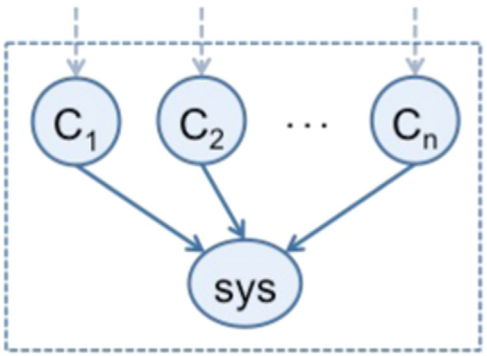

Fig. 1. BN of a system comprised of ***n*** components.

part of the BN, as indicated in the dashed box, which models system performance.

As shown in Fig. 1, the system node is a child node. Therefore, a CPT must be defined to give the probability of each system state for each combination of the states of the components. Let n denote the number of components and assume each component has m states. Each column of the CPT for each system state then has m^n elements. As an example, consider a binary system, where each component as well as the system are in one of two possible states: survival (denoted by 1) or failure (denoted by 0). For a binary system comprised of $n = 100$ components, the CPT column for the system survival state then consists of $2^{100}=1.3 \times 10^{30}$ 0 or 1 elements. This exponential increase in size poses a significant memory storage challenge in constructing and analyzing the BN, quickly rendering the model intractable.

Observe, however, that in general, the state of a system is deterministically known when the states of its constituent components are known. Thus, for a binary system, the system CPT has a special property: All entries in the table are either 0 or 1. The CPT associated with the system node in the BN that defines the probability of survival of the system given each distinct combination of the component states is shown in Table 1. Note that the component states in the CPT need not be stored, as long as they follow a predetermined pattern. The system CPT for a binary system is, therefore, composed of a single column of 0 s and 1 s

The focus of this paper is on binary systems. This is particularly useful for infrastructure systems, including power systems, where components are in one of two possible states, e.g., on or off, survival or failure. For systems characterized by flow across the network, e.g., water, gas, or transportation infrastructure, reliability assessments can also be conducted using an initial binary analysis. This is true if the component states are discretized, e.g., states 0, 1, 2, …, m, and the system performance is defined in a binary sense, e.g., the system is in a survival state if the flow at a particular consumption node is equal to or above a specified threshold, and it is in a failure state if the flow is below the threshold. Of course in this case a much larger combination of component states must be considered, but the system column in the CPT remains a column of 0 s and 1 s. In this paper, we present algorithms developed to take advantage of this property of the system CPT. Specifically, we develop an algorithm for representing the system CPT in a compressed form, and an algorithm to perform inference with compressed matrices.

2.4. Minimum cut set identification for compression algorithm

One of the inputs into our compression algorithm is the set of minimum cut sets (MCSs) of the system. An MCS is a minimum set of components whose joint failure constitutes failure of the system. While generation of the MCSs is an NP-hard problem, several efficient methods have been developed for their identification. These include enumerating MCSs one by one using recursive methods [2], including the recursive decomposition algorithm described in Li et al. [15]; a graphical approach using the connection matrix proposed by Suh and Chang [28]; and methods using a blocking mechanism to ensure every MCS is generated only once [24]. Methods for enumerating cut sets for k-out-of-n networks are proposed by Tan and Yeh [29,34], with the latter proposing a depth-first-search algorithm to determine the MCSs in a tree diagram. MCSs can also be generated directly from FTs, e.g., using the MOCUS (method for obtaining cut sets) algorithm [10,22].

Table 1
Conditional probability table for a binary system.

C_1	⋯	C_{n-1}	C_n	sys
0	⋯	0	0	0
0	⋯	0	1	0
0	⋯	1	0	1
0	⋯	1	1	0
⋮	⋮	⋮	⋮	⋮
1	⋯	0	0	0
1	⋯	0	1	1
1	⋯	1	0	1
1	⋯	1	1	1

3. Compression algorithm

The goal of the compression algorithm is to represent the system state column of length 2^n in a form that reduces the memory storage requirement. To do this, the proposed algorithm integrates two classical compression techniques, run-length encoding and Lempel-Ziv encoding. Both are lossless techniques so that no approximations are made in representing the system CPT. The algorithm will need to process through the full length of the data, however, in this case 2^n combinations of component states. From a computational standpoint, this can be prohibitive for large n. The main objective of the compression algorithm is to address the hard memory storage constraint in constructing BN models. Heuristics can be used to enhance computational efficiency of the algorithms.

3.1. Run-length encoding

In a set of data, a run is defined as consecutive bits of the same value. In run-length encoding, runs are stored as a data value and count [12]. For example, in compressing a sequence of white and black pixels, respectively denoted as "W" and "B," a sequence of 18 Ws is stored as the count 18 and the data value W. Thus, the memory storage requirement is reduced from 18 elements to 2. The number of bits required for storage of each element varies depending on the storage format. Run-length encoding is well suited for data with many repeated values. However, mixed values are stored literally, which results in little gain for mixed data. For example, alternating white and black pixels results in the compressed dataset 1W1B1W1B⋯, which in fact doubles the memory storage requirement compared to the uncompressed form WBWB ⋯. An example of run-length encoding is shown below:

Uncompressed dataset	WWWWWWWBWWWWWBBWWWWWWW WWW
Compressed dataset	7W1B5W2B10W

In this example, the original dataset comprised of 25 elements is compressed to a dataset of 10 total elements. Obviously, the gain achieved by employing run-length encoding to compress a dataset depends on the number and length of runs in the dataset.

3.2. Lempel–Ziv encoding

The classical Lempel–Ziv algorithm [36] finds patterns ("phrases") in the dataset, constructs a dictionary of phrases, and encodes based on repeated instances of phrases in the dictionary. The major advantage of the algorithm lies in the ability to repeatedly call the phrases in the dictionary, while having to store just one

instance of each phrase in the dictionary. An example of the Lempel–Ziv algorithm with the corresponding dictionary is shown below:

Uncompressed dataset	WWBWBBWBWWBB
Compressed dataset	∅ W1B2B2W3

Dictionary:

Phrase number	1	2	3	4
Full phrase	W	WB	WBB	WBW
Encoded as	∅W	1B	2B	2W

In constructing the dictionary, we begin with the empty set ∅. The dictionary is then dynamically constructed as we progress through the data. Moving through the uncompressed dataset, bit by bit, we read the input and find the longest string in the dictionary that matches it. We read the phrase number as part of the compressed output and add the bit that follows. If not present in the dictionary, the phrase is added to the dictionary as a new entry with an assigned number.

In the above dataset, our first bit is W, which we encode as the empty set ∅ plus a W. We then encounter another W, which we have seen before as phrase 1, and now append to that phrase the bit that follows, B. Thus, we create phrase 2 as phrase 1 (W) plus a B and add that to the dictionary. We then encounter another W; however, this is not the longest string in the dictionary that matches the current input. Instead, it is WB, which we have as phrase 2. Therefore, we encode these bits as phrase 2 (WB) plus the B that follows and add this to the dictionary as phrase 3. In a similar way, we encode WBW as phrase 2 (WB) plus a W and add to the dictionary as phrase 4. Finally, we encounter WBB, which has already been encoded as phrase 3. Thus, the end result of the compressed dataset is ∅W1B2B2W3.

The relatively short length of the above example dataset results in limited savings in the compression (9 elements to be stored instead of 12). However, as the size of the dataset and the number of repeated phrases increases, the memory storage gain achieved by the Lempel–Ziv algorithm also increases.

3.3. Proposed compression algorithm

The proposed algorithm integrates run-length and Lempel–Ziv encoding to compress the system column of the CPT. As described earlier, the system column of the CPT takes the values 0 or 1. A consecutive sequence of 0 s is a "0 run," and a consecutive sequence of 1 s is a "1 run." If the next bit in the sequence is a value different from the previous bit, e.g., a 1 following a sequence of 0 s, then that indicates either the beginning of a new run or the beginning of a phrase. If the bit subsequent to that last bit is the same value, i.e., a 1 following the previous 1, then that indicates the beginning of a 1 run. If the subsequent bit is a different value, i.e., a 0 following the previous 1, then that indicates the beginning of a phrase. The phrase is now comprised of at least two elements, the first value, i.e., the 1, and the differing second value, i.e., the 0. As the sequence continues with bits of the second value, the length of the phrase increases, until a bit of the first value appears, which then indicates the end of the phrase. Therefore, each phrase is constructed of two values: the first element of the phrase is one value and it is followed by a sequence of the other value. The dictionary for the proposed algorithm is comprised of these phrases.

Fig. 2 shows the flowchart for the developed compression algorithm, Algorithm A, for a binary system with n components and with $\{MCS\}$ denoting the set of minimum cut sets (MCSs) of the system. The output of the compression algorithm is the compressed system CPT column, $cCPT_{sys}$, and the accompanying dictionary of phrases, d_0. Steps of the algorithm are described below.

For each row $k = 1, \ldots, 2^n$ of the system CPT, the component states $s_1,\ldots,s_n$ are computed in terms of the row number based on the specific pattern used in defining the table. The table is constructed with $C_1,\ldots,C_n$ organized from left to right. Each row of the CPT is one of the mutually exclusive combinations of component states. The specific pattern used to organize these states is: C_1 is in state 0 for rows $k = 1, \ldots, 2^{n-1}$ and in state 1 for rows $k = 2^{n-1}+1, \ldots, 2^n$; C_2 is in state 0 for rows $k = 1, \ldots, 2^{n-2}$ and rows $2^{n-1}+1, \ldots, 2^{n-1}+2^{n-2}$ and in state 1 for rows $k = 2^{n-2}+1, \ldots, 2^{n-1}$ and rows $2^{n-1}+2^{n-2}+1, \ldots, 2^n$; etc. This pattern continues through C_n, which is therefore in states 0 and 1 in alternating rows, i.e., 0 in odd rows and 1 in even rows. Utilizing this pattern in constructing the CPT, we determine the state of component i, $i = 1, \ldots, n$, in row k of the CPT according to the rule

$$s_i = \begin{cases} 0 \text{ if } ceil\left(\frac{k}{2^{n-i}}\right) \in \text{odd} \\ 1 \text{if } ceil\left(\frac{k}{2^{n-i}}\right) \in \text{even} \end{cases} \quad (1)$$

where $ceil(x)$ is the value of x rounded up to the nearest integer. For example, for a system comprised of 20 components, the state of C_{15} in row 450,000 is 0 because $ceil\left(\frac{450000}{2^{20-15}}\right)$=14063 is odd. Using the above rule, the data on component states in the CPT are removed without any loss of information. For a system with multistate components, each component can have states 0,1, …,m_i. Given a pattern of listing component states, one can devise a more complicated formula that gives the states of all components for a given row number of the CPT. One can then determine the state of the system, 0 or 1, depending on whether the system flow is above or below a specified threshold. So everything remains the same, except that formula (1) is updated for multistate components.

