Computing (2012) 94:313–324
DOI 10.1007/s00607-011-0179-3

A verified realization of a Dempster–Shafer based fault tree analysis

Gabor Rebner · Ekaterina Auer · Wolfram Luther

Received: 31 March 2011 / Accepted: 18 November 2011 / Published online: 30 November 2011

Abstract Fault tree analysis is a method to determine the likelihood of a system attaining an undesirable state based on the information about its lower level parts. However, conventional approaches cannot process imprecise or incomplete data. There are a number of ways to solve this problem. In this paper, we will consider the one that is based on the Dempster–Shafer theory. The major advantage of the techniques proposed here is the use of verified methods (in particular, interval analysis) to handle Dempster–Shafer structures in an efficient and consistent way. First, we concentrate on DSI (Dempster–Shafer with intervals), a recently developed tool. It is written in MATLAB and serves as a basis for a new add-on for Dempster–Shafer based fault tree analysis. This new add-on will be described in detail in the second part of our paper. Here, we propagate experts' statements with uncertainties through fault trees, using mixing based on arithmetic averaging. Furthermore, we introduce an implementation of the interval scale based algorithm for estimating system reliability, extended by new input distributions.

Keywords Dempster–Shafer theory · Fault tree analysis · MATLAB · INTLAB · Interval analysis

Mathematics Subject Classification (2000) 65G30 · 60A99

The authors have presented the results of this paper during the SCAN 2010 conference in Lyon, September 2010.

G. Rebner (✉) · E. Auer · W. Luther
University of Duisburg-Essen, 47048 Duisburg, Germany
e-mail: rebner@inf.uni-due.de

1 Introduction

An important task in engineering is to accurately specify the point of time after which a given system attains an undesirable state, for example, the number of hours after which it fails. Fault tree analysis is an approach to compute such characteristics for the whole system from the corresponding characteristics of its low level components. Besides unavoidable discretization errors, traditional numerical implementations based on floating point (FP) arithmetic have several limitations. One of them is the possibility of rounding errors. They arise inevitably on a computer because of the finite nature of machine numbers as opposed to real numbers, and they might lead to severe failures under unfavorable circumstances [1]. Another difficulty is that the above mentioned low level characteristics, for example, failure probabilities, are not known exactly. They might vary depending, for instance, on measurement precision or the uncertainty in statements of experts.

Imprecision can be characterized by providing upper and lower bounds on all possible results using interval analysis [14] or other verified methods. Further options are probability theory, Dempster–Shafer theory (DST), or the Bayes theory. In this paper, we concentrate on the use of DST [16], the significance of which for modeling and propagating uncertainty has grown recently. The newly implemented software DSI (Dempster–Shafer with intervals) [2] builds the basis for our research. It differs from the few existing implementations [10,12] in a number of ways. The most important one is the consistent use of interval analysis to deal with rounding errors and appearing sets.

A method is called verified, if it guarantees the correctness of its output. In this context, interval arithmetic is a widely used approach to verifying results obtained on a computer. It provides a (multidimensional) box—described in terms of FP arithmetic—that is proved to contain the exact result.

Our goal was to combine the advantages of verified methods with fault tree analysis. In this paper, we show a method to analyze failure probabilities using verified DST structures and a scale based method relying on interval analysis.

Accordingly, our paper is structured as follows. In Sect. 2, we provide a short overview of the three concepts important for the understanding of this paper: interval analysis, DST, and fault tree analysis. On their basis, we give an overview of working with DST using DSI and compare it to the older tool IPP [10] from which it originates in Sect. 3. Over the course of the next section, we introduce the recently developed add-on for fault tree analysis and illustrate its verified implementation in MATLAB with INTLAB [15] using an example. Furthermore, we consider a robotic system and compute a lower and upper bound of failure probability at a certain point of the system's operation time, using a verified implementation of the algorithm introduced in [18]. We extended the basic version by adding the possibility to choose as an input distribution not only the trapezoid one, but also the beta and the normal distribution. We conclude by recapitulating the main results and providing a perspective on future research.

2 Basic theory

The algorithms in this paper combine interval and DST methods with fault tree analysis. In this section, we overview briefly the basic ideas of these three research areas.

