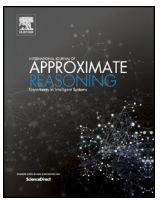
Beyond tree-shaped credal probabilistic circuits

International Journal of Approximate Reasoning 171 (2024) 109047

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

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



Beyond tree-shaped credal probabilistic circuits

David R. Montalván Hernández *, Tijn Centen, Thomas Krak, Erik Quaeghebeur, Cassio de Campos

Eindhoven University of Technology, Eindhoven, The Netherlands

ARTICLE INFO

Keywords:
Credal probabilistic circuits
Sum-product networks
Imprecise probabilities
Exact inference
Generative models
Graphical models

ABSTRACT

Probabilistic circuits are a class of probabilistic generative models that allow us to compute different types of probabilistic queries in polynomial time. Unlike many of the mainstream approaches for generative modeling, they can compute exact likelihoods, marginals, and expectations. Yet, assessing the reliability of their inferences is not straightforward. Credal probabilistic circuits are the imprecise counterpart of probabilistic circuits allowing, among other queries, computations of cautious inferences and sensitivity analyses. In this work, we propose an efficient algorithm to compute the lower and upper expectations for factorizing functions using a credal probabilistic circuit. We discuss under what structural assumptions and types of factorizing functions the algorithm works. We prove that such algorithm has polynomial time complexity in the input size. In the general case, we prove that computing cautious inferences using credal probabilistic circuits is an NP-hard problem, yet the proposed algorithm can be used as an approximation. Some experiments show how the approximation degrades with the complexity of the model structure.

1. Introduction

Probabilistic Graphical Models (PGMs) are a class of models using a graph-based representation to encode a high-dimensional distribution [10]. Probabilistic circuits, including the prominent sum-product networks, represent a type of PGMs [17,19]. Unlike Bayesian networks which have notoriously high complexity for general queries [6,7,18], probabilistic circuits can produce several types of inferences in polynomial time under arguably mild assumptions [9,20]. Sum-product networks are a type of probabilistic circuit which use circuits with three types of nodes: sum, product, and leaf (distribution) nodes. Sum nodes can be interpreted as latent variables [15,22], product nodes encode context-specific independences, and leaves encode (tractable) probability distributions. A sum-product network can be seen as a complex yet tractable mixture model exploiting some conditional independences.

Sum-product networks are *precise* probabilistic models, that is, they assume the existence of a *single* underlying probability distribution. Because of that, it has been claimed that inferences might be unreliable and overconfident [13,12]. One possible way for handling different kinds of uncertainties in probabilistic circuits is presented by Cerutti et al. [4]. In this work, each leaf node is assumed to represent a beta distribution and the imprecision is handled through the use of subjective logic together with a moment-matching approach. One of the limitations of the work of Cerutti et al. [4] is that circuits are constrained to domains with binary variables since the main goal is to run inferences in Probabilistic Logic Programs.

* Corresponding author.
E-mail addresses: d.r.montalvan.hernandez@tue.nl (D.R. Montalván Hernández), t.e.krak@tue.nl (T. Krak), e.quaeghebeur@tue.nl (E. Quaeghebeur), c.decampos@tue.nl (C. de Campos).

https://doi.org/10.1016/j.ijar.2023.109047
Received 3 March 2023; Received in revised form 31 July 2023; Accepted 1 October 2023

Available online 31 October 2023

Table 1
Cases where the algorithm from Section 3 computes exact lower/upper expectations (in polynomial time) according to the type of factorizing function and graph structure (please refer to Section 2.3 and Definition 6 for the precise definitions).

Factorizing function / Graph structure	Arbitrary	Tree	X_q-Queried
General (Theorem 2)	×	✓	×
Non-negative (Theorem 1)	✓	✓	✓
X_q-Focused (Theorem 3)	×	✓	✓

By using the framework of imprecise probabilities [21,1], Mauá et al. [12,13] introduce credal sum-product networks. A credal sum-product network extends a sum-product network by adding flexibility to the weights of each sum node. Namely, instead of having a single weight vector, the sum nodes in a credal sum-product network have weight vectors represented by a credal set (here assumed closed and convex). By adding flexibility to the parameters, one has a more expressive framework which for instance can help with assessing the confidence of the inferences, dealing with adversarial settings, as well as quantifying robustness of predictions. In addition to introducing credal sum-product networks, Mauá et al. [12,13] present an algorithm to compute likelihoods and conditional expectations in polynomial time when the underlying graph is a tree and the random variables are binary.

In this paper we extend the results for credal sum-product networks by proposing an algorithm for a broader class of graph structures (including some types of graphs whose underlying undirected graph has cycles) and a broader class of query functions based on their factorizations in smaller parts. We show the correctness and polynomial complexity of the new algorithm. Our approach also considers the more general case of continuous random variables. Table 1 shows the cases where our proposed algorithm (see Section 3) computes exact lower/upper expectations. We also show the limits of this algorithm by proving that the tasks become NP-hard if they go beyond the presented scope.