For each row, the component states are checked against $\{MCS\}$ to determine the state of the system. If all of the components comprising at least one MCS are in the fail state, then the system is in the fail state; otherwise, the system is in the survival state. This is an operation that is required for all rows of the CPT, and thus must be performed 2^n times. Heuristics can be used in the component ordering to improve the efficiency of the MCS checking process. The resulting value of the system state is then encoded in compressed form as a run or a phrase.

Runs can be 0 runs or 1 runs, and as the number of consecutive repeated values increases, the length of the run is increased. When a phrase is encountered, it is checked against the contents of the dictionary to determine if it exists or is a new one that must be added to the dictionary. Each phrase in the dictionary is defined by four variables: (1) the phrase number, p, (2) the first value in the phrase, v_1, (3) the second and subsequent values in the phrase, v_2, and (4) the length of the phrase L_p. Once the existing or new phrase has been identified, the number of repeated instances of the phrase in sequence, denoted n_p, is updated.

Each row of the compressed CPT is comprised of three values: (1) an indicator variable that defines whether the row is the beginning of a run or a phrase; (2) if a run, the value r of the run; if a phrase, the phrase number p in the dictionary; and (3) if a run, the length L_r of the run; if a phrase, the number n_p of repeated instances of the phrase in sequence. Thus, a run is defined by the values $\{\text{run},r,L_r\}$ and a set of repeated phrases is defined by the values $\{\text{phrase},p,n_p\}$. Once all rows of the system CPT have been processed, the end result of the algorithm is the compressed CPT,

$cCPT_{sys}$, and the dictionary, d_0. The size of this data is typically orders of magnitude smaller than the size of the original CPT.

4. Inference algorithm

The objective in inference analysis in a BN is to update the conditional probability distribution of a "query" node for "evidence" (e.g., observed states) at other nodes. As mentioned earlier, our interest in this paper is in exact inference methods. Two major algorithms used for exact inference in BNs, the variable elimination (VE) algorithm and the junction tree (JT) algorithm, are briefly described below. This is followed by description of the inference algorithm developed by the authors to perform inference with the compressed CPT. First we describe the algorithms for the case of statistically independent states, then extend the developed algorithm to the case of dependent components.

Algorithm A. Compression Algorithm.

4.1. Variable elimination algorithm

In the VE algorithm [8,21], inference is performed by eliminating all nodes, one by one, until one is left with the query node – hence the name *variable elimination*. Elimination of each node corresponds to summing of the joint distribution over all states of the node, resulting in an intermediate factor λ that is used during the next step of elimination. For example, for the system in Fig. 1, suppose we are interested in the updated distribution of the state of component C_1, given a particular state *sys* of the system. Assuming component states are independent, the VE calculation for this query first computes

$$\begin{aligned} p(C_1, sys) &= \sum_{C_2} \cdots \sum_{C_n} p(C_1)p(C_2)\cdots p(C_{n-1})p(C_n)CPT_{sys} \\ &= p(C_1) \sum_{C_2} p(C_2)\cdots \sum_{C_{n-1}} p(C_{n-1}) \sum_{C_n} p(C_n)CPT_{sys} \\ &= p(C_1) \sum_{C_2} p(C_2)\cdots \sum_{C_{n-1}} p(C_{n-1})\lambda_n \ldots = p(C_1)\lambda_2 \end{aligned} \tag{2}$$

where CPT_{sys} is the CPT of the system node and λ_i is the intermediate factor, in the form of a table, created after the elimination of node C_i. The updated probability of interest is then obtained by dividing the joint probability by $p(sys)$, obtained by further elimination of C_1 in $p(C_1,sys)$, such that $p(C_1|sys)=p(C_1,sys)/p(sys)$. In the VE calculation, nodes $C_2,\ldots,C_n$ have been eliminated to arrive at the query node, C_1.

The intermediate factors λ_i need not be stored for subsequent steps in the node elimination process. Therefore, there is lesser demand on memory storage compared to other algorithms, such as the JT. However, when considering multiple queries or different evidence scenarios, need arises for repeated evaluation of some of the λ_i's, resulting in increased computation time. Furthermore, the order in which nodes in a network are eliminated results in different memory storage and computation time requirements. The selection of an optimal elimination order, however, is an NP-hard problem. Heuristics can be used in selecting the order of elimination of the nodes to improve the efficiency of the algorithm.

4.2. Junction tree algorithm

The JT algorithm [26] improves on the VE algorithm by breaking down the network into subsets of the nodes called "cliques." The cliques comprise the junction tree. The CPTs associated with cliques are called "potentials," and the JT is initialized by computing the potentials for all cliques. These potentials are stored and reused in cases of multiple queries or evidence cases, making the JT algorithm more efficient in computation time compared to the VE. However, large memory is required to store the clique potentials.

For a network structured as in Fig. 1, the junction tree is comprised of only one clique of size $n + 1$. As the size of the system increases, the potential over this clique grows exponentially in size and becomes impossible to store. Furthermore, with only one clique, there is no gain in using the JT versus the VE. Of course there are alternative ways of formulating the BN model of the system that are more favorable to the JT algorithm. Bensi et al. [3] discuss these formulations and propose a topology optimization method to reduce the clique sizes. Nevertheless, even with these formulations, the JT algorithm quickly becomes infeasible with increasing system size.

4.3. Proposed inference algorithm for independent components

The proposed inference algorithm is based on the VE method.

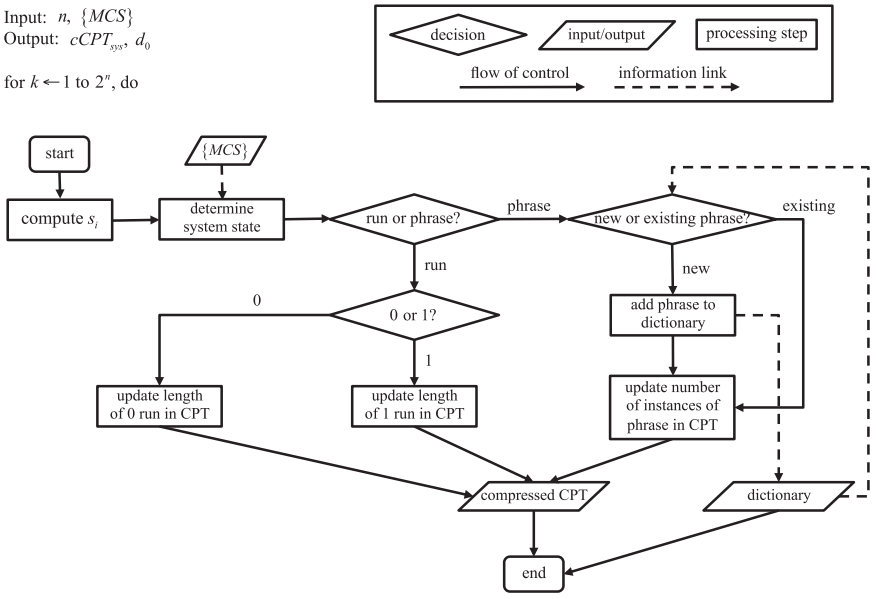

Fig. 2. Flowchart of compression algorithm.

Variables are eliminated one by one to arrive at the query node, with the intermediate factor λ_i created after elimination of component C_i. To reduce memory demand, λ_is are compressed using the same compression algorithm as for CPT_{sys}. Algorithm B described below constructs compressed λ_is using the rules presented in Tables 2 and 3 and performs inference with the compressed CPT_{sys} and λ_is without decompressing or recompressing them. In particular, λ_i is constructed directly from the compressed λ_{i+1} without decompressing the latter.

Algorithm B. Inference algorithm for independent components.

Input: n, $\{p_f\}$, $cCPT_{sys}$, d_0, Q, E
Output: Pr($Q|E$)
For $i \leftarrow n$downto1, do
If $i \in Q$
  Reorder λ_{i+1} such that component i is ordered to extreme left, i.e., numbered 1.
  $c\lambda_{i+1}^{\text{reordered}} \leftarrow \lambda_{i+1}^{\text{reordered}}$ using compression algorithm.
  Do not increment i.
Else for $j \leftarrow 1$tom_{i+1}, do
  Switch $\{$run,phrase$\}$, $S_{i+1}^j \in \{$odd,even$\}$, $\left(L_{r_{i+1}}^j \text{or} L_{p_{i+1}}^j\right) \in \{$odd,even$\}$.
  Construct $c\lambda_i$ and d_i according to the case rules in Tables 2,3.
  Increment i.
end

Let CPT_{sys} denote the conventional uncompressed system CPT, and $cCPT_{sys}$ the compressed one. Similarly, let $c\lambda_i$ represent the compressed version of the intermediate factor λ_i after elimination of the ith component, with $i = 0, n, \ldots, 1$ being the order of elimination. $i = 0$ indicates that no components have been eliminated. Therefore, $\lambda_0 = CPT_{sys}$ and $c\lambda_0 = cCPT_{sys}$. Let m_i indicate the number of rows in $c\lambda_i$.

