https://doi.org/10.1016/j.ress.2021.107533

Scalable k-out-of-n Models for Dependability Analysis with Bayesian Networks

Otto Bibartiu Frank Dürr Kurt Rothermel
University of Stuttgart
Institute for Parallel and Distributed Systems (IPVS)
Universitätsstrasse 38 Stuttgart, Germany
{first name}.{last name}@ipvs.uni-stuttgart.de

Beate Ottenwälder Andreas Grau
Robert Bosch GmbH
Bosch IoT Cloud
Stuttgart, Germany
Beate.Ottenwaelder@de.bosch.com
Andreas.Grau2@de.bosch.com

***Abstract*—Availability analysis is indispensable in evaluating the dependability of safety and business-critical systems, for which fault tree analysis (FTA) has proven very useful throughout research and industry. Fault trees (FT) can be analyzed by means of a rich set of mathematical models. One particular model are Bayesian networks (BNs) which have gained considerable popularity recently due to their powerful inference abilities. However, large-scale systems, as found in modern data centers for cloud computing, pose modeling challenges that require scalable availability models. An equivalent BN of a FT has no scalable representation for the k-out-of-n (k/n) voting gate because the conditional probability table that constitutes the k/n voting gate grows exponentially in n. Thus, the memory becomes the limiting factor.**

We propose a scalable k/n voting gate representation for BNs, based on the temporal noisy adder. The resulting model reduces the initial exponential to polynomial memory growth without a custom inference algorithm. Previous BN implementations of the k/n voting gate could only handle around 30 input events until memory limits make inference infeasible. However, our evaluation shows that our scalable model can handle more than 700 input events per gate, making it possible to evaluate large scale systems.

I. Introduction

Availability analysis plays an integral part in the development of dependable systems. Companies in the industry and service sectors are obligated to provide high availability for their safety and business-critical systems. This includes classical domains such as the automotive, aviation, telecommunication and financial institutions, and also new areas that emerged with the rise of cloud computing, such as smart grids [1], [2], connected cars [3], [4], e-health [5], smart factories [6], [7], to name a few. Due to the criticality of these areas, availability assessments are mandatory during the design and development of such systems. Failing to provide adequate availability might lead to financial or legal consequences or go as far as harming human life.

There are several methods to evaluate the availability of a system, among which Fault tree analysis (FTA) in conjunction with Bayesian networks (BNs) has gained large acceptance in the industry and research fields [8]–[12]. Fault trees (FTs) are graphs that describe how certain combinations of component faults, known as base events, can lead to an undesired system failure, known as the top event. Logic gates are used to create intermediate events, by forming a Boolean expression to describe what combinations of base events lead to a system failure [8]. There are three basic gate types: the AND gate, OR gate, and the k-out-of-n (k/n) voting gate. The AND/OR gates propagate a fault if all or one input event triggers a fault, respectively. The k/n voting gate propagates a fault when more than k-out-of-n components fail, i.e., at least k inputs are fault events. The voting gate is suitable to model groups of redundant components, where a group is considered available as long as no more than $n - k + 1$ components are available.

FTs provide a large variety of quantitative analysis methods to compute the system's fault probability, i.e., the probability to trigger the top event. A well-known method is to enumerate all minimal cut-sets (MCSs), i.e., sets containing a minimal number of base events that can cause the system to fail, and compute the probability that at least one MCS occurs. Finding such sets is in general NP-complete [13], which can make some FTs infeasible to evaluate by solely using this method. Other quantitative analysis methods are also possible, such as exploiting the FT structure to compute MCSs more efficiently [14], applying simulation for approximation [15], or using algebraic methods [16]. In this work, we focus on transforming FTs into BNs, where inference is applied to compute the system's fault probability [17], [18]. BNs are probabilistic graph models, where nodes represent discrete random variables, and directed edges model conditional dependencies between these nodes. BNs have proven useful in representing and evaluating equivalent FTs, due to their powerful modeling abilities and simple notation [19]–[22].

Nevertheless, representing k/n voting gates in BNs poses a scalability problem for a large number of input events regarding the quantitative representation of the BN. Large-scale systems, such as data stores in cloud computing or modern replicated database systems, have hundreds of storage and compute nodes, which are used as a failover in the case of node failures [23]. If we were to model these systems with the help of FTs, we might end up with a BN structure that becomes computational infeasible due to storage limitations. For example, previous methods to evaluate FTs with BNs use one random variable to represent a gate, c.f. Figure 1, where the variable's conditional probability table (CPT) constitutes the Boolean expression of the appropriate gate. We consider this representation the *naive* model. If the gate has many input

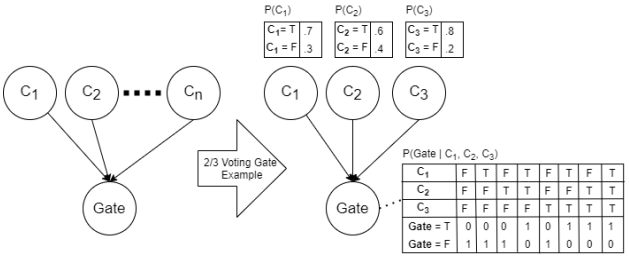


Fig. 1: BNs example of a 2/3 voting gate.

events, then the corresponding CPT of the random variable representing the k/n voting gate grows exponentially with the number of input events. For every possible combination of input states, the CPT stores an appropriate distribution for the random variable that constitutes the k/n voting.

For example, Figure 1 shows the BN representation of a 2/3 voting gate, where node K is a binary random variable that constitutes the voting gate and nodes C_1 to C_3 represent the base events. Since node K is conditionally dependent on three child nodes, denoted by the incoming edges, its CPT requires eight entries to describe the conditional probability distribution of K, c.f. the table in Figure 1. Here, F corresponds to the working state, while T corresponds to the failed state. For every state combination of K's parent nodes, we define a probability distribution for observing the states F and T. If we observe at least two input events in the state T, then the probability of the corresponding gate K triggering a fault event rises up to 1.

Consequently, if the input size is n then the CPT needs to store 2^n probability distributions for K. Assume we want to model the availability of a distributed computing system with 30 nodes where the majority of nodes need to be available to have a working system. Suppose, for simplicity reasons, we model the system as an FT with one k/n voting gate and the parameters $n = 30$ and $k = 15$. The CPT representing the 30/15 voting gate has 2^{30} entries. If each entry amounts to just one byte, then the total memory demand will be 16 *GB* to store the CPT, making memory space a limiting factor. This hinders us from assessing the availability of large-scale systems, such as modern cloud computing infrastructures. The exponential increase of the memory in the number of the input size is also known as the *exponential memory blow-up problem*. So far, BNs have scalable implementations to represent the AND/OR gates [24]–[26]; however, a scalable representation for the k/n voting gate is missing.

To use the k/n voting gate in settings as they occur in current large-scale systems, where we have large sets of redundant server clusters, we provide a *scalable* BN representation of the k/n voting gate. This work is based on our previous work [27], where we showed that the scalable k/n model can be built with the *temporal noisy adder* by Heckerman [24]. The BN model of the noisy adder is a random variable representing a counter and a probability distribution that defines the likelihood of observing a certain count. However, the BN model of the noisy adder still suffers from the exponential memory problem. Therefore, Heckerman proposed the noisy temporal adder, which is the scalable version of the general noisy adder. With the temporal noisy adder notion, we can reduce the space complexity from exponential to polynomial. We will discuss the temporal noisy adder in more detail when we derive the scalable k/n voting gate model later. In this article, we extend our previous work as follows. We enhance the modeling capabilities of the scalable k/n voting gate and provide an extensive evaluation. Experiments show that we can successfully represent and model k/n voting gates with more than 700 input events while still providing reasonable inference performance in the order of a few minutes.

The contributions we make are as follows

1) We show how to build an equivalent and scalable BN model of the k/n voting gate.
2) We discuss two approaches to implement a noisy k/n voting gate representation in BNs.
3) We show how to replace any naive BN implementation with a scalable k/n voting gate in existing BNs.
4) We also show that our scalable k/n voting gate model can be used with any standard (exact or approximate) inference algorithm.
5) We present numerical evidence, showing that the scalable k/n voting gate can be applied in availability analysis for large scenarios.