2.1 Interval analysis

Interval analysis is a field of numerics with applications in engineering, robotics, or medicine. It belongs to the group of verified methods that can guarantee the correctness of an outcome of a simulation using mathematically exact proofs. Let $\mathbf{x} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ be a real interval in infimum–supremum notation, which is defined as $[\underline{\mathbf{x}}, \overline{\mathbf{x}}] = \{x \in \mathbb{R} | \underline{\mathbf{x}} \leq x \leq \overline{\mathbf{x}}\}$. An interval operation $\circ = \{+, -, \cdot, /\}$ is defined for the intervals $\mathbf{x}$ and $\mathbf{y}$ as

$$\mathbf{x} \circ \mathbf{y} = [min\{\underline{\mathbf{x}} \circ \underline{\mathbf{y}}, \underline{\mathbf{x}} \circ \overline{\mathbf{y}}, \overline{\mathbf{x}} \circ \underline{\mathbf{y}}, \overline{\mathbf{x}} \circ \overline{\mathbf{y}}\}, max\{\underline{\mathbf{x}} \circ \underline{\mathbf{y}}, \underline{\mathbf{x}} \circ \overline{\mathbf{y}}, \overline{\mathbf{x}} \circ \underline{\mathbf{y}}, \overline{\mathbf{x}} \circ \overline{\mathbf{y}}\}].$$

The result is an interval that includes all possible combinations of $x \circ y$ with $x \in [\underline{\mathbf{x}}, \overline{\mathbf{x}}]$ and $y \in [\underline{\mathbf{y}}, \overline{\mathbf{y}}]$. In case of division, $\mathbf{y}$ is in general assumed not to contain zero. These notations can be extended to interval vectors and matrices. To indicate interval computations, we use the notation $\boxed{\circ} = \left\{\boxed{+}, \boxed{-}, \boxed{\cdot}, \boxed{/}\right\}$.

To obtain verified results on a computer relying on finite FP arithmetic, the concept of a machine interval has to be introduced. A machine interval can be obtained from a real interval $\mathbf{x}$ by rounding the lower bound downward ($\mathbf{fl}_{\triangledown}\underline{\mathbf{x}}$) and the upper bound upward ($\mathbf{fl}_{\triangle}\overline{\mathbf{x}}$) to the next FP number. Throughout this paper, we use the interval arithmetic implementation supplied by INTLAB. For example, we employ elementary operations and functions to compute natural interval evaluations of non-monotonic functions with uncertainty. This allows us to dispense with optimization methods that are usually used to compute the range. It makes our computations faster and additionally guarantees that the solution is enclosed. The problem of overestimation is addressed in Sect. 3.

2.2 Dempster–Shafer theory

Dempster–Shafer theory [16] is a mathematical theory of evidence developed to combine information supplied by different experts or from other sources. It provides a measure of confidence that a given event occurs. A special feature of this theory is the possibility to characterize uncertainties arising because of lack of knowledge as discrete probability assignments associated with the power set 2^X of a given set X. A mass assignment to each subset of 2^X is known as the basic probability assignment (BPA) m. If $A_1, \ldots, A_n$ are the sets of interest with $A_i \in 2^X$, then a BPA is defined as

$$m : 2^X \rightarrow [0, 1], \quad \sum_{i=1}^{n} m(A_i) = 1, \quad m(\emptyset) = 0. \tag{1}$$

In the continuous case, we assume that each set A_i is an interval. Intervals with masses greater than zero are known as focal elements and can be viewed as evidence given by experts. To work with such structures, the plausibility and the belief functions are defined over all focal elements as

$$PL(Y) := \sum_{A_i \cap Y \neq \emptyset} m(A_i), \quad BEL(Y) := \sum_{A_i \subseteq Y} m(A_i) \quad \text{with} \quad Y \subseteq X. \tag{2}$$

Nevertheless, the mass has to be normalized because real life experts tend to ignore the restriction in Eq. (1). An example of verified normalization is given in Sect. 3. If several experts give their estimations in the same context, BPAs have to be aggregated. A number of aggregation laws have been published [6], of which we use mixing based on arithmetic averaging in this paper.