During the variable elimination process according to Eq. (2), the values in the system column of CPT_{sys} are first multiplied by the state probabilities of component n to arrive at a column of values representing λ_n. The next step involves multiplication of this column with the state probabilities of component C_{n-1} and so on. These multiplications result in values in each column that, in general, are different from 0 or 1, even in the binary case. However, as each elimination step involves the multiplication of a specific component failure or survival probability (or a finite set of component state probabilities, when the component is multistate), the values in λ_i remain finite and the intermediate factor can be compressed. In fact, $c\lambda_i$ is constructed row by row, without ever constructing the uncompressed λ_i.

Similar to the rows of the compressed system CPT, each row j of $c\lambda_i$, $j = 1, \ldots, m_i$, is comprised of three values: (1) an indicator variable that defines whether the row is the beginning of a run or a phrase; (2) if a run, the value r_i^j of the run; if a phrase, the phrase number p_i^j in the dictionary; and (3) if a run, the length $L_{r_i}^j$ of the run; if a phrase, the number $n_{p_i}^j$ of repeated instances of the phrase. Thus, row j of $c\lambda_i$ is stored as either $c\lambda_i^j = \left\{\text{run}, r_i^j, L_{r_i}{}^j\right\}$ or $c\lambda_i^j = \left\{\text{phrase}, p_i^j, n_{p_i}^j\right\}$, depending on whether it is the beginning of a run or a phrase, respectively.

Recall that during compression of CPT_{sys}, an accompanying dictionary d_0 is constructed, which defines the phrases present in $cCPT_{sys}$. Each phrase is constructed of two values: The first element of the phrase is one of the values and it is followed by a sequence of the other value. This is because if the first value repeats, then

Table 2
Rules for constructing $c\lambda_i^j$.

switch			r_i^j or p_i^j	$L_{r_i}^j$ or $n_{p_i}^j$	R^j
run	$S_{i+1}^j \in$odd	$L_{r_{i+1}}^j \in$odd	r_{i+1}^j	$\left(L_{i+1}^j - 1\right)/2$	$r_{i+1}^j \times Pr\ (C_i=0)$
		$L_{r_{i+1}}^j \in$even	r_{i+1}^j	$L_{i+1}^j/2$	0
	$S_{i+1}^j \in$even	$L_{r_{i+1}}^j \in$odd	$r_{i+1}^j \times \Pr(C_i=1) + R^{j-1}$ (also $r_i^{j+1} = r_{i+1}^j$)	1 (also $L_{r_i}^{j+1} = \left(L_{i+1}^j - 1\right)/2$)	0
		$L_{r_{i+1}}^j \in$even	$r_{i+1}^j \times \Pr(C_i=1) + R^{j-1}$ (also $r_i^{j+1} = r_{i+1}^j$)	1(also $L_{r_i}^{j+1} = \left(L_{i+1}^j - 2\right)/2$)	$r_{i+1}^j \times Pr\ (C_i=0)$
phrase	$S_{i+1}^j \in$odd	$L_{p_{i+1}}^j \in$odd	p_{i+1}^j	$n_{p_{i+1}}^j$	$v_{2_{i+1}}^j \times Pr\ (C_i=0)$
		$L_{p_{i+1}}^j \in$even	p_{i+1}^j	$n_{p_{i+1}}^j$	0
	$S_{i+1}^j \in$even	$L_{p_{i+1}}^j \in$odd	p_{i+1}^j	$n_{p_{i+1}}^j$	0
		$L_{p_{i+1}}^j \in$even	p_{i+1}^j	$n_{p_{i+1}}^j$	$v_{2_{i+1}}^j \times Pr\ (C_i=0)$

Table 3
Updating d_i for a new phrase starting in row j of λ_i .

switch			$v_{1_i}^j$	$v_{2_i}^j$	$L_{p_i}^j$
phrase	$S_{i+1}^j \in$odd	$L_{p_{i+1}}^j \in$odd	$\left[v_{1_{i+1}}^j \times Pr\ (C_i=0)\right] + \left[v_{2_{i+1}}^j \times Pr\ (C_i=1)\right]$	$v_{2_{i+1}}^j$	$\left[\left(L_{i+1}^j - 3\right)/2\right] + 1$
		$L_{p_{i+1}}^j \in$even	$\left[v_{1_{i+1}}^j \times Pr\ (C_i=0)\right] + \left[v_{2_{i+1}}^j \times Pr\ (C_i=1)\right]$	$v_{2_{i+1}}^j$	$\left[\left(L_{i+1}^j - 2\right)/2\right] + 1$
	$S_{i+1}^j \in$even	$L_{p_{i+1}}^j \in$odd	$R^{j-1} + \left[v_{1_{i+1}}^j \times Pr\ (C_i=1)\right]$	$v_{2_{i+1}}^j$	$\left[\left(L_{i+1}^j - 1\right)/2\right] + 1$
		$L_{p_{i+1}}^j \in$even	$R^{j-1} + \left[v_{1_{i+1}}^j \times Pr\ (C_i=1)\right]$	$v_{2_{i+1}}^j$	$\left[\left(L_{i+1}^j - 2\right)/2\right] + 1$

the sequence would be a run. The dictionary is updated as new phrases are formed during the construction of $c\lambda_i$. Let d_i denote the updated dictionary after construction of $c\lambda_i$. d_i is constructed row by row. Each row of d_i defines a phrase, which is defined by four variables: (1) the phrase number p, (2) the first value in the phrase, v_1, (3) the second and subsequent values in the phrase, v_2, and (4) the length of the phrase L_p. In the proposed inference algorithm, we begin with $cCPT_{sys}$ and d_0. We then eliminate the components in the order n, …, 1. At each step, we construct $c\lambda_i$ from $c\lambda_{i+1}$ ($c\lambda_n$ from $c\lambda_0$=$cCPT_{sys}$) and d_i from d_{i+1} (d_n from d_0) based on the algorithm described below for the case of binary components. Note that once $c\lambda_i$ and d_i have been constructed, $c\lambda_{i+1}$ and d_{i+1} need not be retained and can be discarded.

Let $\{p_f\}$ be a vector of length n with elements $p_{f,i}=Pr(C_i=0)$, $i=1, \ldots, n$, where $p_{f,i}$ is the failure probability of component i. We define two quantities: S_i^j as the run or phrase start row number in λ_i defined by row j of $c\lambda_i$, and R^j the remainder after processing row j of $c\lambda_i$ (defined in Table 2). Because the remainder is reset after elimination of node i, it does not carry subscript i. Also, let Q define the set of query nodes and E denote the set of nodes for which evidence is entered. The algorithm for constructing $c\lambda_i$ then follows as described in Algorithm B.

Tables 2 and 3 show the rules for constructing $c\lambda_i$ and d_i, respectively. These are rules derived based on the mathematical operations required during variable elimination. Each row in Tables 2 and 3 represents a "switch" case. Given the values in each row of $c\lambda_{i+1}$, the algorithm switches to the appropriate row of the rule table to calculate the values for the next factor $c\lambda_i$. For example, if the row in the $c\lambda_{i+1}$ indicates a phrase that begins on an odd-numbered row in λ_{i+1} with an even phrase length, the rules in the sixth row of Table 2 and the second row of Table 3 would be used to construct $c\lambda_i$ and d_i.

Switch cases are shown in the left three columns of Tables 2 and 3. In Table 2 for $c\lambda_i$, the fourth and fifth columns list the run or phrase values, and the last column is an update of the remainder for row j. The case for a run with $S_{i+1}^j \in$even is special. It involves the remainder from the preceding row of $c\lambda_i$, as described below. The last column of the table lists the remainder value for the jth row of $c\lambda_i$. This is updated for possible use for the next row of $c\lambda_i$ or d_i. In Table 3 for d_i, the last three columns list the first and second values of the new phrase and its length. When $S_{i+1}^j \in$even, the first phrase value involves the remainder corresponding to the preceding row of $c\lambda_i$.

When a run starts on an even row number in λ_i, the first value of the run is multiplied by $\Pr(C_i=1)$, while subsequent pairs of values in the run are multiplied by $\Pr(C_i=0)$ and $\Pr(C_i=1)$, respectively. This first value must be taken into account as an additional run of unit length. Thus, for this special case, we concurrently construct rows j and $j+1$ of $c\lambda_i$. For the first run, $r_i^j=r_{i+1}^j \times \Pr(C_i=1)+R^{j-1}$, where R^{j-1} is the remainder from row $j-1$ of $c\lambda_i$ and $L_{r_i}^j=1$. For the second run, $r_i^{j+1}=r_{i+1}^j$ and $L_{r_i}^{j+1}=\left(L_{i+1}^j-1\right)/2$. Because one row of $c\lambda_{i+1}$ now corresponds to two rows in $c\lambda_i$, subsequent row numbers in $c\lambda_i$ are incremented by a 1, or if there are multiple instances x of this special case, the row numbers are incremented by x.

With the rules described in Tables 2 and 3, the proposed inference algorithm for independent components is able to handle both CPT_{sys} and λ_is in compressed form without decompressing or recompressing them. This enables significant savings in memory storage throughout the inference process.

4.4. Algorithm for dependent components

The inference algorithm presented in the previous section assumes independence between the states of the components of the system. In this section, we discuss the treatment of dependent component states, i.e., components with common parent nodes representing, for example, common capacities or demands.

Consideration of dependent component states requires modifications to the inference algorithm and the rules for constructing $c\lambda_i$ and d_i presented in the previous section. The algorithm developed and rules derived for the dependent case are given as Algorithm C and Tables 4 and 5, respectively.