The rest of the document is organized as follows. In Section 2 we present notation, core concepts and assumptions used throughout the text. Section 3 presents the algorithm to compute inferences in credal sum-product networks. Section 4 establishes the conditions under which our algorithm computes exactly the lower and upper expectations in a credal sum-product network. Section 5 presents the main results on the correctness and complexity of the algorithm. Section 6 contains some experimental analyses showing the degradation of the values obtained by the algorithm when the complexity of the model structure increases. Finally, Section 7 presents our conclusions and future work.

2. Preliminaries

In this section we present the necessary background material to understand our algorithm. We first establish some definitions that will help us on imposing certain structural constraints to the graphical structure of the underlying sum-product network. Next, we present the class of functions our algorithm is able to deal with. Then, we formally introduce the concept of a sum-product network as well as the set of assumptions we use throughout the subsequent sections. Finally, we define a credal sum-product network and present the types of inferences with which we work.

2.1. The structure of SPNs: single-rooted acyclic directed graphs

The structure of a sum-product network is a single-rooted acyclic directed graph (SDAG). A directed graph is defined by a finite set $\mathcal{N}$ of nodes and a set $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ of ordered node pairs, called edges. For example, if $N, C \in \mathcal{N}$ and $(N, C) \in \mathcal{E}$, then we have a directed edge between N and C, which we write compactly as $N \to C$. A directed graph is acyclic when there are no directed cycles—paths of directed edges from any node to itself. (It is allowed to have cycles if we ignore direction of edges.) The graphs we consider have only one root—a node without incoming edges—, denoted by R. We consider that nodes can only belong to a single graph, which means we can generally leave $\mathcal{N}$ and $\mathcal{E}$ implicit in the notation.

For any node N of an SDAG, we can define some important node subsets:

- *Children* $\operatorname{ch} N := \{C \in \mathcal{N} : N \to C\}$ are those nodes to which there is a directed edge from N,
- *Descendants* $\operatorname{de} N := \operatorname{ch} N \cup \bigcup_{C \in \operatorname{ch} N} \operatorname{de} C$ are those nodes to which there is a sequence of directed edges from N,
- *Leaves* $\operatorname{lv} N := \{L \in \operatorname{de} N : \operatorname{ch} L = \emptyset\}$ are descendants of N that do not have children themselves.

Each node N of an SDAG determines a sub-SDAG $N{\downarrow}$ rooted in N with nodes $\{N\} \cup \operatorname{de} N$. So the whole SDAG can be written as $R{\downarrow}$.

There are some further concepts related to SDAGs that are needed further on:

Definition 1 *(Overlapping and disjoint nodes).* Consider a non-leaf node N of an SDAG and two of its children $C_1, C_2 \in \operatorname{ch} N$. Then we say that these children are

- either *overlapping* if and only if $\operatorname{lv} C_1 \cap \operatorname{lv} C_2 \neq \emptyset$,
- or *disjoint* if and only if $\operatorname{lv} C_1 \cap \operatorname{lv} C_2 = \emptyset$.

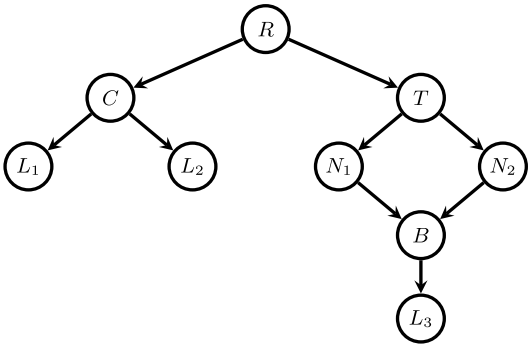


Fig. 1. An illustrative SDAG.

Definition 2 *(Cycle).* Consider two nodes T and B of an SDAG such that $B \in \mathrm{de}\, T$. We say there is a cycle between nodes T and B, if there are two distinct directed paths from T to B and these paths only have T and B as common nodes. We call T the top node of the cycle and B its bottom node.

Fig. 1 shows an SDAG that illustrates the various concepts introduced above. Its root is R and leaves are L_1, L_2, and L_3. The children of R are C and T. The descendants of N_1 are B and L_3. The nodes C and B are disjoint, but N_1 and N_2 are overlapping, as $\mathrm{lv}\, N_1 \cap \mathrm{lv}\, N_2 = \{L_3\}$. The nodes T and B respectively form the top and bottom of a cycle.