The remainder of this paper is structured as follows. First, we present related work in Section II, summarizing the research on BN dependability analysis and model scalability. Then, in Section III, we give a brief introduction to the notion of BNs. We illustrate the exponential memory growth problem of current approaches and state the aim of our research in Section IV. In Section V, we introduce the scalable k/n model, and in Section VI we provide an algorithmic approach on substituting a naive k/n voting gate instance in an already existing BN with the scalable model. Afterward, in Section VII, we provide evaluations to illustrate the performance and memory benefits of our solution. In Section VIII, we discuss possible modeling limitations and future work. Finally, we conclude the paper in Section IX.

II. Related Work

In this section, we discuss important research on FTA with a BN, focusing on the scalability aspects of the model.

Bobbio et al. [17] established the concepts for assessing static FTs with BNs, introducing the naive implementation of the k/n voting gate. Boudali and Dugan [18], [28] noticed the scalability problem of the k/n voting gate without providing a solution to it. However, they did suggest solutions for scalable AND/OR gates. The main idea is to divide the gates into smaller cascading AND/OR gates resulting in a lower number of input events per gate. This concept is also known as the parent-divorce method by Olesen et al. [29].

In the beginning of probabilistic graph modeling, knowledge engineers introduced the notion of causal independence to

provide efficient BN structures and to ease knowledge acquisition [24], [25]. Some important models are the noisy AND/ OR [25], [26], [30]–[32], noisy MAX [33], and the noisy adder [34], [35]. As a result, this research already provides scalable AND/OR gates representations for BNs. However, a scalable k/n model was still missing.

Iris and Kiureghian [36] have recently tackled the exponential memory growth of the k/n voting gate and proposed a lossless compression algorithm based on run-length encoding and Lempel Ziv compression to reduce the exponential size of the CPT. Nonetheless, their evaluations indicated that their approach does not scale for systems with hundreds of redundant components. Moreover, their approach requires a custom inference algorithm to handle the compression.

Our scalable k/n voting gate representation is an extension of the temporal noisy adder by Heckerman [24], which we use to solve the initial sub-problem of enumerating all possible state combinations of input events in a scalable manner. The temporal noisy adder already reduces the space complexity from exponential to polynomial, which helps us to implement the "scalable" part of the k/n voting gate.

III. Background

In this section, we describe necessary background information on BNs and fault modeling.

A (discrete) BN [26] is a directed acyclic graph $G = (X, E)$ with a joint probability $P(X)$. The nodes $X = \{X_1, X_2, \dots X_n\}$ are representations of random variables. We use the term *variable* or *node* interchangeably throughout the paper. Edges $E \subset X \times X$ represent conditional dependencies. The tuple $(X_i, X_j) \in E$ defines an edge where X_i is said to be a parent node of X_j, and X_j is a child node of X_i. We assign a conditional probability distribution to each variable, given its parents nodes $\text{pa}(X_i) = \{X_p : \forall (X_p, X_i) \in E\}$. Hence, $P(X_i = x_i | pa(X_i))$ describes the probability to observe a particular state $X_i = x_i$ given a specific state combinations of its parent variables. Nodes without parents are called root nodes, and have a prior probability distribution $P(X_i = x_i)$.

Since random variables are discrete and finite, it is common to express their conditional probability distributions in tabular form, known as conditional probability tables (CPTs). For example, Figure 1 shows a simple BN with four binary random variables with the states $\{T, F\}$. C_1 to C_3 are root nodes with different prior probabilities. We consider K to be a child node of C_1 to C_3, and C_1 to C_3 as the parent nodes of K. The CPT of K describes the conditional probability distribution given its parent nodes (right table). K's CPT contains a probability distribution for every state combination of K's parent nodes. Here, this BN represents a k/n voting gate (hence the name K for the node) of a FT's 2/3 voting gate. C_1 to C_3 represent the probability of observing a fault event of the input events, and K represents the gate's Boolean expression to trigger a fault when at least 2-out-of-3 input events are in a faulty state F. We set $\text{K} = F$ with probability one whenever the sum of its parent nodes that are in state F is greater or equal to $k = 2$.

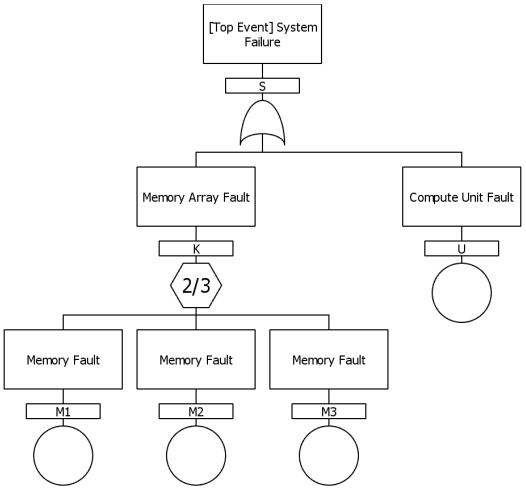


Fig. 2: FT of the computing system.

Finally, a BN defines a joint probability distribution as the product of all its conditional probability distributions:

$$P(X) = \prod_{x \in X} P(x|\text{pa}(x)) \tag{1}$$

In this work, we use the concepts proposed by Bobbio et al. [17] as building blocks in order to translate FT gates into their equivalent BN structure. For example, suppose we want to assess the availability of a simple compute system with the help of a BN. The system S consists of a compute unit U and an array of three redundant main memory units M_1, M_2, and M_3. The system fails when the compute unit or at most two memory units fail. The compute unit fails with a probability of 2%, and each memory unit with a probability of 10%, 15%, and 20%, respectively. Figure 2 shows the corresponding FT. The top event is the system's failure, which is the output of the OR gate S. The OR gate has the fault events of the compute unit and the memory array as its inputs. The memory array triggers a fault event when at least 2-out-of-3 memory units fail, as indicated by the 2/3 voting gate K.

Figure 3 shows the corresponding BN, where we encode the probabilities of a component fault as its eponymous binary random variables $\{U, M_1, M_2, M_3, K, S\}$, with the states $\{F, T\}$. Here, F corresponds to the working state, while T corresponds to the failure state. Node U, M_1, M_2, and M_3 are the so-called root nodes containing prior fault probabilities. Node K represents the 2/3 voting gate and has three parent nodes constituting eight different observable state combinations; from "all memory units work", up to "all units are faulty". For each state combination, we count the number of faults represented by the parent nodes and set the corresponding probability distribution of K with 100% certainty to T if we have more than two faults.

Node S defines the system's failure. Since S is conditionally dependent on two parent nodes, for each state combination of the parents we need to define a probability distribution. In our

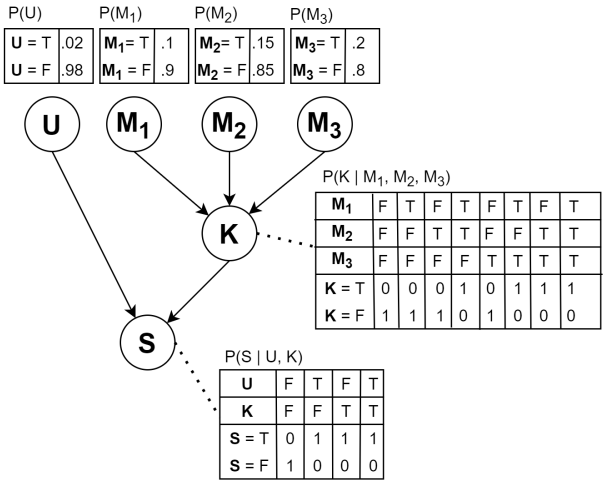


Fig. 3: BNs example of the computing system.

case, we model the system failure with the help of an OR gate representation [17].