Table 5
Updating $\boldsymbol{d_i}$ for a new phrase starting in row $\boldsymbol{j}$ of $\boldsymbol{c\lambda_i}$.

switch			$v_{1_i}^j$	$v_{2_i}^j$	$L_{p_i}^j$
phrase	$S_{i+1}^j \in$odd	$L_{p_{i+1}}^j \in$odd	$v_{1_{i+1}}^j+v_{2_{i+1}}^j$	$2 \times v_{2_{i+1}}^j$	$\left[\left(L_{i+1}^j-3\right)/2\right]+1$
		$L_{p_{i+1}}^j \in$even	$v_{1_{i+1}}^j+v_{2_{i+1}}^j$	$2 \times v_{2_{i+1}}^j$	$\left[\left(L_{i+1}^j-2\right)/2\right]+1$
	$S_{i+1}^j \in$even	$L_{p_{i+1}}^j \in$odd	$v_{1_{i+1}}^j+R^{j-1}$	$2 \times v_{2_{i+1}}^j$	$\left[\left(L_{i+1}^j-1\right)/2\right]+1$
		$L_{p_{i+1}}^j \in$even	$v_{1_{i+1}}^j+R^{j-1}$	$2 \times v_{2_{i+1}}^j$	$\left[\left(L_{i+1}^j-2\right)/2\right]+1$

Table 4
Rules for constructing $\boldsymbol{c\lambda_i^j}$ for dependent components.

switch			r_i^j or p_i^j	$L_{r_i}^j$ or $n_{p_i}^j$	R^j
run	$S_{i+1}^j \in$odd	$L_{r_{i+1}}^j \in$odd	$2 \times r_{i+1}^j$	$\left(L_{i+1}^j-1\right)/2$	r_{i+1}^j
		$L_{r_{i+1}}^j \in$even	$2 \times r_{i+1}^j$	$L_{i+1}^j/2$	0
	$S_{i+1}^j \in$even	$L_{r_{i+1}}^j \in$odd	$r_{i+1}^j+R^{j-1}$(also $r_i^{j+1}=2 \times r_{i+1}^j$)	1 (also $L_{r_i}^{j+1}=\left(L_{i+1}^j-1\right)/2$)	0
		$L_{r_{i+1}}^j \in$even	$r_{i+1}^j+R^{j-1}$ (also $r_i^{j+1}=2 \times r_{i+1}^j$)	1 (also $L_{r_i}^{j+1}=\left(L_{i+1}^j-2\right)/2$)	r_{i+1}^j
phrase	$S_{i+1}^j \in$odd	$L_{p_{i+1}}^j \in$odd	p_{i+1}^j	$n_{p_{i+1}}^j$	$v_{2_{i+1}}^j$
		$L_{p_{i+1}}^j \in$even	p_{i+1}^j	$n_{p_{i+1}}^j$	0
	$S_{i+1}^j \in$even	$L_{p_{i+1}}^j \in$odd	p_{i+1}^j	$n_{p_{i+1}}^j$	0
		$L_{p_{i+1}}^j \in$even	p_{i+1}^j	$n_{p_{i+1}}^j$	$v_{2_{i+1}}^j$

Algorithm C. Inference Algorithm for Dependent Components.

Input: n, $\{p_f\}$, $cCPT_{sys}$, d_0, Q, E
Output: $Pr(Q \mid E)$
For $i \leftarrow 1$ to n, do
  If $C_i \in \hat{C}$
  Calculate the marginalized component failure probability using Eq. (3).
end
For $j \leftarrow 1$ to n_P, do
  If $E \in \hat{C}_j$
  Update probability distribution of P_j using Eq. (4).
end
For $j \leftarrow n_P$ down to 1, do
  If $j = n_P$
  Eliminate P_j according to Eq. (5).
  $c\lambda_{P_i} \leftarrow \lambda_{P_i}$ using compression algorithm.
Else
  Eliminate P_j according to Eq. (6).
  $c\lambda_{P_i} \leftarrow \lambda_{P_i}$ using compression algorithm.
end
For $i \leftarrow n$ down to 1, do
  If $C_i \in \hat{C}$
  If $C_i = \max(\hat{C})$
  Eliminate C_i according to Eq. (7).
  $c\lambda_i \leftarrow \lambda_i$ using compression algorithm.
Else
  Construct $c\lambda_i$ and d_i according to the case rules in Tables 4 and 5.
Else
  Eliminate component using standard VE inference algorithm.
end

Let $P_1,\ldots,P_{n_P}$ denote the parent nodes, where n_P denotes the total number of parents of the n components, $C_1,\ldots,C_n$. Also let $\{p_f\}$ denote prior probabilities of failure of the components, including conditional failure probabilities of components with parents. Note that this is a set of tables, as the conditional component failure probabilities must be defined for all possible combinations of the parent states. To perform inference on query Q given evidence E for a system with dependent components, we employ the following algorithm.

Let n_i denote the number of parents of component C_i and define $P_{i,1},\ldots,P_{i,n_i}$ as the subset of nodes that are parents to C_i. First, we obtain the marginalized prior probability of failure of the component by summing over all states of the parent nodes:

$$p(C_i) = \sum_{P_{i,1}} \cdots \sum_{P_{i,n_i}} p(C_i \mid P_{i,1},\ldots,P_{i,n_i}) p(P_{i,1}) \cdots p(P_{i,n_i}) \tag{3}$$

If the evidence is on a component with parent nodes, we update the probability distributions of the parent nodes using Bayes' rule:

$$p(P \mid C_i) = \frac{p(C_i \mid P) p(P)}{p(C_i)} \tag{4}$$

During elimination of component nodes, if a component does not have any parents, variable elimination is performed according to Algorithm B. If the component has parents, we first eliminate the parents. Let ν_j denote the number of components that share a common parent, P_j, so that $\hat{C}_j = \{C_{j,1},\ldots,C_{j,\nu_j}\}$ is the subset of components that are the children of P_j. The parent nodes are eliminated in order $P_{n_P},\ldots,P_1$ with the parent node P_j eliminated by summing over all its states according to the total probability rule. The elimination of the first parent in the elimination order, P_{n_P}, creates the intermediate factor $\lambda_{P_{n_P}}$,

$$\lambda_{P_{n_P}} = \sum_{P_{n_P}} p\left(C_{n_P,1} \mid P_{(n_P,1),1},\ldots,P_{(n_P,1),n_{n_P,1}}\right) \cdots p\left(C_{n_P,\nu_{n_P}} \mid P_{(n_P,\nu_{n_P}),1},\ldots P_{(n_P,\nu_{n_P}),n_{n_P,\nu_{n_P}}}\right) p(P_{n_P}) \tag{5}$$

This factor, $\lambda_{P_{n_P}}$, is a function of all of the children nodes of P_{n_P}, as well as of all the parents of the children, i.e., $\lambda_{P_{n_P}}(C_{n_P,1},\ldots,C_{n_P,\nu_{n_P}}, P_k)$ where $k = \{[(n_P,1),1],\ldots,[(n_P,1), n_{n_P,1}],\ldots,[(n_P,\nu_{n_P}),1],\ldots,[(n_P,\nu_{n_P}), n_{n_P,\nu_{n_P}}]\}$. $P_{(n_P,1),1}$ indicates the first parent of the first child of parent n_P, $P_{(n_P,1),n_{n_P,1}}$ indicates the n^{th} parent of the first child of parent n_P, etc.

Subsequent elimination of a parent node P_j involves the conditional probability terms of the children of P_j, $C_{j,1},\ldots,C_{j,\nu_j}$; the intermediate factor from the previous elimination step, $\lambda_{P_{j+1}}$; and the marginal probability term of the node P_j. If a child of P_j, $C_{j,i}$, is also a child of a parent $P_{j+1},\ldots,P_{n_P}$, then the conditional probability term $p(C_{j,i} \mid P_{(j,i),1},\ldots P_j,\ldots,P_{(j,i),n_{j,i}})$ has already been accounted for in a previous elimination step, and does not enter into this elimination calculation. That is, only if $\{P_{j+1},\ldots,P_{n_P}\} \notin \{P_{(j,i),1},\ldots,P_{(j,i),n_{j,i}}\}$ is the term $p(C_{j,i} \mid P_{(j,i),1},\ldots P_j,\ldots,P_{(j,i),n_{j,i}})$ taken into account in the elimination of P_j. In addition, $\lambda_{P_{j+1}}$ is only taken into account if it is a function of the node being eliminated P_j, i.e., $j \in k = \{[(j+1,1),1],\ldots,[(j+1,1), n_{j+1,1}],\ldots,[(j+1,\nu_{j+1}),1],\ldots, [(j+1,\nu_{j+1}), n_{j+1,\nu_{j+1}}]\}$ for $\lambda_{P_{j+1}}(C_{j+1,1},\ldots,C_{j+1,\nu_{j+1}}, P_k)$. Otherwise, the intermediate factor $\lambda_{P_{j+1}}$ is moved to the next elimination step. Therefore, elimination of a parent node P_j is performed according to

$$\lambda_{P_j} = \sum_{P_j} \left\{ \left[\prod_{i=1}^{\nu_j} p\left(C_{j,i} \mid P_{(j,i),1},\ldots,P_j,\ldots,P_{(j,i),n_{j,i}}\right) \right] \lambda_{P_{j+1}} p(P_j) \right\} \tag{6}$$

for $C_{j,i}$ such that $\{P_{j+1},\ldots,P_{n_P}\} \notin \{P_{(j,i),1},\ldots,P_{(j,i),n_{j,i}}\}$, and $j \in k$ for $\lambda_{P_{j+1}}(C_{j+1,1},\ldots,C_{j+1,\nu_{j+1}}, P_k)$. In order to preserve the memory storage savings throughout the inference process, these intermediate factors are stored in compressed form using the compression algorithm described earlier.