2.2. The set of random variables described by SPNs

We consider a finite collection $\boldsymbol{X}$ of random variables. To denote that a random variable X belongs to the collection, we write $X \in \boldsymbol{X}$. The set of possible realizations x for a variable X is denoted $\mathrm{val}\, X$. Similarly, $\mathrm{val}\, \boldsymbol{X}$ denotes the set of possible realizations $\boldsymbol{x}$ for all these variables, that is, $\mathrm{val}\, \boldsymbol{X} = \bigtimes_{X \in \boldsymbol{X}} \mathrm{val}\, X$. We use the notation $\boldsymbol{Y} \subseteq \boldsymbol{X}$ to denote a subcollection of the variables in $\boldsymbol{X}$. This means that if $X \in \boldsymbol{Y}$ then $X \in \boldsymbol{X}$, but the converse is not necessarily true. (So collections of random variables and their realizations are in boldface, but single random variables and their realizations are not.)

Joint realizations $\boldsymbol{x} \in \mathrm{val}\, \boldsymbol{X}$ or $\boldsymbol{y} \in \mathrm{val}\, \boldsymbol{Y}$ can be projected onto a subspace. For example, if $\boldsymbol{Y} \subseteq \boldsymbol{X}$, then $\boldsymbol{x}|_{\boldsymbol{Y}} \in \mathrm{val}\, \boldsymbol{Y}$ and given a single variable $X \in \boldsymbol{Y}$, we use the same notation $\boldsymbol{y}|_X \in \mathrm{val}\, X$ to project $\boldsymbol{y}$ onto X.

With every node N in an SDAG, we will associate a collection of random variables, called its *scope*. These are indicated by subscripting with the node symbol, e.g., $\boldsymbol{X}_N$. Leaf nodes L will have a specified scope $\boldsymbol{X}_L$, which often consists of a single random variable. The scope of a non-leaf node N is the union of the child scopes: $\boldsymbol{X}_N = \bigcup_{C \in \mathrm{ch}\, N} \boldsymbol{X}_C$. The root scope is $\boldsymbol{X}_R = \boldsymbol{X}$, which means that every random variable is in the scope of at least one leaf, but possibly more. For collections that are node scopes, we can then express projection using the node symbol: $\boldsymbol{x}|_N := \boldsymbol{x}|_{\boldsymbol{X}_N}$.

As an example, consider again Fig. 1. Let $\boldsymbol{X}_{L_1} = \boldsymbol{X}_{L_2} := \{X\}$ and $\boldsymbol{X}_{L_3} := \{Y\}$. Then $\boldsymbol{X}_C = \{X\}$, $\boldsymbol{X}_{N_1} = \{Y\}$, and $\boldsymbol{X}_R = \{X, Y\} = \boldsymbol{X}$.

2.3. Factorizing functions

The inferences on which we will focus can be expressed as functions that factorize over the variables $\boldsymbol{X}$:

Definition 3 *(General factorizing function).* A general factorizing function, $f : \mathrm{val}\, \boldsymbol{X} \to \mathbb{R}$, is defined for all $\boldsymbol{x} \in \mathrm{val}\, \boldsymbol{X}$ by

$$f(\boldsymbol{x}) = \prod_{X \in \boldsymbol{X}} f_X(\boldsymbol{x}|_X), \tag{1}$$

where, for all $X \in \boldsymbol{X}$, $f_X : \mathrm{val}\, X \to \mathbb{R}$ is some (appropriately measurable) real-valued univariate function.

We can also use, for any subcollection $\boldsymbol{Y} \subseteq \boldsymbol{X}$, the partial-product function $f_{\boldsymbol{Y}}$ defined for all $\boldsymbol{y} \in \mathrm{val}\, \boldsymbol{Y}$ by $f_{\boldsymbol{Y}}(\boldsymbol{y}) = \prod_{X \in \boldsymbol{Y}} f_X(\boldsymbol{y}|_X)$. In case the subcollection is the scope of a node N, i.e., $\boldsymbol{Y} = \boldsymbol{X}_N$, we can use the shorthand $f_N := f_{\boldsymbol{X}_N}$.

In this paper, we also consider the following two specific types of factorizing functions:

Definition 4 *(Non-negative factorizing function).* A non-negative factorizing function, $f : \mathrm{val}\, \boldsymbol{X} \to \mathbb{R}_{\geq 0}$, is a general factorizing function with the restriction that every factor, f_X, is non-negative, that is, for all $X \in \boldsymbol{X}$, we have that $f_X : \mathrm{val}\, X \to \mathbb{R}_{\geq 0}$.