Finally, we use inference to compute the system's fault probability, i.e., the top event. Generally, inference computes the posterior probability distribution $P(Q|K)$ of some query $Q \subset X$ of uncertain variables, where $K \subset X \backslash Q$ is a subset of given observations of the remaining variables. Exact and approximate inference algorithms are in general NP-hard [34], [37]. Yet, many BNs allow for fast inference times. For example, inference in BNs with a polytree topology has linear computation complexity [25]. Throughout this work, we will use abbreviations for readability reasons: the general term $X_i = x_i$ simply as lower capital x_i to distinguish between the random variable from its asserted/observed state. We compute the fault probability of our example by inferring the marginalization of the top event $P(S = T)$.

$$P(S=T) = \sum_{\forall u,m_1,m_2,m_3,k \in \{F,T\}} P(u, m_1, m_2, m_3, k, S{=}T)$$

where we use the definition from Equation 1 to compute the joint probability using the CPTs from the BN.

$$P(u, m_1, m_2, m_3, k, S{=}T) = P(S = T|u, k) \times P(u) \times P(k|m_1, m_2, m_3) \times P(m_1) \times P(m_2) \times P(m_3)$$

IV. Problem Statement

Consider the BN in Figure 4, also known as a *converging network* that constitutes the general structure to implement a k/n voting gate for arbitrary k and n. We refer to this BN structure as the *naive* implementation. For every state combination of the parent nodes, we set $K = T$ with probability one if at least k-out-of-n parent nodes are in state T. The CPT for K has the following definition for the k/n model, where we use the notation $c_1, \ldots, c_n$ to define one instance of state combinations of the parent nodes.

$$\forall c_1, \ldots, c_n \in \{F, T\}^n$$
$$P(K = T|c_1, \ldots, c_n) = \begin{cases} 1 & \sum_{i=1}^{n} \mathbf{1}_T(c_i) \geq k \\ 0 & \text{otherwise} \end{cases}$$
$$P(K = F|c_1, \ldots, c_n) = 1 - P(K = T|c_1, \ldots, c_n) \quad (2)$$

where $\mathbf{1}_T(x)$ is an indicator function such that

$$\mathbf{1}_T(x) := \begin{cases} 1 & \text{if } x = T, \\ 0 & \text{otherwise.} \end{cases}$$

For each combination of parent states we use Equation 2 to create an entry $P(K|C_1 = c_1, \ldots, C_n = c_n)$ for K's CPT. Hence, if all cause variables have a binary state, we need 2^n entries in K's CPT leading to an exponential growth of its probability table for an increasing number of parent nodes.

Complex systems with many redundant components, $n = 30$, already require a CPT with 16 GB of memory. For example, suppose we have a larger redundant storage service consisting of 60 units tolerating 30 faults. The CPT of the node K requires, in the end, 2^{60} entries because we explicitly enumerate and provide a conditional probability distribution for every fault combination of the gate's input events. If we need one byte to store one entry in the CPT of K, we need 2^{60} bytes to store the whole table. This results in 1 Zetabyte of memory for one random variable, which is impossible in the case of state-of-the-art compute infrastructures. Consequently, our objective is to reduce the exponential memory growth to a polynomial memory growth, in order to model larger redundant systems with BNs.

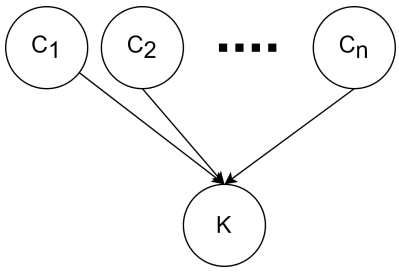


Fig. 4: Converging BN of a k/n voting gate model (Simple scenario).

V. Scalable k-out-of-n Model

Next, we show how to build the scalable k/n model for BNs by extending the temporal noisy adder to support the semantics of the k/n voting gate. First, we introduce the temporal noisy adder. Then, we continue with the description of the scalable k/n model. In the end, we discuss two possibilities to also implement a noisy k/n voting gate with BNs.

A. Temporal Noisy Adder Model

The *temporal noisy adder* BN model was introduced by Heckerman [24] as a scalable solution to the general noisy adder. A noisy adder is a random variable that represents a counter. Its probability distribution defines the likelihood of observing a certain count. In a general setting, the random variable of the noisy adder is conditionally dependent on the so-called *cause variable*. The noisy adder's goal is to infer the probability of observing a certain number of cause variables being in a defined distinguished state (including some uncertainty in the form of noise). The qualitative representation of the noisy adder would again resemble a converging BN, as shown in Figure 4. Here, the nodes C_1 to C_n would be the cause variables with binary state $\{F, T\}$, and instead of having binary states for node K, node K now has the domain $dom(X) = [0, n]$. Suppose the distinguished state of the cause variables is T, then $P(K = m|c_1, \ldots, c_2)$ would define the conditional probability of observing m cause variables being in the state T, given the observed states of the cause variables. Thus, $P(K = m)$ would then define the posterior probability of observing m cause variables in the state T. However, using a converging BN structure to represent a noisy adder leads again to the exponential memory blow-up problem.

To solve the exponential memory blow-up problem, Heckerman [24] proposed the temporal noisy adder. Figure 5 (gray boy) shows the general BN structure of the temporal noisy adder, which is a causal chain consisting of two node types. The upper row are cause variables and the bottom row with the nodes E_1 to E_n define the adder part. E_1 to E_n are also known as *contribution variables*. We use the contribution variable E_i to store the intermediate count of all cause variables that are in the distinguished state up to the i-th cause variable C_i. Thus, E_i can count at most i observations, hence its possible states are $\text{dom}(E_i) = \{0, \ldots, i\}$. The noisy adder considers noise by supplementing each contribution variable E_i with the probability q_i that determines the chance of actually incrementing the counter when $C_i = T$. We define the CPT of the contribution variables E_i as follows:

$$
\begin{aligned}
&\forall i \in [1, n] \forall m : m < i \\
&\quad P(E_i = m + 1 | E_{i-1} = m, C_i = T) = q_i \\
&\quad\quad P(E_i = m | E_{i-1} = m, C_i = T) = 1 - q_i \\
&\quad P(E_i = m + 1 | E_{i-1} = m, C_i = F) = 0 \\
&\quad\quad P(E_i = m | E_{i-1} = m, C_i = F) = 1
\end{aligned}
$$

Here, we increment E_i whenever C_i is assumed to be in state T. Assuming that the previous state of E_{i-1} is m, we set E_i to $m + 1$ with probability q_i. However, if $C_i = F$, then E_i propagates the current state of his parent node E_{i-1}. Note, that the CPT of E_0 is initialized to $P(E_0 = 0) = 1$. Finally, the last contribution variable E_n contains the count of N of the observed states of all causes.

B. Scalable k-out-of-n Model Construction

In the following section, we show how to build the scalable k/n voting gate for BNs. We start with the naive k/n voting gate representation, whose structure we gradually change into its scalable form.

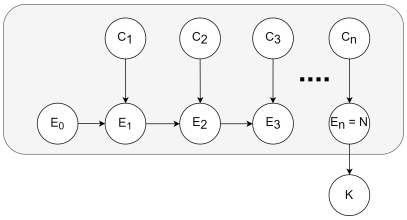


Fig. 5: Temporal representation of the noisy adder (gray box). The overall structure, including node K, represents the scalable BN structure for the k/n voting gate.

Consider the naive implementation of the k/n voting gate from Equation 2 and its representation from Figure 4. The conditional probability $P(K = T|c_1, \ldots, c_n)$ is 1 if at least k of the parent variables are in state T, and 0 otherwise. We compute the marginal probability distribution $P(K = T)$ with the help of the joint probability distribution (c.f. Equation 1).

$$
\begin{aligned}
&P(K = T) = \\
&\sum_{c_1, \ldots, c_n} P(K = T|c_1, \ldots, c_n) P(c_1) \ldots P(c_n)
\end{aligned}
$$

If the parent variables are not independent, we can change the products $P(c_1) \times \cdots \times P(c_n)$ to their joint probability distribution $P(c_1 \times \cdots \times c_n)$.

In Equation 2, the number of parent variables in state T influences the conditional probability of node *K*. Assuming causal independence, we define a distinguished state, denoted as F, as a neutral event that does not influence *K*. Causal independence assumes that each parent node has an individual influence on the conditional probability of their child node [38]. Thus, we rewrite the conditional probability distribution of $P(K = T|c_1, \ldots, c_2)$ as $P(K = T|N = n)$ where *K* is conditionally dependent on a new random variable *N* with the domain $[0, n]$. Here, *N* represents the number of parent variables that are in state T. We define the CPT of *N* as follows:

$$
P(N = m|c_1, \ldots, c_n) = \begin{cases} 1 & \sum_{i=1}^{n} \mathbf{1}_T(c_i) = m \\ 0 & \text{otherwise.} \end{cases}
$$

Thus, *N* implements an adder model since *N* counts the occurrences of a distinguished state of its parent variables.