Next, the components are eliminated in the order $C_n,\ldots,C_1$. The final intermediate factor created after the elimination of the parent nodes, λ_{P_1}, is a function of the components with parents, $\hat{C}$. This factor must be taken into account when eliminating the components, and will occur at the first instance of eliminating a component with parents. Since the components are eliminated in reverse.

order, this is the component with the highest index in $\hat{C}$. If $C_i = \max(\hat{C})$, the elimination of C_i is performed according to

$$\lambda_i = \sum_{C_i} \lambda_{P_1} \lambda_{i+1} \tag{7}$$

Note that if $i = n$, then λ_{i+1}, the intermediate factor from the previous elimination step, is equivalent to λ_{P_1}, and should not be counted twice. Therefore, in that case, Equation (7) becomes $\lambda_i = \sum_{C_i} \lambda_{P_1}$. Once the parent intermediate factor has been taken into account, the elimination of the remaining components with

parents only requires a summing over the states of C_i, i.e., $\lambda_i = \sum_{C_i} \lambda_{i+1}$.

Rules for constructing $c\lambda_i$ as well as the accompanying dictionary d_i are shown in Tables 4 and 5, respectively. Similar to the inference algorithm described previously, these rules are developed according to switch cases: run or phrase; the run or phrase start-row number S^j_{i+1} in λ_{i+1} defined in row j of $c\lambda_{i+1}$ being odd or even; and the length of the run or phrase, $L^j_{r_{i+1}}$ or $L^j_{p_{i+1}}$, being odd or even. In addition, as before, the case of a run with $S^j_{i+1} \in$ even is special and one row of $c\lambda_{i+1}$ corresponds to two rows in $c\lambda_i$, with the values as indicated in Table 4. The variable elimination of all components continues until the query components Q are reached. The full algorithm to perform inference on a system with dependent components is as given in Algorithm C.

5. Test example

To illustrate the proposed compression algorithm, we apply it to the example system shown in Fig. 3, which is adopted from [3]. The system consists of a parallel subsystem (C_1, C_2, C_3) and series subsystems (C_4, C_5, C_6) and (C_7, C_8). Component states are assumed to be statistically independent. For this system, the set of MCSs is $\{MCS\} = \{(C_1, C_2, C_3, C_4), (C_1, C_2, C_3, C_5), (C_1, C_2, C_3, C_6), (C_7), (C_8)\}$.

The full system CPT is as shown in Table 1 for $n = 8$. The first 8 columns give the states of components $C_1,\ldots,C_8$, constructed according to the pattern described in Section 3.2. The right-most column gives the state of the system given the states of the components in that row. Because the system is comprised of 8 components, the system CPT consists of 2^8=256 rows.

5.1. Application of compression algorithm

We proceed with implementing Algorithm A to compress the system CPT of the example system In row $k = 1$, $\{s_1,\ldots,s_8\} = \{0, \ldots, 0\}$. As all components in the system are in the failed state, clearly the system is also in the failed state. More rigorously, checking against $\{MCS\}$, we see that all components in the first MCS, (C_1, C_2, C_3, C_4), have failed. Therefore, $sys = 0$. Note that once we find that all components in a given MCS are in the failed state, we need not check the component states against the remaining MCSs as the failure of any one MCS indicates failure of the system.

As we continue through the rows $k = 2, \ldots, 31$, we find the system to be in the failed state, until we reach row $k = 32$. Therefore, the compressed CPT begins with a 0 run of length 31. At $k = 32$, $\{s_1,\ldots,s_8\} = \{0,0,0,1,1,1,1,1\}$. Checking against $\{MCS\}$, we see that none of the MCSs have failed. In fact, the path from source to sink connects through C_4, C_5, C_6, C_7, C_8. Thus, $sys = 1$, and we have reached the end of the initial 0 run. Looking at $k = 33$, $\{s_1,\ldots,s_8\} = \{0,0,1,0,0,0,0,0\}$. We see that the component in the fourth MCS (C_7) has failed. Therefore, $sys = 0$ and we have a phrase beginning at row $k = 32$. Had the value in row $k = 33$ been $sys = 1$, we would have had a 1 run.

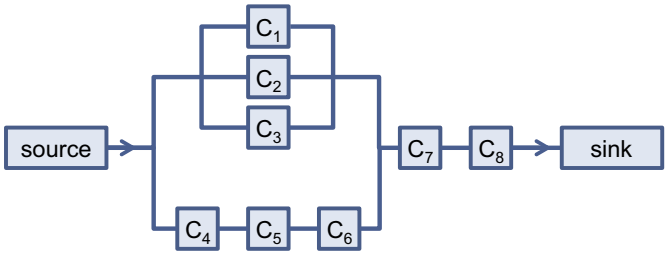

Fig. 3. Example system.

At this point, we have not yet encountered any phrases and our dictionary is empty. Therefore, this is the beginning of a new phrase. For $k = 34$ and $k = 35$, $sys = 0$. At $k = 36$, $\{s_1,\ldots,s_8\} = \{0,0,1,0,0,0,1,1\}$ and $sys = 1$, indicating the end of the phrase. The full phrase, therefore, is $\{1,0,0,0\}$ for $k = 32, \ldots, 35$, which is stored in the dictionary as $\{1,1,0,4\}$. These four values indicate that the phrase number is $p = 1$, the first value in the phrase is v_1=1, the second and subsequent values in the phrase are v_2=0, and the length of the phrase is L_p=4. Note that in the binary case, v_1 and v_2 are complements, so knowing one value enables us to know the other. However, while it is not necessary to store both values in the initial construction of the compressed system CPT, in the subsequent process for inference, v_1 and v_2 can take values different from 0 and 1 and are no longer guaranteed to be complements. Therefore, during inference, we must store both v_1 and v_2.

After determining $sys = 1$ for $k = 36$, we continue and find that for $k = 37, \ldots, 39$ the system state is $sys = 0$ and for $k = 40$ the system state $sys = 1$; thus we again encounter the phrase $sys = \{1,0,0,0\}$. This is now an existing phrase that we call from the dictionary. Therefore, in the compressed system CPT, we reference phrase $p = 1$ and increase the number of instances of this phrase by 1. We continue through the remaining rows $k = 40, \ldots, 2^n$=256. Once we have processed all the rows of the system CPT, the end result for the compressed CPT and the dictionary are as given in Tables 7 and 8, respectively.

Tables 7 and 8 indicate that the total number of elements to be stored for the compressed system CPT is 9 and for the dictionary is 4 for a total of 13 elements. We can verify that the number of rows represented in the compressed CPT equals the number of rows of the full CPT: We start with a run of length 31, have 56 instances of a phrase of length 4, and end with a run of length 1. This gives $31 + 56 \times 4 + 1 = 256$, which equals the number of rows of the

Table 6
Compressed system CPT, $\boldsymbol{cCPT_{sys}}$, for the example system.

run or phrase	r or p	L_r or n_p
run	0	31
phrase	1	56
run	1	1

Table 7
Dictionary, $\boldsymbol{d}_0$, for the example system.

p	v_1	v_2	L_p
1	1	0	4

Table 8
Uncompressed intermediate factor λ_8 after elimination of $\boldsymbol{C}_8$ in the example system.

C_1	…	C_6	C_7	λ_8
0	…	0	0	0
0	…	0	1	0
0	…	1	0	0
0	…	1	1	0
…		…	…	…
1	…	0	0	0
1	…	0	1	0.99
1	…	1	0	0
1	…	1	1	0.99

original CPT. Thus, the total number of elements to be stored has been reduced from 256 to 13. Note that the compression is lossless.

The pattern of 0 s and 1 s in the system CPT column is governed by the system topology. Therefore, the amount of compression achieved by Algorithm A varies from system to system. The least favorable scenario is when the sequence of 0 s and 1 s forms single instances of short phrases, as in {0,0,1,0,1,0,0,1,0,1,0,0,1,0,1…}, such that each appearance of a phrase requires another entry in the compressed matrix. However, simple modifications of the proposed algorithm can be used to achieve improved performance across topologies. For the above example, a 512-length sequence of this nature would require 615 elements in the compressed CPT, as well as 8 elements in the dictionary. However, by combining the two phrases {1,0} and {1,0,0} into a single repeated phrase {1,0,1,0,0} the compressed CPT will only have 6 elements. One way to automate this process of combining phrases is by running the compression algorithm through the first-pass compressed data a second time. This time, no runs will be identified. However, the algorithm will identify 102 repetitions of the new phrase {1,0,1,0,0}, which is created by a concatenation of the original {1,0} and {1,0,0} phrases.

5.2. Application of inference algorithm

We now illustrate the proposed inference algorithm for the example system. We define the prior probabilities of failure as 0.2 for the components in parallel and 0.01 for the components in series so that $\{p_f\}=\{0.2,0.2,0.2,0.01,0.01,0.01,0.01,0.01\}$. We begin with the compressed system CPT in Table 6 and the accompanying initial dictionary in Table 7. Suppose we are interested in the backward inference problem of obtaining the posterior probability distribution of C_1 given that the system has failed. The order of elimination of components is 8, …, 2.

In the first step, we eliminate C_8. For reference, the uncompressed intermediate factor λ_8 created after this elimination is shown in Table 8. To construct the compressed factor $c\lambda_8$ directly from $cCPT_{sys}$, we proceed as follows: $cCPT_{sys}$ consists of $m_{sys}=3$ rows. Looking at row $j=1$, we see that it is a run; we are starting in row 1, so $S^1_{sys}\in$odd; the length of the run is 31, so $L^1_{r_{sys}}\in$odd. Therefore, to construct the first row of λ_8, we use the rules in the first row of Table 2: $r^1_8=r^1_{sys}=0$, $L^1_{r_8}=\left(L^1_{sys}-1\right)/2=(31-1)/2=15$, and we have the remainder $R^1=r^1_{sys}\times\Pr(C_8=0)=0$.