Furthermore, a BPA is associated with a probability box (p-box). Ferson [5] defines a p-box as an enclosure of a cumulative distribution function (CDF) which is characterized by two non-decreasing functions:

$$\underline{F} = \mathbb{R} \to [0, 1] \tag{3}$$

$$\overline{F} = \mathbb{R} \to [0, 1] \tag{4}$$

$$\underline{F}(x) \leq \overline{F}(x). \tag{5}$$

In this paper, we employ finite BPAs with focal elements expressed as closed real intervals. We obtain the associated p-box from a BPA A with n focal elements $(A_i = ([\underline{A}_i, \overline{A}_i], m(A_i)) \mid i = 1 \ldots n)$ by using the formulas [6]

$$\underline{F}(x) = \sum_{\overline{A}_i < x} m(A_i) \tag{6}$$

$$\overline{F}(x) = \sum_{\underline{A}_i \leq x} m(A_i). \tag{7}$$

In the context of DST, we can assume that formula (6) expresses the cumulative believe function and formula (7) the cumulative plausibility function.

In this subsection, we introduced two representations of BPAs, which can be used to implement an interface to RAMAS Risk Calc [5]. In the future, we plan to compare the results of Risk Calc with DSI.

2.3 Fault tree analysis

Fault tree analysis is a method to examine a system with respect to the likelihood (from the probabilistic point of view) of it attaining an undesirable state based on the information about its lower level parts. A fault tree consists of basic events, logical gates and one top event. A *basic event* is a part of a system that is not subdivided into smaller parts. A *logical gate* (in our case, either the AND or the OR gate) combines failure probabilities from different sources. The *top event* is at the head of the tree and represents the main failure of the system.

Usually, probabilities are assigned to the basic events and propagated through the gates to the top event using boolean logic. We discuss how to integrate imprecise information about such probabilities into a fault tree using DST in Sect. 4.

3 DSI toolbox

The focus of this section is the DSI toolbox, which implements DST in a verified way. It is realized in MATLAB extended by INTLAB. Functionalities of DSI include

Table 1 Averaged CPU times in seconds

Benchmark	IPP	DSI
Distribution sampling	1.37725	0.19144
Aggregation	0.50923	0.00837
Non-monotonic functions	2.68565	1.44652

generation of BPAs from uncertain data and several kinds of distributions in a verified way. BPAs can be aggregated and propagated through various system functions, and results can be represented graphically afterwards.

In our toolbox, we assume that experts give independent evidence. To compute solutions based on dependent evidence, a user has to employ other tools, for example, Risk Calc that provides a p-box circumscribing all possibilities. In the future, we plan to enhance our toolbox with this option too.

DSI is based on IPP [10] for R, which makes no use of interval analysis or directed rounding. For a more detailed description of DSI key features (see [2]). In this section, we explain in what sense our implementation is verified using an example and compare DSI to IPP.

Our implementation verifies the results of DST computations by taking rounding and conversion errors into account. The meaning of verification in our context can be illustrated by our interpretation of the normalization. If a BPA consists of n focal elements $A_1, \ldots, A_n$, the sum of all masses has to equal one. If this condition does not hold, the masses have to be normalized in a verified way. To do so, each mass of a focal element is represented by a point interval of the form $\mathbf{m}(A_i) = [\underline{\mathbf{m}(A_i)}, \overline{\mathbf{m}(A_i)}]$. To obtain a normalized mass $\mathbf{m}_{\text{new}}(A_i)$ for $i = 1 \ldots n$, we compute the hull of the two intervals

$$\underline{\mathbf{m}(A_i)} \boxed{/} \left(\mathbf{fl}_{\triangle} \boxed{\sum_{j=1}^{n}} \underline{\mathbf{m}(A_j)} \right), \quad \overline{\mathbf{m}(A_i)} \boxed{/} \left(\mathbf{fl}_{\triangledown} \boxed{\sum_{j=1}^{n}} \overline{\mathbf{m}(A_j)} \right).$$

In DSI, users can sample CDFs with uncertain parameters by an outer discretization method [17]. The number of samples influences the unavoidable discretization error. More samples lead to tighter enclosures and a lower discretization error. Each CDF can be used to model the probability of a basic event in fault tree analysis.

CDFs can be used to work with system models described by (non-)monotonic functions. In contrast to IPP, there is no need to use optimization routines to compute the (exact) range of a function over a given interval. In DSI, we use INTLAB interval operations, standard functions, and their combinations to obtain an enclosure of the range, which is faster than the implementation in IPP, especially for non-monotonic functions. To work with a system described by such a function, the user has to specify the number of Monte-Carlo samples, a CDF, and the function itself.