Definition 5 *(General focused query function).* Consider a so-called query variable $X_q \in \boldsymbol{X}$. A general focused query function, $f : \mathrm{val}\, \boldsymbol{X} \to \mathbb{R}$, is defined for all $\boldsymbol{x} \in \mathrm{val}\, \boldsymbol{X}$ by

$$f(\boldsymbol{x}) = f_q(\boldsymbol{x}|_{X_q}) \prod_{X \in \boldsymbol{X} \setminus \{X_q\}} f_X(\boldsymbol{x}|_X) \tag{2}$$

where for all $X \in \boldsymbol{X}\backslash\{X_q\}$, we have $f_X : \operatorname{val} X \to \mathbb{R}_{\geq 0}$, that is, f_X is non-negative. On the other hand, the function f_q can take on any real value.

2.4. Sum-product networks

A sum-product network (SPN) is a computational structure representing a probabilistic model over a collection of variables $\boldsymbol{X}$ [16,17,19]. It consists of an SDAG, with a number of structural constraints, together with (numerical) parameters $\boldsymbol{w}$ (associated to sum nodes), distribution functions at leaf nodes, and recursive algorithms that are used to evaluate the model.

We wish to calculate for a given function f the expectation $\mathbb{E}_{R\downarrow,\boldsymbol{w}}(f)$ defined by the SPN with root R and weights $\boldsymbol{w}$. For this, an algorithm will be used that propagates values from leaves to root in the SDAG. With each node N we will therefore associate a value $N(f)$. The idea is to take $\mathbb{E}_{R\downarrow,\boldsymbol{w}}(f)$ equal to $R(f)$. Therefore, we must present the value-propagating algorithm and prove that it defines a proper expectation operator.

Let us now explain the characterization of the SPN in full. The SDAG used contains three distinct types of nodes: sum nodes, product nodes, and (leaf) distribution nodes. We discuss each of them below.

A *sum node* S in the SPN is associated with a function $\boldsymbol{w}_S$ on $\operatorname{ch} S$ that returns edge weights $w_{S\to C} := \boldsymbol{w}_S(C)$ for any $C \in \operatorname{ch} S$. These edge weights should be such that $\sum_{C\in\operatorname{ch} S} w_{S\to C} = 1$ and $w_{S\to C} \geq 0$ for all $C \in \operatorname{ch} S$. We also call $\boldsymbol{w}_S$ the *weight vector* associated with S. The numerical parameters $\boldsymbol{w}$ of the entire SPN can now simply be seen as the map $\boldsymbol{w} : S \mapsto \boldsymbol{w}_S$ that identifies these vectors for every sum node in the SPN. The value $S(f)$ of any sum node S is recursively defined in terms of the values of its children, as

$$S(f) := \sum_{C\in\operatorname{ch} S} w_{S\to C} C(f). \tag{3}$$

A *product node* P in the SPN is not endowed with any additional parameters. Its value, $P(f)$, is also recursively defined in terms of the values of its children, as

$$P(f) := \prod_{C\in\operatorname{ch} P} C(f). \tag{4}$$

Finally, a node D of the SPN is a *distribution node* if and only if it is a leaf in the SDAG, i.e., if it has no children. Any distribution node must have a non-empty scope $\boldsymbol{X}_D$. Moreover, D is associated with a probability distribution for $\boldsymbol{X}_D$. We use $\mathbb{E}_D$ to denote the expectation operator with respect to this distribution. In the sequel, we assume that all variables in $\boldsymbol{X}_D$ are pairwise independent under this distribution (note this is a local constraint within that node, and it is used to ensure that we can efficiently handle factorizing functions when computing the expectation at that leaf node).

Because distribution nodes are always leaves of the SDAG, they are also the terminal points of the recursive definition of the node values. For this reason, they are the point where we control the specific inference that the model will output. In other words, we will here need to make a choice depending on the argument f of these value functions. In this work, we focus on factorizing functions, f, shown in Section 2.3, and for any distribution node D in the SPN define its value as

$$D(f) := \mathbb{E}_D(f_D), \tag{5}$$

which actually does not depend on $\boldsymbol{w}$. We assume in the sequel that we can *efficiently evaluate* the expectations on the right-hand side of this definition.

To complete the characterization of SPNs, we need the following two structural assumptions to ensure we obtain a proper probabilistic model [16]; for any sum node S and any product node P in the SPN it holds that

A1: $\boldsymbol{X}_{C_1} = \boldsymbol{X}_{C_2}$ for all $C_1, C_2 \in \operatorname{ch} S$; *(smoothness)*

A2: $\boldsymbol{X}_{C_1} \cap \boldsymbol{X}_{C_2} = \emptyset$ for all $C_1, C_2 \in \operatorname{ch} P$. *(decomposability)*

A sum-product network that meets Assumptions (A1) and (A2) is called a *valid* SPN. Moreover, we make the following additional assumptions:

A3: the root R of the SDAG is a sum node;
A4: every product node P has at least two children;
A5: every distribution node D represents a univariate distribution, i.e. $|\boldsymbol{X}_D| = 1$.