Next, we include *N* into the conditional probability distribution of *K* to compute the marginal $P(K = T)$.

$$
\begin{aligned}
&P(K = T) = \\
&\sum_{c_1, \ldots, c_n} \sum_{m \in [0, n]} P(K = T|N = m) \times P(N = m|c_1, \ldots, c_n) \\
&\times P(C_1 = c_n) \ldots P(C_2 = c_2)
\end{aligned}
$$

The CPT of *K* has linear size now, due to its new conditional dependency to *N*.

$$
P(K = T|N = m) = \begin{cases} 1 & m \geq k \\ 0 & \text{otherwise.} \end{cases}
$$

However, the issue of the exponentially growing CPT shifts to N now, due to its conditional dependency on the variables C_1 to C_n. We rearrange the summations to isolate N and to apply the temporal transformation of the adder model.

$$\begin{aligned} &P(K=T) = \\ &\sum_{m\in[0,n]} P(K=T|N=m) \\ &\times \sum_{c_1,\ldots,c_n} \underbrace{P(N=m|c_1,\ldots,c_n)}_{B} \times P(c_1)\ldots P(c_n) \end{aligned} \tag{3}$$

We substitute the conditional probability distribution of N with the temporal-adder model in B, where B is defined as follows:

$$B \coloneqq P(E_0=0) \sum_{\substack{e_1,\ldots \\ e_n=m}} \prod_{i\in[1,n]} P(E_i=e_i|E_{i-1}=e_{i-1}, C_i=c_i).$$

The last term defines the adder model of *N*. Figure 5 depicts the resulting BN of Equation 3. *K* is now a child node of the contribution variable E_n in the causal chain. Since E_n represents the total count of the causes C_1 to C_n, *K* can enforce the Boolean expression of the k/n voting gate. *K* sets the conditional probability for $K=T$, when the count is at least k with $P(K=T|N\geq k)=1$, otherwise 0. By extending the temporal adder model with an additional random variable at the end of the chain, we can build a scalable k/n voting gate model. As a result, we can use any standard exact or approximate inference algorithm for inference because we did not deviate from the standard BN formalism.

C. Noisy k/n Voting Gate

Next, we discuss two options to implement a *noisy* k/n voting gate. The first option includes uncertainty in propagating a fault even in the case of more than k fault events at the inputs. The second option presupposes an uncertain observation of the fault event of certain inputs. We might observe inputs that trigger faults, although no fault has actually occurred (false positives) or vice-versa, where inputs trigger faults but we failed to observe them. Both options can be combined to a noisy k/n voting gate that models input and output uncertainty.

For the first option, we add noise at node K, where K triggers a fault with a constant probability q_k:

$$P(K=T|N=m) = \begin{cases} q_k & m \geq k \\ 0 & \text{otherwise.} \end{cases}$$

Even though we count more than k fault events, the k/n voting gate might still not trigger a fault for $m \geq k$ with probability $P(K=F|N=m) = 1-q_k$.

As a second option to implement a noisy k/n voting gate, we can use the original notation of the temporal *noisy*-adder. Each contribution variable has its probability of accepting the result of its cause variable to the counter. In this way, the knowledge engineer can assign a probability of observing the fault event correctly or not to each input event of the k/n voting gate.

D. Complexity

In the following, we analyze the computational and space complexity of the scalable k/n model.

The CPT of node K grows linearly in the number of cause variables. However, the overall space complexity of the temporal noisy adder grows polynomially with the number of cause variables. The total number of contribution variables equals the number of cause variables. The table size of a contribution variable is at most n^2. Hence, since we have a total of n CPTs, we also have a space complexity of $O(n^3)$, constituting a polynomial growth in the number of causes.

The computational complexity of node K is linear in the number of cause variables, and the computational complexity of the temporal noisy adder is $O(n^3)$ [24]. Hence, the overall computational complexity is $O(n^3)$.

VI. The Scalability Transformation of the Naive k:n Model

Next, we show how to translate a BN that contains a naive implementation of the k/n voting gate into an equivalent scalable model.

Assume that C_1 to C_n are parent nodes of a converging network structure with child node K. Then, the conversion of this BN into a scalable k/n model includes the following steps:

1) For each C_i a contribution variable E_i with the domain $[0,i]$ must be created.
2) Let E_i be a child of C_i and E_{i-1}. Let K be a child E_n. For each E_i define the temporal adder model:

$$\begin{aligned} P(E_i=k+1|E_{i-1}=k, C_i=T) &= 1 \\ P(E_i=k+1|E_{i-1}=k, C_i=F) &= 0 \\ P(E_i=k|E_{i-1}=k, C_i=T) &= 0 \\ P(E_i=k|E_{i-1}=k, C_i=F) &= 1 \end{aligned}$$

3) Define the conditional probability distribution of $P(K|E_n)$ with:

$$\begin{aligned} P(K=T|N\geq k) &= 1 \\ P(K=F|N\geq k) &= 0 \\ P(K=T|N< k) &= 0 \\ P(K=F|N< k) &= 1 \end{aligned}$$

This algorithm introduces new nodes into the BN. These nodes should not have any other parent nodes, except for their associated cause variables. If the existing BN model has multiple voting gates, denoted by the nodes K_1 to K_m, then we apply the transformation in the same manner to each gate. We denote the contribution variable of the i-th cause node that belongs to the j-th voting gate model as $E_{i,j}$, to certify that association to the voting gate is unambiguous.

VII. Numerical Evaluation

We evaluate the performance of our scalable k/n voting gate implementation w.r.t. inference time and memory consumption for three different scenarios. The first scenario uses a converging BN representing a set of redundant components.

The remaining scenarios use a mixed serial-parallel reliability block diagram example introduced by Tien et al. [36]. For each scenario, we model the system in two ways. First, we use the naive implementation for the BN availability model of the system, and afterwards we use the scalable k/n voting gate.

A. Experimental Setup

We evaluated all BN models on a 64-bit machine with an Intel(R) Xeon CPU with eight 3.4 GHz processor cores and 16 GB of RAM. We used three different BN libraries to perform inference.

- pgmpy (ver. 0.1.7) Python package, which offers an exact inference algorithm (the variable elimination method)
- bnlearn (ver. 4.5) R library for BN learning, which provides an approximate inference algorithm using Monte Carlo particle filters [39].
- gRain (ver. 1.2.3) R library for probabilistic graph inference [40], which provides an exact inference using the Lauritzen and Spiegelhalter propagation algorithm [41].

We apply these three inference libraries/algorithms in all our experiments to the naive and scalable BN models, in order to show that our scalable k/n model can be used by any standard inference algorithm.

B. Experimental Scenarios

The first scenario is a simple system with n redundant components where at least k component failures are tolerated. We model this system as one k/n voting gate using the converging BN shown in Figure 4. We call this scenario the *simple scenario*. All random variables have the states $\{F, T\}$. C_i represents the fault of the i-th component. For simplicity reasons and w.l.o.g. each component has a fault probability of 20%. However, the scalable model also supports independent input events with different fault probabilities. We use a fixed k, i.e., the majority set with $k = \lceil \frac{n}{2} \rceil$, in all three scenarios for convenience, because different k do not change the size of the conditional probability table of the k/n voting gate.

The next two scenarios are based on the example problems from Tien et al. [36]. Figure 6a (top) depicts a reliability block diagram of (RBD) a mixed serial-parallel system with its corresponding FT (bottom). The upper part of the RBD belongs to a parallel subsystem of redundant components, where each component has a fault probability of 20%. The bottom and right parts are serial subsystems where each component has a fault probability of 1%. When the parallel and the bottom serial subsystems, or C_4, or C_5 fail, the system fails. We call this scenario the *parallel scenario*.

Figure 7a shows the corresponding BN structure of the parallel scenario (using the naive implementation). In each experiment, we increase the parallel sub-system's component number (from C_1 to C_n). We define the probability of a system failure with the random variable `SYS`. The system fails if the parallel (C_1 to C_n) and the serial (from C_{n+1} to C_{n+3}) subsystems fail, which we enforce with the `AND` node. We model the serial subsystem by connecting all its nodes to the $\texttt{OR}_1$ node, which encompasses the Boolean expression of the OR gate.