In Table 1, we compare IPP and DSI with respect to the CPU time for generating the triangle, the uniform and the Gumbel CDF, each over 2^{10} Monte-Carlo samples (row 2), aggregating them by using Dempster's rule, unweighted and weighted mixing (row 3) and evaluating ranges of non-monotonic functions over the obtained intervals (the last row). We used four non-monotonic functions, $\sin(x^2)$, $\cos(x^2)$, $\cos(x^2) + \sin(y)$

and the Rosenbrock function $f(x, y) = (1 - x)^2 + 100 \cdot (y - x^2)^2$. In each case, we called the corresponding routines up to 200 times and averaged the resulting CPU times. The times are obtained on an Intel® Core2™ Quad Q9450 CPU at 2.66 GHz with MATLAB 7.11.0.

The comparison shows that DSI is faster than IPP, even though interval arithmetic and directed rounding are in use. Furthermore, this (non-)monotonic function propagation can be used to model probability for basic events in fault tree analysis (see Sect. 4.1).

However, naive interval evaluation might overestimate the true solution set considerably. To deal with this problem, we implemented a range of options. For example, we test functions for monotonicity using a method based an algorithmic differentiation. To compute a tight enclosure of the range of a function which is monotonic and contains exclusively basic arithmetic operations $\{+, -, *, /\}$ and their compositions, we utilize FP arithmetic with directed rounding. This kind of computation can be used because of the definition of basic arithmetic operations and their rounding behavior in the FP standard IEEE754 [9]. Otherwise, we have to use interval arithmetic to get a verified enclosure of the solution.

A list of all functions available in DSI can be found at http://udue.de/DSI.

4 Fault tree analysis add-on for DSI

We implemented two possible approaches to estimating the reliability of a system described by a fault tree. The first one is to analyze the whole lifetime of a device based on expert estimations of the corresponding failure probabilities (see Sect. 4.1). Here, we extend the approach from [4,8] to compute a p-box that is bounded by the belief and the plausibility functions. The second approach is to assess the likelihood of a failure at a certain point in the lifetime of the device based on interval analysis (see Sect. 4.2). We implement the approach from [18] using DSI and extend it with two further models to represent the input uncertainty. The functionalities from Sect. 3 form the basis of our implementation.

4.1 Fault tree analysis with Dempster–Shafer

The aim of this approach is the computation of belief and plausibility at the top event. For each basic event, two cases can be distinguished. If the corresponding estimation depends on one source only (a BPA), there is no need for aggregation. However, it becomes necessary if more than one estimation is given. In this situation, we use unweighted mixing based on arithmetic averaging to avoid information loss. The corresponding formula for aggregating q finite random sets with BPAs $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_q$ is shown below:

$$\mathbf{m}^{\text{new}}(A_i) = \left(\boxed{\sum_{j=1}^{q}} \mathbf{m}_j(A_i) \right) \boxed{/} \, q \,, \quad i = 1 \ldots n. \tag{8}$$

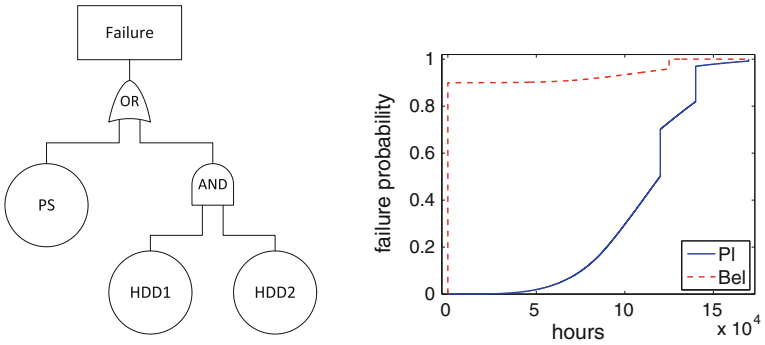

Fig. 1 Fault tree of a small RAID (*left*) and belief and plausibility of its failure (*right*)

We use BPAs to model estimations. The basic events can be defined by using CDFs and (non-)monotonic functions.
To propagate the failure probabilities of the basic events to the top event, the extended concept of logical gates from [4,8] is adapted to interval arithmetic. Generally, the formulas for the AND and the OR gate are the following:

$$P(A \wedge B) = P(A) \cdot P(B), \tag{9}$$

$$P(A \vee B) = 1 - (1 - P(A)) \cdot (1 - P(B)), \tag{10}$$

where A and B are events, which we can interpret as continuous parts of lifetime in hours, and $P(X)$ gives the failure probability of X. If A and B are BPAs, Eq. (9) turns into [4]:

$$\begin{aligned} BEL(A \wedge B) &= BEL(A) \boxed{\cdot} BEL(B) \\ PL(A \wedge B) &= PL(A) \boxed{\cdot} PL(B). \end{aligned} \tag{11}$$