Moving to row $j=2$, we have a phrase that starts in row 32 with a length of 4. Therefore, we use the last row of Table 2 to construct the second row of $c\lambda_8$: $p^2_8=p^2_{sys}=1$, $n^2_{p_8}=n^2_{p_{sys}}=56$, and we have a remainder $R^2=v^2_{2_{sys}}\times\Pr(C_8=0)=0$. Because we are now dealing with phrases, we also need to update our dictionary d_8 with this new phrase starting in row $j=2$ of $c\lambda_8$. Given the even starting row number and even phrase length, we again use the last row, now in Table 3, to update d_8: $v^2_{1_8}=R^1+\left[v^2_{1_{sys}}\times Pr\,(C_8=1)\right]=0+\left[1\times 0.99\right]=0.99$, $v^2_{2_8}=v^2_{2_{sys}}=0$ and $L^2_{p_8}=\left[\left(L^2_{sys}-2\right)/2\right]+1=2$.

We see that run values r and phrase values v_1 and v_2 can be different from 0 or 1, even in the case of a binary system. However, the intermediate factors λ_i can still be compressed because the calculations for each row only involve the failure probability of one component so that the number of values that the runs and phrases take on is finite.

Finally, moving to row $j=3$, we have a run that starts in row 256 with a length of 1. Therefore, we use the third row of Table 2 to construct the last row of $c\lambda_8$: $r^3_8=r^3_{sys}\times\Pr(C_8=1)+R^2=(1\times 0.99)+0=0.99$, and $L^3_{r_8}=1$. In this case, because the run is of length 1, we only have a value for the first run, and there is no second run that needs to be taken into account for the special case of $S^3_{sys}\in$even. The end result is the constructed $c\lambda_8$ and d_8, as shown in Tables 9 and 10, respectively.

We continue the elimination process until we have eliminated components $C_2,\ldots,C_8$ and arrive at the query node of interest, C_1. Our original inference question was to obtain the posterior probability distribution of C_1 given an observation of system failure. From the VE process, we obtain the joint probability $p(C_1,sys)$. To arrive at the posterior probability, we also need $p(sys)$. We obtain $p(sys=0)=0.0201$ by further eliminating C_1 and then compute the final result: $p(C_1|sys=0)=p(C_1,sys=0)/p(sys=0)=[0.2093\ 0.7907]$ for C_1 being in the failure or survival state, respectively. Given the observation that the system has failed, the probability of failure of C_1 has been updated from a prior failure probability of 0.2 to a posterior failure probability of 0.2093.

Comparing the uncompressed intermediate factor λ_8 given in Table 8 with the compressed $c\lambda_8$ and accompanying dictionary d_8 given in Tables 10 and 11, respectively, we see that the memory storage requirement has been reduced from $2^7=128$ elements to a total of 13 elements, 9 for $c\lambda_8$ and 4 for d_8. Similar reductions in memory storage are achieved at each step of the elimination process for inference. We can again verify that the number of rows represented in $c\lambda_8$ corresponds with the number of rows of λ_8: In $c\lambda_8$, we have a run of length 15, 56 instances of a phrase of length 2, and end with a run of length 1. This equals $15+56\times 2+1=128$, which equals the number of rows in λ_8. The compression is lossless and we again emphasize that we do not need to decompress $cCPT_{sys}$ or any of the compressed intermediate factors $c\lambda_i$ to conduct the inference.

5.2.1. Application of inference algorithm for dependent components

For an illustration of the treatment of dependent component states, we add a parent node H representing a common hazard on components $\{C_1, C_2, C_3\}$ of the system, see Fig. 4.

The probabilities of failure of the parallel components are now dependent on the occurrence of the hazard, where $H=0$ and $H=1$ indicate non-occurrence and occurrence of the hazard, respectively. The BN is initialized such that without the hazard occurring, the failure probability of each component is as before, 0.2. However, if the hazard occurs, the failure probability of each

Table 9
Compressed intermediate factor $c\lambda_8$ constructed after elimination of C_8 in the example system.

run or phrase	r or p	L_r or n_p
Run	0	15
Phrase	1	56
Run	0.99	1

Table 10
Dictionary d_8 constructed after elimination of C_8 in the example system.

p	v_1	v_2	L_p
1	0.99	0	2

Table 11
Compressed intermediate factor of parent $c\lambda_H$ for example system.

run or phrase	r or p	L_r or n_p
run	0.0713	1
run	0.1809	2
phrase	1	1

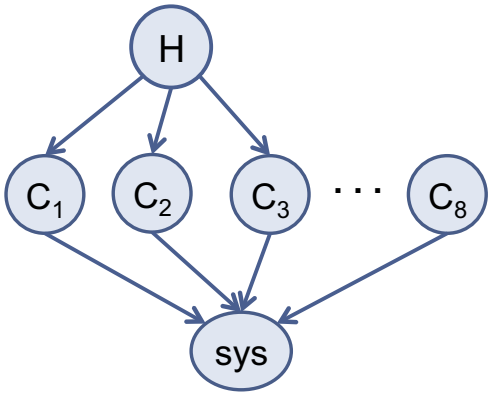

Fig. 4. BN of example system with dependent components.

component increases to 0.4. Assume the probability of the hazard occurring is $\Pr(H=1)=0.15$. Now, suppose we have evidence that component 1 has failed. We use the algorithm for dependent components presented in Section 4.3 to perform inference on the state of the system.

The first step in the algorithm is to calculate the marginalized prior probabilities of failure of the components with parent nodes. For example, for C_1, we sum over the states of H to obtain $p(C_1=0)=\sum_P p(C_1|H)p(H)=0.23$. Next, given the evidence, we update the probability distribution of H using Bayes' rule to obtain the updated hazard probability $p(HC_1=0)=p(C_1=0H)p(H)/p(C_1=0)=0.2609$.

Next, we eliminate the parent node and create the compressed intermediate factor. In our example, in the first row of λ_H, $\{s_1, s_2, s_3\}=\{0,0,0\}$. Thus, the value in the first row of λ_H is calculated as $\sum_H p(C_1=0|H)p(C_2=0|H)p(C_3=0|H)p(H)=0.0713$. We continue through the remaining rows and utilize the compression algorithm to result in the compressed intermediate factor $c\lambda_H$ and the accompanying dictionary d_H as shown in Tables 11,12, respectively.

The above algorithm is particularly efficient in creating the compressed intermediate factors according to Equation (5) when many components have similar dependencies on the parents, i.e., have identical conditional probabilities. Because the values in the intermediate factors of the parents are calculated from the state probabilities of the children components, under this condition the intermediate factors assume a limited number of values. For example, if there are ten components numbered 1, …, 10 that are children of a common parent, if the conditional probabilities are identical, then values in λ_H for the case where C_1 and C_2 are in the failed state are the same as those when C_1 and C_3 are failed, C_1 and C_4 are failed, C_2 and C_3 are failed, etc. This condition leads to an efficient construction of $c\lambda_H$. Additionally, if the components have many different parents, the algorithm is also tractable in that the number of terms in the expression for (and thus the size of) the intermediate factors for the parents is small. The first scenario describes a situation where many components share similar dependencies on, e.g., a common hazard. The second case describes the more detailed modeling of specific components, e.g., small groups of components having specialized dependencies on particular parent nodes.

After compressing the intermediate factor of the parent nodes, we begin the variable elimination process for inference. We begin with the compressed system CPT as shown in Tables 7 and 8. First, components $C_8,\ldots,C_4$ are eliminated using the standard VE algorithm. Since $C_3\in\hat{C}$, its parent intermediate factor λ_H must be accounted for. Thus, C_3 is eliminated by summing over the product of the intermediate factor from the previous step, λ_4, and the intermediate factor of the parent λ_H, i.e., $\lambda_3=\sum_{C_3}\lambda_H\lambda_4$. This results in the compressed intermediate factor and dictionary shown in Tables 13 and 14, respectively.

Table 12
Dictionary $\boldsymbol{d_H}$ for intermediate factor of parent for example system.

p	v_1	v_2	L_p
1	0.5669	0	5

Table 13
Compressed intermediate factor $c\lambda_3$ after elimination of $\boldsymbol{C}_3$ in the example system with dependent component states.

run or phrase	r or p	L_r or n_p
run	0.2451	1
phrase	1	1

Table 14
Dictionary $\boldsymbol{d}_3$ after elimination of $\boldsymbol{C}_3$ in the example system with dependent component states.

p	v_1	v_2	L_p
1	0.7329	0	3

Table 15
Compressed intermediate factor $c\lambda_2$ after elimination of $\boldsymbol{C}_2$ in the example system with dependent component states.

run or phrase	r or p	L_r or n_p
phrase	1	1

Table 16
Dictionary $\boldsymbol{d}_2$ after elimination of $\boldsymbol{C}_2$ in the example system with dependent component states.

p	v_1	v_2	L_p
1	0.9780	0	2

In the next step, we eliminate C_2, which is also an element of $\hat{C}$. However, its parent intermediate factor λ_P has already been accounted for. Thus, we construct intermediate factor λ_2 using the case rules in Tables 4 and 5. Working from row 1 of $c\lambda_3$, we have a run with $S_3^1\in$odd and $L_{r_3}^1\in$odd. Therefore, to construct the first row of the compressed factor $c\lambda_2^1$, we use the rules in the first row of Table 4: $r_2^1=2\times r_3^1=2\times 0.2451=0.4902$, $L_{r_2}^1=\left(L_3^1-1\right)/2=(1-1)/2=0$, and we have a remainder $R^1=r_3^1=0.2451$. Note that as the length of this run is 0, the value of this run is discarded. The remainder, however, remains and is brought forward to the next row.

In row 2, we have a phrase with $S_3^2\in$even and $L_{r_3}^2\in$odd. Thus, to construct row 2 of the compressed intermediate factor $c\lambda_2$, we use the rules in the seventh row of Table 4: $p_2^2=p_3^2=1$, $n_{p_2}^2=n_{p_3}^2=1$, and no remainder; and the third row of Table 5 to construct the dictionary: $v_{1_2}^2=v_{1_3}^2+R^1=0.7329+0.2451=0.9780$, $v_{2_2}^2=2\times v_{2_3}^2=2\times 0=0$, and $L_{p_2}^2=\left[\left(L_3^2-1\right)/2\right]+1=2$. The end results for $c\lambda_2$ and d_2 are shown in Tables 15 and 16, respectively.