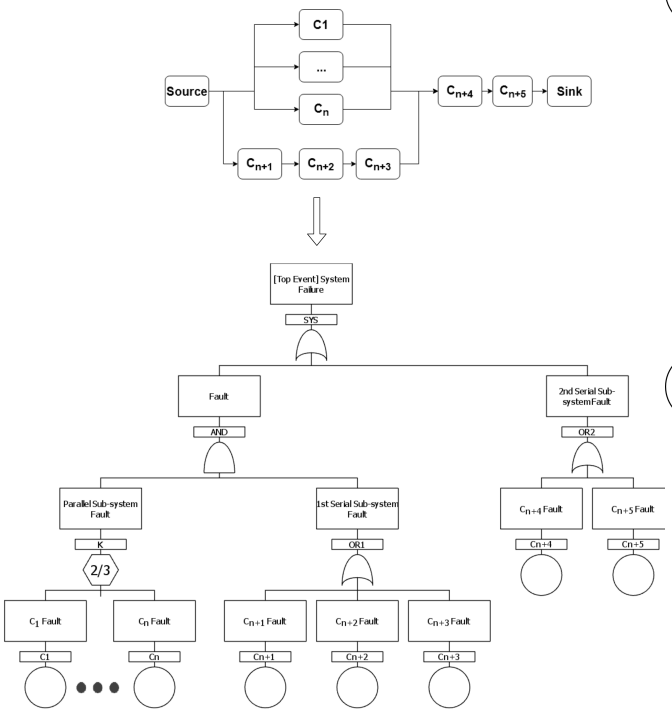
Next, Figure 6b illustrates the third scenario, using the same system as before, with an emphasis on the serial subsystem. The serial subsystem is modeled through the $\texttt{OR}_1$ model. As discussed in Section II, there are already existing scalable AND/OR implementations [24]. Therefore, we use the third scenario to contrast how the scalable k/n model compares to the AND/OR gate's scalable versions. We call this scenario the *serial scenario*. Components C_1 to C_3 are part of the parallel sub-systems using a 2-out-of-3 redundancy model. The components C_6 to C_{n+6} are part of the bottom serial subsystem. The fault probabilities are the same as in the second scenario. The goal is to analyze the performance and memory space of the $\texttt{OR}_1$ node for increasing the component number in the serial subsystem. Note that the scalable BN implementation of the $\texttt{OR}_1$ node uses binary contribution variables resulting in a linear memory growth. Figure 7b shows the corresponding BN of the third scenario. This BN is similar to the network of the parallel scenario, with a variable number of component nodes for the serial subsystem.

We evaluate all scenarios with two different BNs. The first network uses the naive approach to implement the k/n voting gate and the AND/OR models, while the second network uses their scalable representations. For example, Figure 8 shows the BN of the parallel scenario using the scalable model for the voting gate with $n = 4$ and $k = 2$. The nodes E_0 to E_4 are the contribution variables of the temporal noisy adder, whereas the CPT of $K_{2:4}$ implements the at least two-out-of-four semantic.

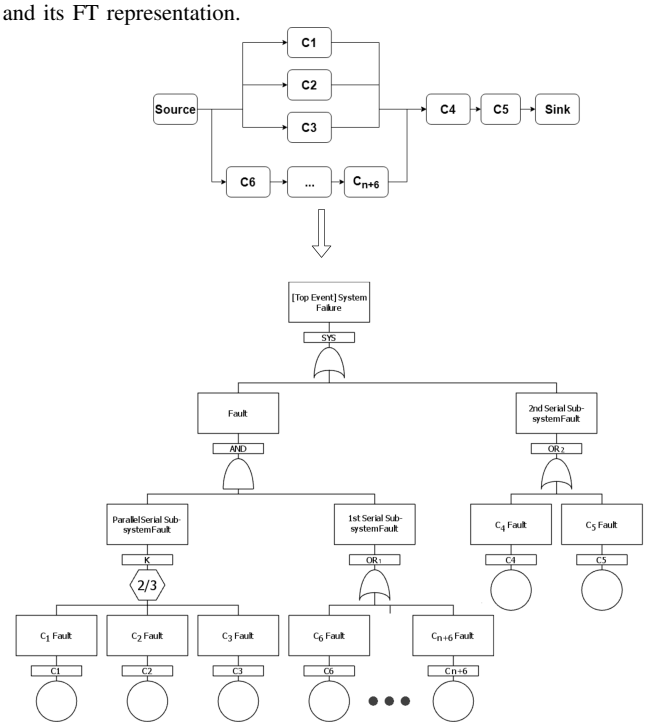
C. Availability

We compute the availability, for all systems, for all three scenarios for increasing numbers of components n, using the inference algorithms from pgmpy, bnlearn, and gRain to infer their availability. For the first scenario, we compute the system's availability with the query $P(K = F)$, and for the second and third scenarios, we compute the availability with the query $P(SYS = F)$, i.e., the top event is not triggered. We did not use evidence-based queries for simplicity reasons, although the scalable implementation also supports such queries.

Figure 9 shows the resulting system availability for each scenario where we compare the naive and scalable BN implementations to each other. All three inference algorithms compute the same expected availability demonstrating that the scalable model is equivalent to the naive model. We repeated each experimental run 20 times and computed their 95% confidence intervals since the result of the approximate inference algorithms might vary slightly by nature with every execution for both models. We can state with 95% confidence that there is no significant difference in the inference results

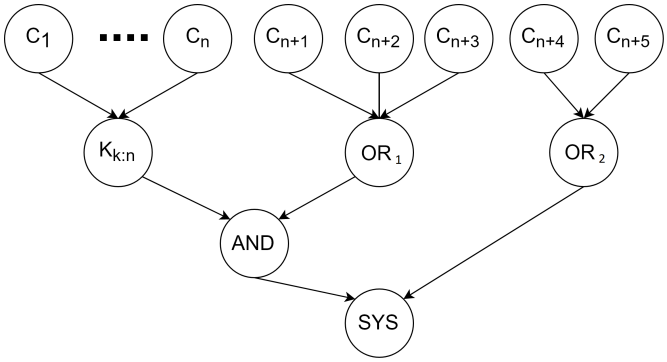


(a) Parallel scenario: complex system with variable parallel sub-system and its FT representation.

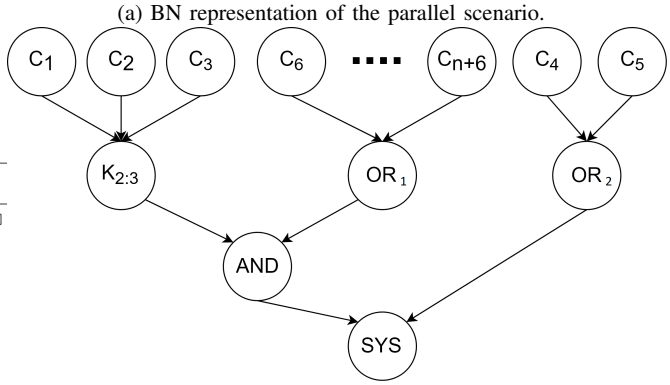


(b) Serial scenario: complex system with variable serial sub-system and its FT representation.

Fig. 6: Reliability block diagram and FT of a complex system.



(a) BN representation of the parallel scenario.



(b) BN representation of the serial scenario.

Fig. 7: BNs of the complex system.

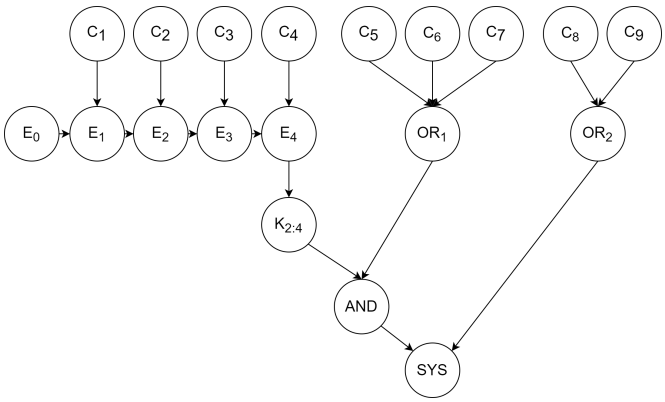
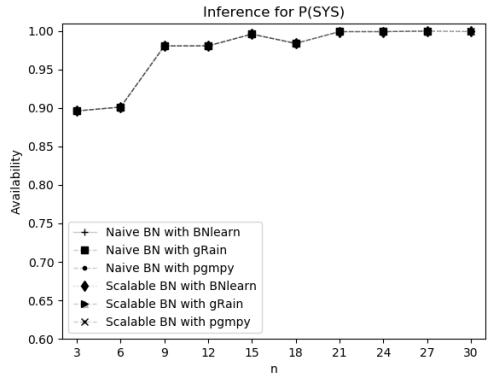


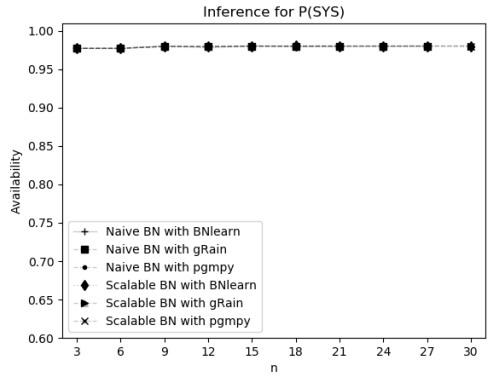
Fig. 8: BN representation of the parallel sub-system using the scalable model for the k/n voting gate with $n = 4$ and $k = 2$.

for all inference algorithms for the naive and scalable implementations in all scenarios.