The general formula to propagate uncertainties through the OR gate is

$$\begin{aligned} BEL(A \vee B) &= 1 \boxed{-} (1 \boxed{-} BEL(A)) \boxed{\cdot} (1 \boxed{-} BEL(B)) \\ PL(A \vee B) &= 1 \boxed{-} (1 \boxed{-} PL(A)) \boxed{\cdot} (1 \boxed{-} PL(B)). \end{aligned} \tag{12}$$

In DSI, we implemented the routines `dsiand` and `disor` for that purpose. As an example of how this works, consider a fault tree of a small redundant array of independent disks (RAID) (shown in Fig. 1, left), which is composed of two hard disc drives (HDD) and one power supply (PS). The system fails, if the PS or both HDDs fail. Furthermore, we assume that the HDDs have the same specification. The distribution on the time to failure for the power supply is assessed by two experts. The first expert estimates it to be in the interval $[0, 14 \times 10^4]$ hours. The second expert states that the time to failure for the PS is between 0 and 12×10^4 h of work with 80% confidence and between 12.5×10^4 and 17×10^4 h with 20% confidence. We use unweighted

mixing to aggregate these pieces of evidence without loss of information. The third expert estimates the failure probability for HDDs to follow a triangular distribution between zero and 20×10^4 h, with an unsure mode of $[8, 9] \times 10^4$.

The belief and plausibility of the failure at the top event are shown in Fig. 1, right. The RAID will fail after 17×10^4 h with belief and plausibility of 100%. This example illustrates that the belief function represents the worst case scenario as it rises directly after 0 h while the plausibility function is still near zero.

4.2 Interval based approach

The purpose of this subsection is to introduce a verified implementation for computing the lower and the upper bound of failure probability similar to [18]. In contrast to the previous subsection, this algorithm is based on the interval propagation of meta-information about failure probabilities themselves.

The quality of the result at the top event of a fault tree that is produced by the model we use correlates strongly with the CPU time needed to obtain it. The interval $p = [0, 1]$ of failure probabilities is split into s intervals to which a certain weight probability is assigned. The higher the number s is, the less conservative the obtained bounds are, but the more computing time is required. As a trade-off between the quality and the performance, the first k intervals (situated near zero, where it matters most) are further subdivided into l segments each to minimize possible pessimism in the estimation. To work with such elements at the basic events, we implemented the class `scale`, which used a matrix to store the intervals and the corresponding probabilities.

Consider a robot described in detail in [3]. Each joint of a robot consists of a motor and two sensors working independently. The top event is reached if two of the joints fail. This occurs if either the motor or both sensors of a joint are not functioning. Two basic events, motor and sensor, are necessary for the fault tree. For example, if we want to associate a `scale` object with the motor, we might call the constructor `motor = scale(s,k,l,bound)` with `bound` $\in \{LB, UB\}$ for the lower or upper bound computation. It subdivides the whole probability interval into $(s-k)+k \cdot l$ segments. Usually, a trapezoid distribution is used to assign weight probabilities to these segments. We added the possibility to employ two further distributions for this purpose (cf. Sect. 4.3). For such an assignment, the user should call the routine `fillscale`, a member function of the class `scale`.

The `scale` objects are then propagated through logical gates of the fault tree. In general, the usual laws are used for the gates. For example, if **x** and **y** are scale segments of independent basic events with weight probabilities f_x, f_y that are to be propagated through an AND gate, the result is the interval $[\underline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}]$ with the probability of $f_x \cdot f_y$. After that, the new intervals and their weights have to be fitted into the original segmentation of $(s - k) + k \cdot l$ intervals. The rules and their illustrations are to be found in [18]. To guarantee the correctness of the obtained bounds, it is necessary to round downwards while calculating intervals and upwards while computing weight probabilities, if we want to obtain the lower bound on the failure probability, and vice versa for the upper bound. The roundings do not depend on the kind of gate.