None of these assumptions lose generality. Without changing what the model encodes, we can always add a dummy sum node to be the root if it is not a sum node yet, we can discard any product node with a single child, and we can split any non-univariate leaf into univariate ones using an additional product node.

As consequences of Assumptions (A1) and (A2) we have the following lemmas.

Lemma 1. *Consider a valid SPN and let P be one of its product nodes, then any pair of children $C_1, C_2 \in \operatorname{ch} P$ are disjoint.*

Proof. The proof is by contradiction. Suppose that C_1 and C_2 are overlapping nodes. This implies that $\operatorname{lv} C_1 \cap \operatorname{lv} C_2 \neq \emptyset$ or, equivalently, that $\boldsymbol{X}_{C_1} \cap \boldsymbol{X}_{C_2} \neq \emptyset$, but this is a contradiction with decomposability Assumption (A2). □

Lemma 2. *Consider a valid SPN and let T and B be two nodes in the SDAG such that $B \in \mathrm{de}\, T$. Suppose there is a cycle between T and B, then T is a sum node.*

Proof. The proof is by contradiction. Suppose T is a product node. Since there are two different directed paths from T to B, T has at least two children C_1 and C_2 that are part of these paths and hence $\mathrm{lv}\, C_1 \cap \mathrm{lv}\, C_2 \neq \emptyset$, that is, C_1 and C_2 are overlapping. By Lemma 1 this is a contradiction. □

Then as mentioned above, for factorizing functions f as in Equation (1), the expectation $\mathbb{E}_{R\downarrow,\boldsymbol{w}}(f)$ defined by the SPN can be efficiently evaluated by the value of the root R of the SPN, as

$$\mathbb{E}_{R\downarrow,\boldsymbol{w}}(f) = R(f). \tag{6}$$

This is a trivial result under the aforementioned assumptions about the SPN, since the SPN represents a proper density/probability function for the variables in it [16].

2.5. Credal sum-product networks

There has recently been an interest in constructing *robust* SPNs using the theory of *imprecise probabilities* [1,21]. Such models are also called *credal* probabilistic circuits or *credal* SPNs (CSPNs) [11,12]. As with *precise* (that is, non-credal) SPNs, a CSPN can be characterized by an SDAG and a number of numerical parameters. The SDAG has all the same properties as discussed in Section 2.4; in particular, we will take this to satisfy Assumptions (A1)–(A5). Moreover, we still assume that all distribution nodes in the CSPN have fixed distribution functions given to us.

However, the models differ in their remaining numerical parameterization. Recall that in an SPN, we associate a weight vector $\boldsymbol{w}_S$ to any of its sum nodes S; these weight vectors jointly form the numerical parameter $\boldsymbol{w}$ of the model. Conversely, in a CSPN we associate a *set* $\mathcal{W}_S$ with every of its sum nodes S. The elements $\boldsymbol{w}_S \in \mathcal{W}_S$ obey the constraints of the weights vectors for precise SPNs; the quantities $w_{S\to C}$, with $C \in \mathrm{ch}\, S$, are non-negative and sum to one. Following Mauá et al. [12], in the sequel we assume that every such set $\mathcal{W}_S$ is non-empty, closed, and convex (for every $\boldsymbol{w}_S, \boldsymbol{w}'_S \in \mathcal{W}_S$ and every $\lambda \in [0,1]$, there is some $\boldsymbol{w}^{\lambda}_S \in \mathcal{W}_S$ such that $\boldsymbol{w}^{\lambda}_S(C) = \lambda w_{S\to C} + (1-\lambda)\boldsymbol{w}'_S(C)$ for all $C \in \mathrm{ch}\, S$). The numerical parameters of the CSPN are then captured by the map $\mathcal{W} : S \mapsto \mathcal{W}_S$ that identifies these sets of weight vectors for all of its sum nodes.

The set-valued numerical parameters $\mathcal{W}$ can be understood as capturing *model uncertainty* with respect to the specification of the model. Roughly, we might say that we include in these sets all weight vectors that we deem to be relevant in some sense. With this model, one is then interested in computing inferences that are somehow *robust* with respect to variation in these sets. This might be done as follows: for every sum node S in the CSPN, choose a single weight vector $\boldsymbol{w}_S \in \mathcal{W}_S$, and aggregate all these vectors in the map $\boldsymbol{w} : S \mapsto \boldsymbol{w}_S$. For simplicity, let us write $\boldsymbol{w} \in \mathcal{W}$ for any map constructed in this manner. Then, as in Section 2.4, the given SDAG together with the weight $\boldsymbol{w}$ characterize an SPN, which in turn induces a probabilistic model with expectation operator $\mathbb{E}_{R\downarrow,\boldsymbol{w}}$. By considering all the different ways to make the aforementioned selections $\boldsymbol{w} \in \mathcal{W}$, we essentially generate a *set* of precise SPNs; they all have the same graphical structure, but they differ in their numerical parameters $\boldsymbol{w}$.