We continue the elimination process to arrive at a posterior system failure probability of $p(sys=0)=0.0220$ given the evidence that component 1 has failed.

5.3. Expanded example systems

To demonstrate the performance of the proposed algorithms in modeling systems of increasing size, we modify the example system

in Fig. 3 by adding components either to the series or the parallel subsystem so that the total number of components in the system is n. The resulting test systems are shown in Figs. 5 and 6, respectively. The MCSs of these systems are $\{(C_1, C_2, C_3, C_6),\ldots,(C_1, C_2, C_3, C_n), (C_4), (C_5)\}$, and $\{(C_1,\ldots,C_{n-5},C_{n-4}), (C_1,\ldots,C_{n-5}, C_{n-3}), (C_1,\ldots,C_{n-5}, C_{n-2}), (C_{n-1}), (C_n)\}$, respectively. The BNs for the test systems are initialized with prior probabilities of failure of 0.2 for components in the parallel subsystem and 0.01 for components in the series subsystems. Component states are assumed to be statistically independent. We are interested in updating the probabilities of failure of the system and component 1, given evidence as described below.

The resulting analyses of these two expanded systems demonstrates how the proposed algorithms perform compared to existing algorithms for systems of increasing size. We note that the above systems can be more efficiently represented as a system of three super-components, each representing a series or parallel subsystem, as described in [20] and further explored in [9,25,3] for system reliability problems. However, here we disregard this advantage in order to investigate the system size effect.

5.3.1. Performance: inference

Fig. 7 shows results for updated probabilities of system failure, given the evidence that component 1 has failed, i.e., $\Pr(sys = 0)C_1=0)$. Fig. 8 shows results for the updated probabilities of failure of component 1, given that the system has failed, i.e., $\Pr(C_1=0)sys = 0)$. The updated probabilities are plotted against the total number of components in the system, n.

In these figures, "increase series" indicates the results for the system in Fig. 5 and "increase parallel" indicates the results for the system in Fig. 6.

In Fig. 7, we see that the updated probability that the system fails increases as the number of components in the series subsystem increases, as there are more MCSs that can fail and lead to system failure. In contrast, as the number of components in the parallel subsystem increases, the updated failure probability of the system initially decreases and eventually becomes essentially constant for systems of 11 or more components. This is because the probability of failure of MCSs involving the increasing number of parallel components diminishes and the system failure probability becomes dominated by the failure probabilities of the two single-component MCSs $\{C_{n-1}\}$ and $\{C_n\}$.

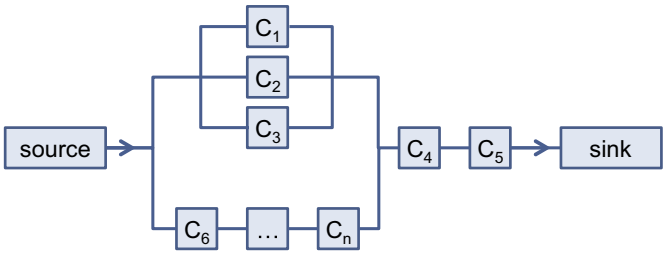

Fig. 5. Example test system: expanded with increased number of components in series subsystem.

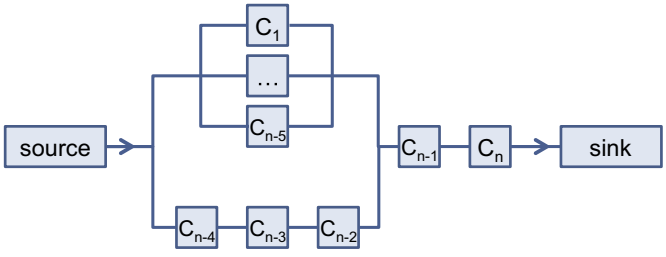

Fig. 6. Example test system: expanded with increased number of components in parallel subsystem.

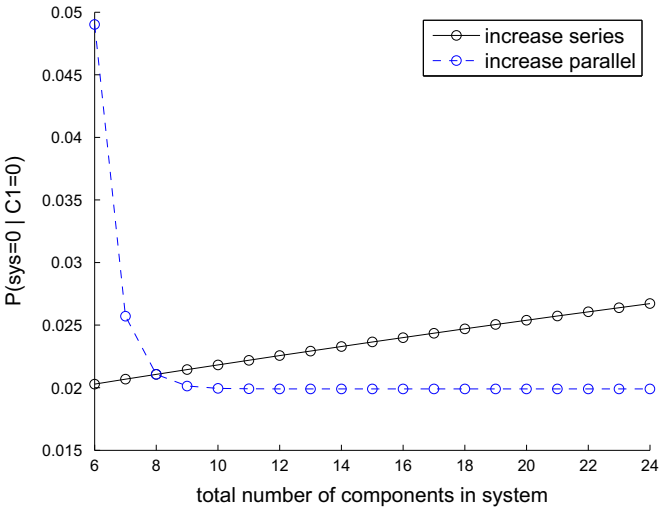

Fig. 7. Updated probabilities of system failure given component failure as a function of increasing system size.

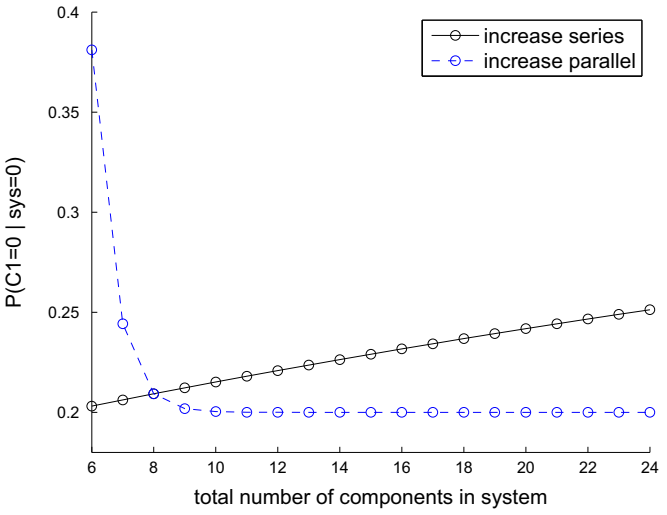

Fig. 8. Updated probabilities of component failure given system failure as a function of increasing system size.

In Fig. 8, we see that as the number of components in the series subsystem increases, the conditional probability that C_1 has failed given that the system has failed increases. With an increased number of components in the series subsystem, there are an increased number of MCSs that involve component 1, i.e., $\{C_1, C_2, C_3, C_i\}$, $i = 6, \ldots, n$. Since failure of any of these MCSs leads to system failure, an increase in the system size gives a higher probability that failure of component C_1 was "necessary" for the system to fail. In contrast, as the number of components in the parallel subsystem increases, the updated probability of failure of component C_1given system failure again converges for a system of 11 components or more. In this case, the evidence on the system state is not informative for the component, and the updated probability of failure converges to the prior probability of failure of component C_1. This is because the parallel system being highly reliable, it is unlikely to have contributed to the system failure. Figs. 7 and 8 show that the proposed algorithms successfully perform inference with evidence on both component and system states.

5.3.2. Performance: memory storage

We examine the performance of the new algorithms compared to an existing method on two measures: memory storage and computation time. For the existing method, we use the JT algorithm as implemented in the Bayes Net Toolbox by [18]. The algorithms are run in MATLAB v7.10 on a 32 Gb RAM computer with 2.2 GHz Intel Core i7 processor.

Fig. 9 shows the maximum number of values that must be stored in memory during the running of the algorithms, which is used as a proxy to assess the memory storage requirements of each algorithm. For a given n, the two example systems produce identical results. The values for the "New" algorithm are the maximum number of elements stored in $cCPT$ or $c\lambda_i$ and their associated dictionaries. The values for the "Existing" algorithm indicate the maximum number of elements stored using the JT and the naïve BN formulation shown in Fig. 1. The symbol "X" marks the maximum size of the system after which the existing algorithm can no longer be used because the memory demand exceeds the available memory storage capacity.

Fig. 9 shows that the proposed algorithm achieves significant gains in memory storage demand compared to the existing algorithm. For the existing JT algorithm, the memory storage demand, as measured by the number of values that must be stored, increases exponentially with the number of components in the system. In fact, on our computer with 32 Gb RAM memory, the algorithm runs out of memory for a system comprised of 24 components. For the proposed algorithm, the memory storage demand not only does not increase exponentially, but remains constant, even as the number of components in the system increases. The total number of values stored is 15, compared to 2^{n+1} for the size of the full system CPT. For the example system, this number remains constant at 15 total elements stored for $n \geq 24$.

5.3.3. Performance: computational efficiency

Fig. 10 shows the computation times required to run the two algorithms as a function of increasing system size. Computation times are broken into the various functions for each algorithm. The bars labeled "New - compression" indicate the time required to compress the system CPT using the proposed compression algorithm. The next two bars indicate the times required to perform inference on the system given $\{C_1{=}0\}$ and on component C_1 given $\{sys = 0\}$ using the proposed algorithm. The bars labeled "Existing - initialization" indicate the times required to initialize the BN using the JT algorithm. The next two bars indicate the times required to perform inference on the system given $\{C_1{=}0\}$ and inference on component C_1 given $\{sys = 0\}$ using JT. The computation times are recorded for systems of increasing size, as indicated by the total number of components in the system. The results in Fig. 10 are for the system in Fig. 5.