Suppose a factory produced a certain amount of robots. Our goal is to find out with which probability a given percentage of them does not fail in the first 1,000 h of oper-

Table 2 Comparison of the failure probabilities for the robot

x (%)	Low. bound [18]	Up. bound [18]	Lower bound	Upper bound	Point [11]
5	1.48×10^{-4}	1.60×10^{-4}	1.34×10^{-4}	1.574×10^{-4}	1.6×10^{-4}
16	3.6×10^{-4}	3.74×10^{-4}	3.4×10^{-4}	3.7×10^{-4}	3.7×10^{-4}
25	5.6×10^{-4}	5.76×10^{-4}	5.4×10^{-4}	5.7×10^{-4}	5.68×10^{-4}
38	9.46×10^{-4}	9.68×10^{-4}	9.2×10^{-4}	9.6×10^{-4}	9.54×10^{-4}
45	1.224×10^{-3}	1.248×10^{-3}	1.186×10^{-3}	1.24×10^{-3}	1.2×10^{-3}
84	7.58×10^{-3}	7.63×10^{-3}	7.47×10^{-3}	7.6×10^{-3}	7.8×10^{-3}
95	2.78×10^{-2}	2.84×10^{-2}	2.68×10^{-2}	2.82×10^{-2}	2.82×10^{-2}

Table 3 Comparison of the computation time with and without parallelization (in hours)

Benchmark	Total CPU time	Real time
Without parallelization	35.1	35.1
With parallelization	35.7	11.71

ation. We simulate the robot example using the scale with $s = 5{,}000$, $k = 100$, and $l = 60$ using the same input data as in [18] to be able to compare results. In Table 2, the upper and lower bounds obtained in DSI (columns 4 and 5) are shown in juxtaposition to those from [18] (columns 2 and 3). Additionally, we compare them with the FP results from [11] (column 6). The numbers in Table 2 indicate that x percent of the weight probability for the failure is located in the interval $[0, p]$. The value p is the true failure probability enclosed between its computed lower and upper bounds.

The results in Table 2 are interpreted as follows. For 95% of the produced robots, an error probability of $p = [2.68 \times 10^{-2},\ 2.82 \times 10^{-2}]$ (columns 4 and 5) is obtained. This implies that p percent of 95% of all robots will fail before 1,000 h. This means that $0.95 - 0.95 \cdot p = [92.32, 92.454]$ percent of all produced robots will not fail in the first 1,000 h of their operation time.

Our values for the upper bound follow closely the values published in [11] (the value at 84% in column 6 seems to be a misprint), whereas there is a small downward shift in comparison to the enclosures in [18]. Note that the results are nonetheless consistent because their intersection is not empty.

Furthermore, we had to optimize these routines, because the complexity of the presented algorithms leads to a long computation time. To speed up the implementation, we used the "Parallel Computing toolbox" [13] for MATLAB to parallelize the computations on the four available cores. We obtained an improvement of factor 2.99 for real time computation for the example of the robot. The computation times with and without optimization are given in Table 3.The bounds are obtained by utilizing the same computer system as in Sect. 3.

4.3 Representing input failure probabilities

In the previous subsection, we assumed that the trapezoid distribution was used to obtain weight probabilities at the basic events. In our implementation, we can use two

further distributions, which are similar to the beta and the normal distribution. In case of these distributions, it is not necessary to verify all computations. It is sufficient to show that the sum of the probabilities of each interval is a tight enclosure of one.

The CDF of the beta distribution is well suited for our algorithm because it is defined in the interval [0, 1] and equals zero outside. Furthermore, both axis symmetric and asymmetric structures can be defined in dependence on two real parameters, $a > 0$ and $b > 0$. We utilize the build in function of MATLAB to compute the inclosure.

To define a distribution similar to the normal one in the domain [0, 1], we suggest to use the formula $f(x) = k \cdot \cos^{n-1}(a(x-m))$ for the corresponding probability density function (PDF). Now, we have to compute the parameters a, m, n and k, so that the integral of $f(x)$ over [0, 1] equals one ($k \cdot \int_0^1 \cos^{n-1}(a(x-m))\,dx = 1$). According to [7], the relation $\int_{m+\frac{\pi}{2a}}^{m-\frac{\pi}{2a}} \cos^{n-1}(a(x-m))\,dx = \frac{1}{\pi} \cdot 2^{n-1} \cdot \beta(\frac{n}{2}, \frac{n}{2}) =: \frac{1}{k}$ holds. In our case, $m + \frac{\pi}{2a} = 1$ and $m - \frac{\pi}{2a} = 0$ so that $m = 0.5$ and $a = \pi$. The mean μ of this distribution can be set to 0.5 because the domain is [0, 1] and the PDF is symmetric. The variance σ^2 is defined in dependence on n as:

$$k \cdot \underbrace{\int_0^1 \left(x - \frac{1}{2}\right)^2 \cdot cos^{n-1}\left(\pi\left(x - \frac{1}{2}\right)\right) dx}_{I(n)} = \sigma^2. \tag{13}$$