An obvious way to capture robustness is to consider this set of precise SPNs as the model, and to define the corresponding *lower* and *upper expectations* as, respectively,

$$\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) := \inf_{\boldsymbol{w}\in\mathcal{W}} \mathbb{E}_{R\downarrow,\boldsymbol{w}}(f), \qquad \text{and} \qquad \overline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) := \sup_{\boldsymbol{w}\in\mathcal{W}} \mathbb{E}_{R\downarrow,\boldsymbol{w}}(f) \tag{7}$$

for every function f. In other words, these lower and upper expectations express tight lower and upper bounds over the expectations of *all* precise SPNs with the CSPN's SDAG and weight vector $\boldsymbol{w} \in \mathcal{W}$.

The above approach is taken by Mauá et al. [12] and Centen [3], and some results about the tractability of solving this problem have been known for some time. Namely, Mauá et al. [12] propose an algorithm to calculate the bounds in Equation (7) and show that for the particular case of *credal likelihood*, these bounds can be computed in polynomial time in the number of edges and without imposing any structural constraint on the SDAG. Even more, it is shown that, unless every sum and product node in the CSPN has only one parent (tree-like structure) then, computing the values in Equation (7) is generally an *NP-hard* problem. More recently, Centen [3] proposes a new algorithm (we describe this algorithm in Section 3) to obtain lower and upper expectations in Expression (7) and investigates under what type of functions f and graphical structures (SDAG) it is still possible to tractably compute such bounds.

3. Algorithm for inference in CSPNs

As mentioned in Section 2.5, a CSPN is a set of precise SPNs, each having the same SDAG but different numerical parameters $\boldsymbol{w} \in \mathcal{W}$. As a consequence, the computational approach for CSPNs turns out to be more involved than in the precise case. Namely, instead of one recursively computed value $N(f)$ for each node N, we use *two* values $\underline{N}(f)$ and $\overline{N}(f)$ to compute inferences for CSPNs. These are, respectively, lower- and upper bounds on values for all of the nodes in the CSPN. Their definition is again recursive and depends on the type of node; let us consider each of them in turn.

First, fix a factorizing function f as in Equation (1), and any distribution node D of the CSPN. Recall from Section 2.5 that the distribution corresponding to D is given. Hence in this case we simply define

$$\underline{D}(f) := \overline{D}(f) := \mathbb{E}_D(f_D). \tag{8}$$

Similar to the precise case, we assume that the expectation in Equation (8) can be *efficiently evaluated*. Next, let us consider a sum node S of the CSPN. Then by construction, this has an associated non-empty, closed, and convex set $\mathcal{W}_S$ of weight vectors. We define the values for this node in terms of the values of its children, as

$$\underline{S}(f) := \min_{\boldsymbol{w}_S \in \mathcal{W}_S} \sum_{C \in \mathrm{ch}\, S} w_{S\to C}\, \underline{C}(f) \tag{9}$$

and

$$\overline{S}(f) := \max_{\boldsymbol{w}_S \in \mathcal{W}_S} \sum_{C \in \mathrm{ch}\, S} w_{S\to C}\, \overline{C}(f). \tag{10}$$

By assumption, the set $\mathcal{W}_S$ is closed and convex; hence, if values of children are known, the optimization problems in Equations (9) and (10) are linear programs whose values can be computed efficiently (in the worst case using a general linear programming solver, but even more efficiently if the sets have further properties, for example linear-vacuous sets admit a solution in time $O(n + logn)$, where n is the number of children, 2-monotone sets in time $O(n \cdot logk)$, where k is the number of values in child nodes, etc).

Finally, let us consider a product node P of the CSPN. Let $k = |\mathrm{ch}\, P|$ be the number of children of P, and let us enumerate these children in an arbitrary order as $\mathrm{ch}\, P = \{C_1, \ldots, C_k\}$. For all $i = 1, \ldots, k$, we now define auxiliary values $\underline{B}_i, \overline{B}_i$, in terms of the values associated with children $1, \ldots, i$, as follows. First, we simply define $\underline{B}_1 := \underline{C_1}(f)$ and $\overline{B}_1 := \overline{C_1}(f)$. Next, for all $i = 2, \ldots, k$, consider the set $\mathcal{B}_i$ containing all values obtained by pairwise-multiplying $\underline{B}_{i-1}, \overline{B}_{i-1}$ with $\underline{C_i}(f), \overline{C_i}(f)$, i.e. let

$$\mathcal{B}_i := \left\{ \underline{B}_{i-1}\underline{C_i}(f),\ \underline{B}_{i-1}\overline{C_i}(f),\ \overline{B}_{i-1}\underline{C_i}(f),\ \overline{B}_{i-1}\overline{C_i}(f) \right\}.$$