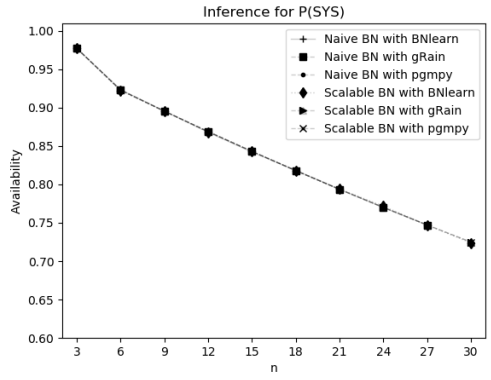
We only compared the availability for systems with n up to 27 components, because a further increase of components exceeds the main memory for the naive implementations in our experimental setup. It makes no sense to increase the server's main memory to evaluate larger networks for the naive k/n voting gate model. A system with 100 redundant components using the naive implementation is beyond the reach of any



(a) Simple scenario



(b) Parallel scenario



(c) Serial scenario

Fig. 9: Computing the availability for all three scenarios with an increasing number of components.

computational effort. However, the scalable k/n voting gate model is capable of representing even larger systems. For example, Figure 11a shows evaluations of the simple scenario with the scalable BN representation for $n > 700$, which we discuss next.

D. Time

Our scalable k/n voting gate introduces new nodes into the BN. These nodes increase the original network size influencing the inference time. Consequently, we compare the inference time of the naive model with the scalable model for all three scenarios. Our evaluation shows that the scalable model reduces the memory overhead and reduces inference time for these BN models.

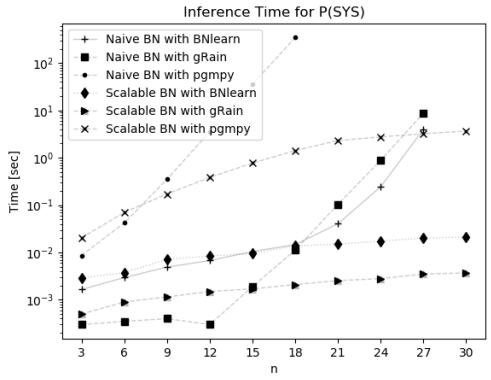
Figure 10 shows the mean inference time that is necessary to compute the fault probability. The y-axis has a log-scale, plotting the inference time against the system's size, i.e., the number of components n. We repeated each experiment 20 times, and we are able to report a 95% confidence level of a significant difference in the inference time for all three inference algorithms in all naive and scalable implementations for all scenarios for $n > 18$.

Larger systems profit from the scalable BN implementation, whereas smaller systems still profit from the naive representation of the gates. For $n \geq 24$, the scalable model is two orders of magnitude faster than the best inference time of the naive models. However, for $n \leq 18$, the scenarios show no significant difference between the scalable and the naive implementations in the majority of experiments. For $n \leq 12$, the naive model outperforms the scalable model when using gRain (exact inference). The naive model performs better for smaller n because it does not have the additional nodes of the scalable model that adds an initial overhead to the BN.

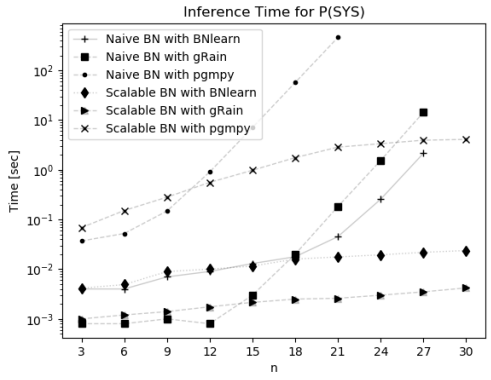
The scalable k/n model enables us to build and evaluate BNs with more than 700 nodes, as shown in Figure 11 (a continuation of Figure 10). The gRain library performs better than the approximate inference algorithm of bnlearn until $n = 300$. The serial scenario in Figure 11c uses an efficient OR gate implementation. Since the efficient OR only has linear space complexity, gRain performs best, followed by bnlearn. The logarithmic growth of the curve with increasing number of components shows that the average inference time grows polynomially in our scenarios. In contrast, the inference time for the naive model increases exponentially, as seen by the linear increase in the semi-log plot in Figure 10. Although inference is theoretically NP-hard, our solution scales up to two orders of magnitude with our improvements compared to the naive implementation.

E. Space

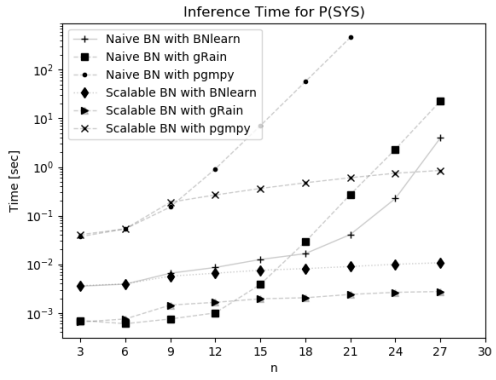
Next, we compare the memory utilization and the number of CPT entries in the case of the naive and scalable implementation, and show that the scalable model exhibits a polynomial memory growth. Figure 12 shows the memory size of serializing the BN to main memory. We measured the memory space of every BN instance in all three scenarios.



(a) Simple scenario

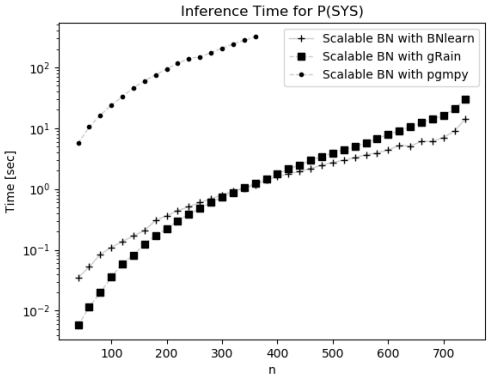


(b) Parallel scenario

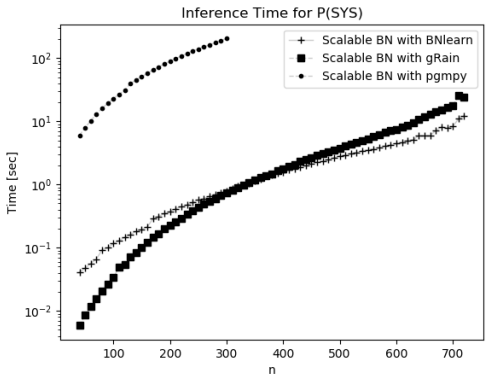


(c) Serial scenario

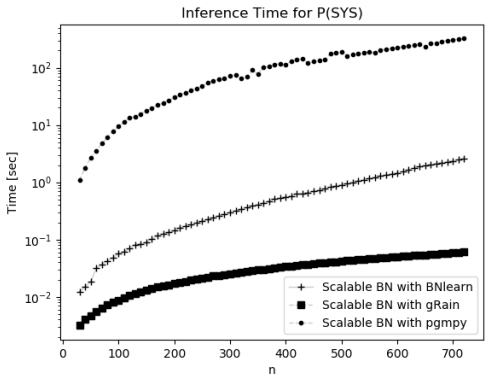
Fig. 10: Inference time to compute the availability of all three scenarios for increasing n up to 30



(a) Simple scenario



(b) Parallel scenario



(c) Serial scenario

Fig. 11: Inference time for larger systems with scalable models with n up to 700.