After substituting $y = \pi \cdot (x - \frac{1}{2})$ for x, we obtain

$$I(n) = \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \left(\frac{y}{\pi}\right)^2 \cdot \cos^{n-1}(y) \cdot \frac{1}{\pi}\,dy = \frac{2}{\pi^3} \int_0^{\frac{\pi}{2}} y^2 \cdot \cos^{n-1}(y)\,dy. \tag{14}$$

To compute $I(n)$, we use the formula $I(n) = \frac{2}{\pi^3} C(n-1)$ for $n > 0$, where

$$C(n-1) = \int_0^{\frac{\pi}{2}} x^2 \cdot \cos^{n-1}(x)\,dx = \frac{n-2}{n-1} \cdot C(n-3) - \frac{2}{(n-1)^2} \cdot D(n-1) \tag{15}$$

and

$$D(n-1) = \int_0^{\frac{\pi}{2}} \cos^{n-1}(x)\,dx = \frac{(n-2)!!}{(n-1)!!} \cdot \begin{cases} 1, & \text{if } n \text{ even} \\ \frac{\pi}{2}, & \text{if } n \text{ odd.} \end{cases} \tag{16}$$

The resulting distribution is defined by its CDF as

$$F(x, n) = \frac{\int_0^1 \cos\left(\pi\left(x - \frac{1}{2}\right)\right)^{n-1} dx}{\frac{1}{\pi} \cdot 2^{n-1} \cdot \beta\left(\frac{n}{2}, \frac{n}{2}\right)}. \tag{17}$$

In Fig. 2, a few examples of the beta and the normal-like distribution are shown for the sample scale with the same parameters as in Sect. 4.2 ($s = 5{,}000$, $k = 100$,

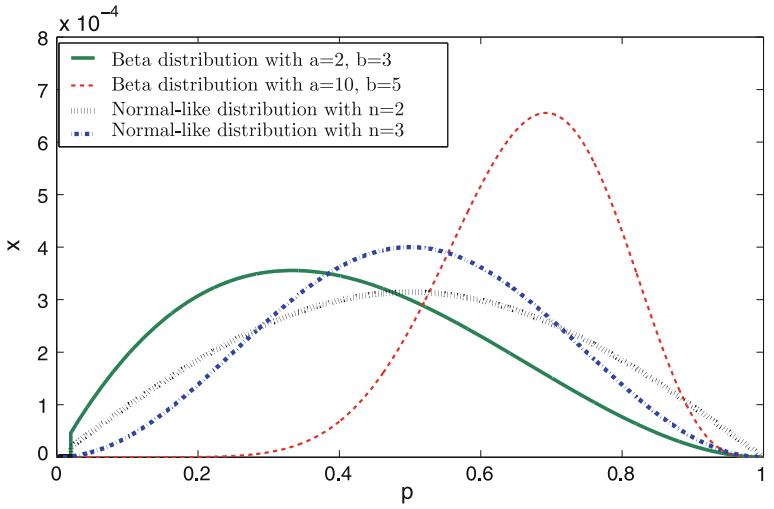

Fig. 2 Normal-like and beta distributions computed for $s = 5{,}000$, $k = 100$ and $l = 60$

Table 4 Failure probabilities for the robot with the beta and the normal distribution as input

x (%)	Lower bound	Upper bound
5	8.18×10^{-2}	8.38×10^{-2}
16	1.132×10^{-1}	1.154×10^{-1}
25	1.3116×10^{-1}	1.338×10^{-1}
38	1.556×10^{-1}	1.578×10^{-1}
45	1.682×10^{-1}	1.706×10^{-1}
84	2.63×10^{-1}	2.654×10^{-1}
95	3.296×10^{-1}	3.322×10^{-1}

$l = 60$). The graphs are obtained by plotting the midpoint of each interval against the weight of these intervals. Because of the small intervals in the fine scale, the probabilities of this intervals approach zero.