We then define the auxiliary values $\underline{B}_i := \min \mathcal{B}_i$ and $\overline{B}_i := \max \mathcal{B}_i$. Finally, we define the values for node P as

$$\underline{P}(f) := \underline{B}_k \quad \text{and} \quad \overline{P}(f) := \overline{B}_k. \tag{11}$$

Equation (11) generalizes the algorithm proposed by Mauá et al. [12] for product nodes. Furthermore, from the definition of the algorithm it can be easily seen that its time complexity is $O(k)$.

4. Intermediate results

In this section, we demonstrate the conditions under which the algorithm, as introduced in Section 3, enables the exact computation of both lower and upper expectations. To achieve this, we carefully examine the node's characteristics, along with the type of factorizing function utilized.

Lemma 3 *(Case distribution nodes). Let D be a distribution node from a CSPN and let f be any general factorizing function as in Definition 3. Then*

$$\underline{D}(f) = \overline{D}(f) = \mathbb{E}_D(f). \tag{12}$$

Proof. This follows from definition in Equation (8) and assumption of precise leaves. □

Lemma 4 *(Case product nodes). Let P be a product node from a CSPN and let f be a general factorizing function as in Definition 3. Assume that for every descendant node $N \in \mathrm{de}\, P$ we have computed its lower and upper expectation according to Section 3 and that the resulting values are equal to their theoretical counterpart. Then, if we use the computational rule given by Equation (11) we have*

$$\underline{P}(f) = \underline{\mathbb{E}}_{P\downarrow,\mathcal{W}}(f) \qquad \text{and} \qquad \overline{P}(f) = \overline{\mathbb{E}}_{P\downarrow,\mathcal{W}}(f). \tag{13}$$

Proof. The proof can be done by induction. Let us first illustrate the base case when P has only two children C_1 and C_2. By assumption we have $\underline{B}_1 := \underline{C_1}(f) = \underline{\mathbb{E}}_{C_1\downarrow,\mathcal{W}}(f_{C_1})$, $\overline{B}_1 := \overline{C_1}(f) = \overline{\mathbb{E}}_{C_1\downarrow,\mathcal{W}}(f_{C_1})$ and similarly for C_2, $\underline{C_2}(f) = \underline{\mathbb{E}}_{C_2\downarrow,\mathcal{W}}(f_{C_2})$ and $\overline{C_2}(f) = \overline{\mathbb{E}}_{C_2\downarrow,\mathcal{W}}(f_{C_2})$. In the case of an SPN, the value of P is given by Equation (4). Now, in the case of a CSPN each node has associated a lower and upper value, therefore we need to find the right combination of these quantities that minimizes (lower expectations) or maximizes (upper expectations) the product F_1F_2, where $F_1 \in \{\underline{B}_1, \overline{B}_1\}$ and $F_2 \in \{\underline{C_2}(f), \overline{C_2}(f)\}$; this can be accomplished using interval arithmetic [8], namely by multiplying the intervals $\left[\underline{B}_1, \overline{B}_1\right]$ and $\left[\underline{C_2}(f), \overline{C_2}(f)\right]$.

$$[\underline{B}_2, \overline{B}_2] := \left[\underline{B}_1, \overline{B}_1\right] \times \left[\underline{C_2}(f), \overline{C_2}(f)\right] = \left[\min \mathcal{B}_2, \max \mathcal{B}_2\right], \tag{14}$$

where

$$\mathcal{B}_2 := \left\{ \underline{B}_1\underline{C_2}(f),\ \underline{B}_1\overline{C_2}(f),\ \overline{B}_1\underline{C_2}(f),\ \overline{B}_1\overline{C_2}(f) \right\}, \tag{15}$$

which is exactly the algorithm for the case of two children. Suppose now that the algorithm works for the first $k-1$ children of a product node P. Again, by assumption, for the $k-$th children, C_k, we have $\underline{C_k}(f) = \underline{\mathbb{E}}_{C_k\downarrow,\mathcal{W}}(f_{C_k})$ and $\overline{C_k}(f) = \overline{\mathbb{E}}_{C_k\downarrow,\mathcal{W}}(f_{C_k})$. In this case, we need to minimize (lower expectations) or maximize (upper expectations) the product F_1F_2 where $F_1 \in \{\underline{B}_{k-1}, \overline{B}_{k-1}\}$ and $F_2 \in \{\underline{C_k}(f), \overline{C_k}(f)\}$, using interval arithmetic we obtain

$$[\underline{B}_k, \overline{B}_k] := \left[\underline{B}_{k-1}, \overline{B}_{k-1}\right] \times \left[\underline{C_k}(f), \overline{C_k}(f)\right] = \left[\min \mathcal{B}_k, \max \mathcal{B}_k\right], \tag{16}$$

where

$$\mathcal{B}_k := \left\{\underline{B}_{k-1}\underline{C_k}(f),\ \underline{B}_{k-1}\overline{C_k}(f),\ \overline{B}_{k-1}\underline{C_k}(f),\ \overline{B}_{k-1}\overline{C_k}(f)\right\}. \quad \square \tag{17}$$

As consequence of Lemma 4 we have the following corollary.