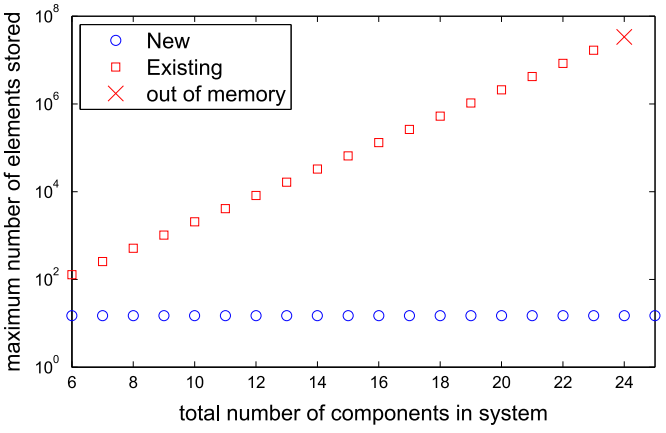

Fig. 9. Memory storage requirements for the proposed new algorithm compared to existing method as a function of system size.

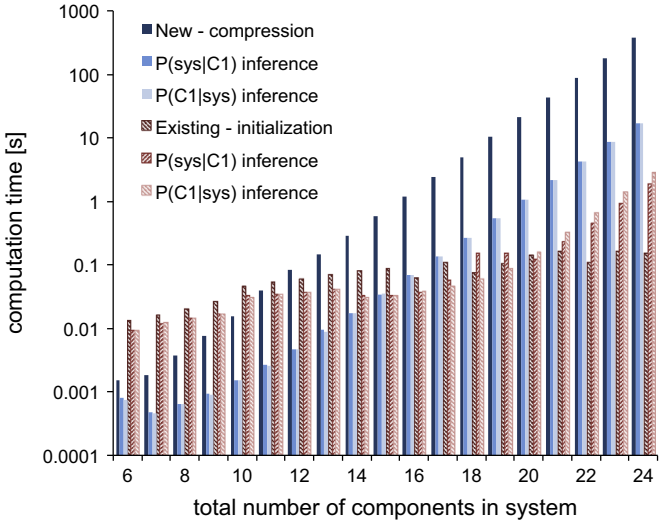

Fig. 10. Computation times for proposed and JT algorithms as a function of system size for the system with increasing components in series.

Examining Figs. 9 and 10 together, we see the classic storage space-computation time trade-off, as described in [8] and shown in [31]. Fig. 9 shows significant gains in memory storage demand achieved by the proposed algorithm compared to the JT algorithm. Fig. 10 shows that the new algorithm requires longer computation times than the JT algorithm. We note, however, that as the system become large, i.e., $n > 20$, the time to perform inference for both algorithms increases exponentially with the system size. For the JT algorithm, this is due to the increasing size of the cliques. For the new algorithm, it is due to the computations needed for compressing larger CPTs and intermediate factors λ_i during the variable elimination process. As these increases are both exponential in nature, it is estimated that the difference in computational efficiency between the two approaches will remain similar as the number of components in the system increases, as is seen in the trend of the computation time results for $22 \leq n \leq 24$ in Fig. 10.

It is important to note that the natures of the memory and time constraints are fundamentally different. Memory storage is a hard constraint. If the maximum size of the required memory exceeds the storage capacity of a program or machine, no analysis can be performed. While it is true that memory can be distributed, e.g., in cloud storage, there still exists a hard limit on the maximum. In contrast, computation time is more flexible. Indeed, various recourses are available to address the computational time, such as parallel computing. In addition, there are several heuristics that can be used to significantly enhance computational efficiency of the proposed algorithms, including component ordering and use of super-components. A future article will address these enhancements of the proposed algorithms.

6. Conclusions

Novel algorithms are presented that address the major system size limitation issue in the use of BNs for modeling large systems. These include a compression algorithm that significantly reduces the required memory size to store the conditional probability table (CPT) associated with the system node and the intermediate factors required in the variable elimination method for inference in the BN, and an algorithm that performs inference with compressed matrices without decompressing or recompressing them.

The latter enables the memory storage savings from the compression algorithm to be preserved throughout the inference process. Compared to a widely used existing algorithm, the new algorithms are shown to achieve orders of magnitude savings in memory storage requirement; however, this comes at the expense of increased computation time. The proposed algorithms enable the use of BNs to model and assess the reliability of large infrastructure systems, which cannot be handled with existing algorithms.

Acknowledgments

The first author acknowledges support from the National Science Foundation Graduate Research Fellowship from 2011 to 2014. Additional support was provided by the National Science Foundation Grant no. CMMI-1130061, which is also gratefully acknowledged.

References

[1] Akers SB. Binary decision diagrams. IEEE Trans Comput 1978;C-27(6):509–16.
[2] Benaddy M, Wakrim M. Cutset enumerating and network reliability computing by a new recursive algorithm and inclusion exclusion principle. Int J Comput Appl 2012;45(16):22–5.
[3] Bensi M, Der Kiureghian A, Straub D. Efficient Bayesian network modeling of systems. Reliab Eng Syst Saf 2013;112:200–13.
[4] Birolini A. Reliability engineering: theory and practice. 4th ed.,. Berlin: Springer,; 2004.
[5] Bobbio A, Portinale L, Minichino M, Ciancamerla E. Improving the analysis of dependable systems by mapping fault trees into Bayesian networks. Reliab Eng Syst Saf 2001;71(3):249–60.
[6] Boudali H, Dugan JB. A discrete-time Bayesian network reliability modeling and analysis framework. Reliab Eng Syst Saf 2005;87:337–49.
[7] Bryant RE. Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput 1986;C-35(8):677–91.
[8] Dechter R. Bucket elimination: a unifying framework for reasoning. Artif Intell 1999;113:41–85.
[9] Der Kiureghian A, Song J. Multi-scale reliability analysis and updating of complex systems by use of linear programming. Reliab Eng Syst Saf 2008;93:288–97.
[10] Fard NS. Determination of minimal cut sets of a complex fault tree. Comput Ind Eng 1997;33(1–2):59–62.
[11] Gilks WR, Richardson S, Spiegelhalter DJ. Markov chain Monte Carlo in practice. 1st ed.,. . London: Chapman & Hall,; 1996. p. 486.
[12] Hauck EL. Data compression using run length encoding and statistical encoding. U.S. Patent 4 626 829 A; December 2, 1986.
[13] Jensen FV, Nielsen TD. Bayesian networks and decision graphs. 2nd ed.,. New York: Springer,; 2007.
[14] Kim MC. Reliability block diagram with general gates and its application to system reliability analysis. Ann Nucl Energy 2011;38:2456–61.
[15] Li J, Qian Y, Liu W. "Minimal cut-based recursive decomposition algorithm for seismic reliability evaluation of lifeline networks. Earthq Eng Eng Vib 2007;6(1):21–8.
[16] Mahadevan S, Zhang R, Smith N. Bayesian networks for system reliability reassessment. Struct Saf 2001;23:231–51.
[17] Modarres M, Kaminskiy M, Krivtsov V. Reliability engineering and risk analysis: a practical guide. 2nd ed.,. Boca Raton, Florida: CRC Press,; 2010.
[18] Murphy KP. The Bayes Net Toolbox for Matlab. Computing Science and Statistics: Proceedings of the Interface. Vol. 33, October 2001.
[19] Nielsen TD, Wuillemin PH, Jensen FV. Using ROBDDs for inference in Bayesian networks with troubleshooting as an example. In: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford University, Stanford, CA, June 30–July 3; 2000. pp. 426–35.
[20] Pages A, Gondran M. System reliability: evaluation and prediction in engineering.New York: Springer-Verlag,; 1986.
[21] Pearl J. Probabilistic reasoning in intelligent systems.Networks of Plausible Inference. San Francisco, CA: Morgan Kaufmann,; 1988.
[22] Rausand M, Hoyland A. System reliability theory. Models, statistical methods, and applications. 2nd ed.,. Hoboken, New Jersey: John Wiley & Sons,; 2004.
[23] Salmeron A, Cano A, Moral S. Importance sampling in Bayesian networks using proability trees. Comput Stat Data Anal 2000;34:387–413.
[24] Shin YY, Koh JS. An algorithm for generating minimal cutsets of undirected graphs. Korean J Comput Appl Math 1998;5(3):681–93.
[25] Song J, Ok SY. Multi-scale system reliability analysis of lifeline networks under earthquake hazards. Earthq Eng Struct Dyn 2010;39:259–79.
[26] Spiegelhalter DJ, Dawid AP, Lauritzen SL, Cowell RG. Bayesian Analysis in Expert Systems. Stat Sci 1993;8(3):219–47.
[27] Straub D. Stochastic modeling of deterioration processes through dynamic Bayesian networks. J Eng Mech 2009;135(10):1089–99.
[28] Suh H, Chang CK. Algorithms for the minimal cutsets enumeration of networks by graph search and branch addition. Proceedings of the 25th Annual IEEE Conference on Local Computer Networks, Tampa, FL7; November 8–10, 2000. pp. 100–10.
[29] Tan Z. Minimal cut sets of s-t networks with k-out-of-n nodes. Reliab Eng Syst Saf 2003;82(1):49–54.
[30] Tien I. Bayesian network methods for modeling and reliability assessment of infrastructure systems [Doctoral Thesis].Berkeley: University of California; 2014.
[31] Tien I, Der_Kiureghian A. Compression Algorithm for Bayesian Network Modeling of Binary Systems. In: Deodatis G, Ellingwood B, Frangopol D, editors. Safety, Reliability, Risk and Life-Cycle Performance of Structures and Infrastructures. New York: CRC Press; 2013. p. 3075–81.
[32] Torres-Toledano JG, Succar LE. Bayesian networks for reliability analysis of complex systems. Lect Notes Artif Intell 1998;1484:195–206.
[33] Vesely WE, Goldberg FF, Roberts NH, Haasl DF. Fault Tree Handbook (NUREG-0492). Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission; January 1981.
[34] Yeh W-C. A new algorithm for generating minimal cut sets in k-out-of-n networks. Reliab Eng Syst Saf 2006;91(1):36–43.
[35] Yuan C, Druzdzel MJ. Importance sampling algorithms for Bayesian networks: Principles and performance. Math Comput Model 2006;43(9–10):1189–207.
[36] Ziv J, Lempel A. A universal algorithm for sequential data compression. IEEE Trans Inf Theory 1977;23(3):337–43.