Consider the robot from Sect. 4.2 again. Now, we use a beta distribution with $a = 3$ and $b = 219$ for modeling the sensor failure probability and the normal distribution with $n = 3$ for the motor probability. The results obtained are shown in Table 4. In the case of $x = 95\%$, [63.440, 63.688] percent of all produced robots will not fail in the first 1,000 h of their operation time.

5 Conclusion and perspective

In this paper, we presented the new verified implementation DSI for DST in MATLAB. Additionally, we compared it to the older tool IPP with respect to CPU times. Based on DSI, we introduced an add-on for verified fault tree analysis with Dempster–Shafer structures. The advantage is the ability to compute verified failure probabilities with uncertainties in evidence of experts over a lifetime of a system. Further, we showed

how to compute the lower and the upper bound of the failure probability similar to the works of Carreras, Luther, and Traczinski [3,11,18]. We extended the original algorithm by the normal and the beta-like distributions to represent the input uncertainty. Finally, we demonstrated the functionality of the approach and its implementation in DSI using a descriptive example.

In our future work, we plan to apply the verified DST to Markov chains with uncertainty, which can be of use, for example, for modeling aging processes in SO fuel cells.

References

1. Arnold D (2011) Computer arithmetic tragedies. http://www.ima.umn.edu/~arnold/455.f96/disasters.html
2. Auer E, Luther W, Rebner G, Limbourg P (2010) A verified MATLAB toolbox for the Dempster-Shafer theory. In: Proceedings of the workshop on the theory of belief functions. http://www.udue.de/DSIPaperone, http://www.udue.de/DSI
3. Carreras C, Walker I (2001) Interval methods for fault-tree analyses in robotics. IEEE Trans Reliab 50:3–11. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00935010
4. Cheng Y (2000) Uncertainties in fault tree analysis. http://www2.tku.edu.tw/~tkjse/3-1/3-1-3.pdf
5. Ferson S (2002) RAMAS Risk Calc 4.0 software: risk assessment with uncertain numbers. Lewis Publishers, Boca Raton
6. Ferson S, Kreinovich V, Ginzburg L, Myers D, Sentz K (2003) Constructing probability boxes and Dempster–Shafer structures. SAND2002-4015. Sandia National Laboratories, Washington
7. Gradstein I, Ryshik I (1981) Summen, Produkt- und Integraltafeln. Harri Deutsch, Germany
8. Guth M (1991) A probability foundation for vagueness and imprecision in fault tree analysis. IEEE Trans Reliab 1(40):563–570
9. IEEE Standard for Floating-Point Arithmetic (2008) IEEE Std 754-2008, pp 1–58. doi:10.1109/IEEESTD.2008.4610935
10. Limbourg P (2011) Imprecise probability propagation toolbox (IPP toolbox). http://www.uni-due.de/il/ipptoolbox.php
11. Luther W, Dyllong E, Fausten D, Otten W, Traczinski H (2001) Numerical verification and validation of kinematics and dynamical models for flexible robots in complex environments. In: Perspectives on enclosure methods. Springer, Wien, pp 181–200
12. Martin A (2009) Implementing general belief function framework with a practical codification for low complexity. In: Smarandache F, Dezert J (eds) Advances and applications of DSmT for information fusion. Collected works, vol 3. American Research Press, Rehoboth, pp 217–273
13. MathWorks (2011) Parallel Computing toolbox. http://www.mathworks.de/products/parallel-computing/?s_cid=0210_webg_js_390079
14. Moore R, Kearfott B, Cloud M (2009) Introduction to interval analysis. Society for Industrial and Applied Mathematics, Philadelphia
15. Rump S (1999) INTLAB–INTerval LABoratory. Dev Reliab Comput 1:77–104. http://www.ti3.tu-harburg.de/
16. Shafer G (1976) A mathematical theory of evidence. Princeton University Press, Princeton
17. Tonon F (2004) Using random set theory to propagate epistemic uncertainty through a mechanical system. Reliab Eng Syst Saf 85(1–3):169–181
18. Traczinski H (2006) Integration von Algorithmen und Datentypen zur validierten Mehrkörpersimulation in MOBILE. Dissertation, Universität Duisburg-Essen. Logos-Verlag, Berlin. ISBN 978-3-8325-1457-0