Solid lines stand for the CPT count and memory size of the naive implementations, whereas dashed lines illustrate the CPT count and memory sizes of the scalable implementations.

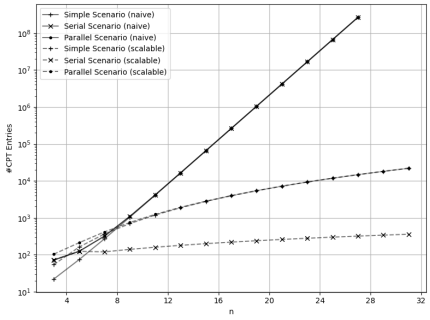
Figure 12a and Figure 12b show the total number of CPT entries and the memory consumption of the naive and scalable BNs. The CPT size of the naive implementations grows exponentially for the naive case, such that all scenarios have almost the same memory size and number of CPT entries for $n > 9$. Systems with $n \leq 9$ require less memory and has less number of CPTs with the naive implementation by comparison to the scalable approach. This is the case, because of the initial number of CPTs by the contribution variables. Again, the naive implementations reach their memory limit at $n = 27$. Overall, as one would expect, the number of CPT entries is direct proportional to the memory size of the BN since the CPT has the largest memory contribution when serializing the BN.

Figure 12c and Figure 12d show the continuation of the former figures, showing just scalable BN implementations of the scenarios. The logarithmic growth of all three scenarios in the semi-log plot indicates a polynomial growth of the memory size and number of CPT entries, respectively. Moreover, the serial scenario grows slower than the simple and parallel scenarios. This occurs because we use the efficient OR gate implementation for node OR_1, which uses only binary contribution variables; each contribution variable has just eight CPT entries. Therefore, the overall size has a linear space complexity of $O(n)$. This clearly shows the contrast to the k/n voting gate that uses $O(n^3)$ memory space compared to the OR gate implementation.

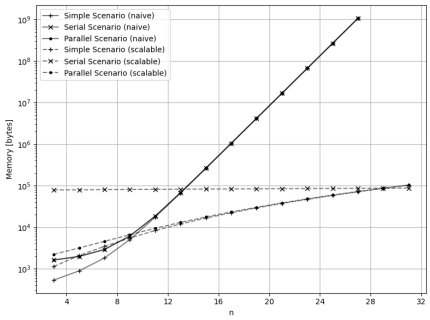
VIII. Discussion

Our evaluations show that the scalable k/n implementation increases our capabilities to evaluate and represent gates with hundreds of input events. However, if a voting gate has few input events, e.g., $n < 10$, we can still use the naive voting gate implementation. As our evaluations indicate, the naive voting gate performs better for small n because it does not have the initial overhead of the contributing variables of the scalable implantation, which hinders performance benefits in the beginning. Nevertheless, for larger n, one should use the scalable implementation instead.

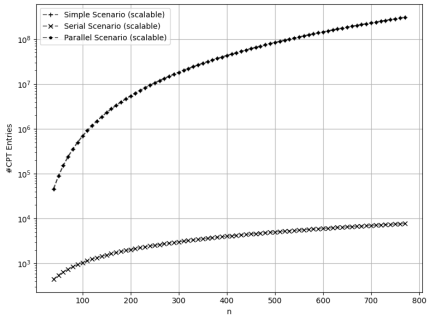
However, we cannot assume that, in general, inference time decreases as well. The execution time depends on the overall BN and the type of the inference algorithm. For instance, the scalable k/n model might be a sub-graph in a much larger BN. If we use an exact inference algorithm, such as the Lauritzen and Spiegelhalter (L&S) propagation algorithm [41] used in `gRain`, then the performance is dependent on finding an optimal triangulation in the BN. However, searching for an optimal triangulation is NP-complete [42], but we can consider heuristics to approach this problem. To provide an example, according to the documentation of `gRain`, it uses the minimum clique weight heuristic by Kjærulff [43], [44]. We recommend using heuristics to avoid performance degradation in exact inference algorithms since we add new random



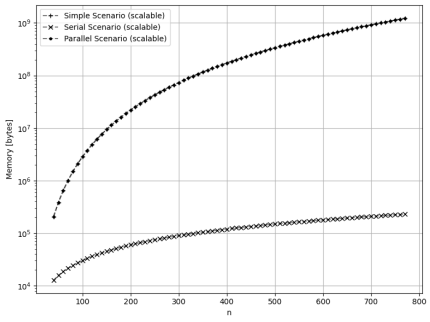
(a) Number fo CPT entries of naive and scalable scenarios for n up to 30.



(b) Memory size of the scenarios with the scalable BN structures for n up to 700.



(c) Number fo CPT entries of naive and scalable scenarios for n up to 760.



(d) Memory size of the scenarios with the scalable BN structures for n up to 700.

Fig. 12: Number of CPT and memory size of naive and scalable BN for all three scenarios.

variables with the scalable k/n model, thus increasing the BN complexity.

Generally, the scalable k/n model is applicable whenever representing the FT's k/n voting gate and is suitable for independent input events, with different constant failure rates. However, we need to resort to the naive implementation if the parent (cause) nodes, i.e., the input events of the gate, are not causally independent [38]. This is a special case when one wants to model a voting gate with noise, and the noise depends on some *specific* state combinations of the input events. Causal independence is obtained when each parent node has a separate influence on the child node's probability distribution. For the standard k/n voting gate and the here presented noisy k/n voting gate, the input events are causally independent. So, suppose the CPT of a child node representing a gate cannot be related to the individual contributions of each input event. In that case, the causal independence assumption might be invalid, and we cannot use the scalable implementation. In this case, we have to use the naive implementation instead of expressing the fault relations for each state combination individually.

IX. Conclusion and Future Work

In this work, we presented a scalable k/n voting gate representation for BNs, which enables us to assess system dependability with hundreds of redundant components. We showed that a naive implementation of the k/n voting gate for BNs is impractical, due to the exponential growth of the CPT for large n. To solve this problem, we proposed a scalable k/n model that reduces the space complexity from exponential to polynomial by exploiting the causal independence between input events. Our solution is an extension of the temporal noisy adder for BNs, where we count the number of fault occurrences and check if they exceeded the required number of faults. In our evaluation, we compared the inference time and memory size of the naive with the scalable k/n voting gate implementation for three complex systems in different size variations. Our experimental results show a significant decrease in memory space while still computing the same availability and preserving its compatibility with standard inference algorithms.

Currently, all gates have a scalable BN model for static FTs. As future work, we plan to provide scalable BN representations for gates of important FT extensions, such as dynamic FTs. Dynamic FTs provide new gates that have temporal properties. While dynamic BN solutions that represent dynamic FTs are already available, scalable implementations for the new gates do not exist. However, since dynamic FTs use the dynamic BN formalism, we might face new modeling challenges and scalability issues. We therefore consider these challenges for future work.

X. Acknowledgments

This work was supported by the Robert Bosch GmbH.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*. ACM Press, 2012. doi: 10.1145/2342509.2342513 pp. 13–16.

[2] N. Mishra, V. Kumar, and G. Bhardwaj, "Role of cloud computing in smart grid," in *2019 International Conference on Automation, Computational and Technology Management (ICACTM)*. IEEE, apr 2019. doi: 10.1109/icactm.2019.8776750. ISSN null pp. 252–255.

[3] B. Yin, L. Mei, Z. Jiang, and K. Wang, "Joint cloud collaboration mechanism between vehicle clouds based on blockchain," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, apr 2019. doi: 10.1109/sose.2019.00039. ISSN 2640-8228 pp. 227–2275.

[4] J. Kang, D. Lin, E. Bertino, and O. Tonguz, "From autonomous vehicles to vehicular clouds: Challenges of management, security and dependability," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, jul 2019. doi: 10.1109/icdcs.2019.00172. ISSN 1063-6927 pp. 1730–1741.

[5] N. Sultan, "Making use of cloud computing for healthcare provision: Opportunities and challenges," *International Journal of Information Management*, vol. 34, no. 2, pp. 177–184, apr 2014. doi: 10.1016/j.ijinfomgt.2013.12.011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0268401213001680

[6] F. Shrouf, J. Ordieres, and G. Miragliotta, "Smart factories in industry 4.0: A review of the concept and of energy management approached in production based on the internet of things paradigm," in *2014 IEEE International Conference on Industrial Engineering and Engineering Management*. IEEE, dec 2014. doi: 10.1109/ieem.2014.7058728. ISSN 2157-362X pp. 697–701.

[7] H. Choi, J. Song, and K. Yi, "Brightics-IoT: Towards effective industrial IoT platforms for connected smart factories," in *2018 IEEE International Conference on Industrial Internet (ICII)*. IEEE, oct 2018. doi: 10.1109/icii.2018.00024. ISSN null pp. 146–152.

[8] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. Minarick, and J. Railsback, "Fault tree handbook with aerospace applications," *Office of safety and mission assurance NASA headquarters*, 2002.

[9] E. Ruijters and M. Stoelinga, "Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools," *Computer science review*, vol. 15, pp. 29–62, 2015.

[10] Y. Zhang and W. Weng, "Bayesian network model for buried gas pipeline failure analysis caused by corrosion and external interference," *Reliability Engineering & System Safety*, vol. 203, p. 107089, nov 2020. doi: 10.1016/j.ress.2020.107089

[11] O. Kammouh, P. Gardoni, and G. P. Cimellaro, "Probabilistic framework to evaluate the resilience of engineering systems using bayesian and dynamic bayesian networks," *Reliability Engineering & System Safety*, vol. 198, p. 106813, jun 2020. doi: 10.1016/j.ress.2020.106813

[12] W. Xiang and W. Zhou, "Bayesian network model for predicting probability of third-party damage to underground pipelines and learning model parameters from incomplete datasets," *Reliability Engineering & System Safety*, vol. 205, p. 107262, jan 2021. doi: 10.1016/j.ress.2020.107262. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832020307614

[13] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 993–1002, 1996. doi: 10.1109/12.537122

[14] K. Stecher, "Evaluation of large fault-trees with repeated events using an efficient bottom-up algorithm," *IEEE Transactions on Reliability*, vol. 35, no. 1, pp. 51–58, April 1986. doi: 10.1109/tr.1986.4335344

[15] K. D. Rao, V. Gopika, V. S. Rao, H. Kushwaha, A. Verma, and A. Srividya, "Dynamic fault tree analysis using monte carlo simulation in probabilistic safety assessment," *Reliability Engineering & System Safety*, vol. 94, no. 4, pp. 872–883, apr 2009. doi: 10.1016/j.ress.2008.09.007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832008002354

[16] G. Merle, J.-M. Roussel, J.-J. Lesage, and A. Bobbio, "Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events," *IEEE Transactions on Reliability*, vol. 59, no. 1, pp. 250–261, mar 2010. doi: 10.1109/tr.2009.2035793

[17] A. Bobbio, L. Portinale, M. Minichino, and E. Ciancamerla, "Improving the analysis of dependable systems by mapping fault trees into bayesian

networks," *Reliability Engineering & System Safety*, vol. 71, no. 3, pp. 249–260, mar 2001. doi: 10.1016/s0951-8320(00)00077-6

[18] H. Boudali and J. Dugan, "A discrete-time bayesian network reliability modeling and analysis framework," *Reliability Engineering & System Safety*, vol. 87, no. 3, pp. 337–349, mar 2005. doi: 10.1016/j.ress.2004.06.004

[19] R. xing Duan and H. lin Zhou, "A new fault diagnosis method based on fault tree and bayesian networks," *Energy Procedia*, vol. 17, pp. 1376–1382, 2012. doi: 10.1016/j.egypro.2012.02.255

[20] M. Bensi, A. D. Kiureghian, and D. Straub, "Efficient bayesian network modeling of systems," *Reliability Engineering & System Safety*, vol. 112, pp. 200–213, apr 2013. doi: 10.1016/j.ress.2012.11.017

[21] J. G. Torres-Toledano and L. E. Sucar, "Bayesian networks for reliability analysis of complex systems," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1998, pp. 195–206.

[22] P. Weber and L. Jouffe, "Complex system reliability modelling with dynamic object oriented bayesian networks (DOOBN)," *Reliability Engineering & System Safety*, vol. 91, no. 2, pp. 149 – 162, feb 2006. doi: 10.1016/j.ress.2005.03.006 Selected Papers Presented at QUALITA 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832005000967

[23] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, apr 2010. doi: 10.1145/1773912.1773922. [Online]. Available: https://doi.org/10.1145/1773912.1773922

[24] D. Heckerman, "Causal independence for knowledge acquisition and inference," in *Uncertainty in Artificial Intelligence*. Elsevier, 1993, pp. 122–127.

[25] J. Kim and J. Pearl, "A computational model for causal and diagnostic reasoning in inference systems," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (I)*, vol. 1, 1983, pp. 190–193.

[26] J. Pearl, "Probabilistic reasoning in intelligent systems: Networks of plausible reasoning," *Morgan Kaufmann Publishers, Los Altos*, 1988.

[27] O. Bibartiu, F. Durr, K. Rothermel, B. Ottenwalder, and A. Grau, "Towards scalable k-out-of-n models for assessing the reliability of large-scale function-as-a-service systems with bayesian networks," in *Proceedings of the 12th IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, jul 2019. doi: 10.1109/cloud.2019.00095. ISSN 2159-6182 pp. 514–516.

[28] H. Boudali and J. Dugan, "A new bayesian network approach to solve dynamic fault trees," in *Proceedings of the Annual Reliability and Maintainability Symposium, 2005*, IEEE. IEEE, 2005. doi: 10.1109/rams.2005.1408404 pp. 451–456.

[29] K. G. Olesen and S. Andreassen, "Specification of models in large expert systems based on causal probabilistic networks," *Artificial Intelligence in Medicine*, vol. 5, no. 3, pp. 269–281, 1993. doi: 10.1016/0933-3657(93)90029-3

[30] K. Laskey, "Sensitivity analysis for probability assessments in bayesian networks," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 6, pp. 901–909, jun 1995. doi: 10.1109/21.384252

[31] S. Srinivas, "A generalization of the noisy-or model," in *Uncertainty in Artificial Intelligence*. Elsevier, 1993, pp. 208–215.

[32] K. Zhou, A. Martin, and Q. Pan, "The belief noisy-OR model applied to network reliability analysis," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 24, no. 06, pp. 937–960, nov 2016. doi: 10.1142/s0218488516500434

[33] F. J. Diez, "Parameter adjustment in Bayes networks. The generalized noisy OR–gate," in *Uncertainty in Artificial Intelligence, 1993*. Elsevier, 1993, pp. 99–105.

[34] P. Dagum and M. Luby, "Approximating probabilistic inference in bayesian belief networks is NP-hard," *Artificial Intelligence*, vol. 60, no. 1, pp. 141–153, mar 1993. doi: 10.1016/0004-3702(93)90036-b

[35] P. Dagum and A. Galper, "Additive belief-network models," in *Uncertainty in Artificial Intelligence*. Elsevier, 1993, pp. 91–98.

[36] I. Tien and A. D. Kiureghian, "Algorithms for bayesian network modeling and reliability assessment of infrastructure systems," *Reliability Engineering & System Safety*, vol. 156, pp. 134–147, dec 2016. doi: 10.1016/j.ress.2016.07.022

[37] G. F. Cooper, "The computational complexity of probabilistic inference using bayesian belief networks," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 393–405, mar 1990. doi: 10.1016/0004-3702(90)90060-d

[38] N. L. Zhang and D. Poole, "Exploiting causal independence in bayesian network inference," *Journal of Artificial Intelligence Research*, vol. 5, pp. 301–328, dec 1996. doi: 10.1613/jair.305

[39] M. Scutari, "Learning bayesian networks with thebnlearnRPackage," *Journal of Statistical Software*, vol. 35, no. 3, 2010. doi: 10.18637/jss.v035.i03

[40] S. Højsgaard, "Bayesian networks in R with the gRain package," *Rel téc Aalborg University*, pp. 1–15, 2015.

[41] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 50, no. 2, pp. 157–224, jan 1988. doi: 10.1111/j.2517-6161.1988.tb01721.x

[42] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[43] S. H. jsgaard, "Graphical independence networks with thegRainPackage forR," *Journal of Statistical Software*, vol. 46, no. 10, pp. 1–26, 2012. doi: 10.18637/jss.v046.i10. [Online]. Available: https://www.jstatsoft.org/v046/i10

[44] U. Kjærulff, "Triangulation of graphs–algorithms giving small total state space," 1990.