Corollary 1. *Assume the conditions of Lemma 4 hold but f is a non-negative factorizing function as in Definition 4. Then*

$$\underline{P}(f) = \prod_{C\in \mathrm{ch}\, P} \underline{C}(f) = \underline{\mathbb{E}}_{P\downarrow,\mathcal{W}}(f) \tag{18}$$

and

$$\overline{P}(f) = \prod_{C\in \mathrm{ch}\, P} \overline{C}(f) = \overline{\mathbb{E}}_{P\downarrow,\mathcal{W}}(f) \tag{19}$$

in other words, to compute $\underline{P}(f)$, Equation (11) only requires lower expectations from every child, similarly, to compute $\overline{P}(f)$ we only require the upper expectations.

Proof. This follows by noticing that in the set

$$\mathcal{B}_i := \left\{\underline{B}_{i-1}\underline{C_i}(f),\ \underline{B}_{i-1}\overline{C_i}(f),\ \overline{B}_{i-1}\underline{C_i}(f),\ \overline{B}_{i-1}\overline{C_i}(f)\right\} \tag{20}$$

the terms $\underline{B}_{i-1}\overline{C_i}(f)$ and $\overline{B}_{i-1}\underline{C_i}(f)$ appear to consider the cases where the sign of $\underline{B}_{i-1}$ is different from the sign of $\overline{C_i}(f)$. Since f is non-negative, these terms never achieve the minimum (or maximum) and thus are ignored. $\square$

Lemma 5 *(Sum nodes with disjoint children and general functions). Let S be a sum node from a CSPN and let f be a general factorizing function as in Definition 3. Assume that for every descendant node $N \in \mathrm{de}\, S$ we have computed its lower and upper expectation according to Section 3 and that the resulting values are equal to their theoretical counterpart. Further, assume that every pair of nodes in $\mathrm{ch}\, S$ is disjoint. Then, using Equations (9) and (10), we have*

$$\underline{S}(f) = \underline{\mathbb{E}}_{S\downarrow,\mathcal{W}}(f) \qquad \text{and} \qquad \overline{S}(f) = \overline{\mathbb{E}}_{S\downarrow,\mathcal{W}}(f). \tag{21}$$

Proof. We only prove the case for the lower bound since the same reasoning applies for upper bound. Let $\boldsymbol{Y} = \boldsymbol{X}_S$ be the scope of the sum node S. By Assumption (A1), $\boldsymbol{Y}$ is also the scope of every child node $C \in \mathrm{ch}\, S$. To simplify notation, in what follows we will just write f without mentioning its factors that involve $\boldsymbol{Y}$. Using these notational conventions we can write the lower expectation as

$$\underline{\mathbb{E}}_{S\downarrow,\mathcal{W}}(f) = \min_{\boldsymbol{w}\in\mathcal{W}} \mathbb{E}_{S\downarrow,\boldsymbol{w}}(f) \tag{22}$$

$$= \min_{\boldsymbol{w}\in\mathcal{W}} \int_{\mathcal{Y}} \sum_{C\in \mathrm{ch}\, S} f(\boldsymbol{y}) w_{S\to C} C_{\boldsymbol{w}}(\boldsymbol{y}) \mathrm{d}\boldsymbol{y}, \tag{23}$$

where in Equation (23) we use $C_{\boldsymbol{w}}(\boldsymbol{Y})$ to denote the distribution represented by the child node C. As implied by the notation, this distribution can depend on the weights from its descendants. Moreover the integral is a Lebesgue integral, thus it can represent a countable sum (when $\boldsymbol{Y}$ contains discrete random variables) or a Riemann integral (when $\boldsymbol{Y}$ contains absolutely continuous random variables). Using the linearity of expectation we obtain

$$\min_{\boldsymbol{w}\in\mathcal{W}} \int_{\mathcal{Y}} \sum_{C\in \mathrm{ch}\, S} f(\boldsymbol{y}) w_{S\to C} C_{\boldsymbol{w}}(\boldsymbol{y}) \mathrm{d}\boldsymbol{y} \tag{24}$$

$$= \min_{\boldsymbol{w}\in\mathcal{W}} \sum_{C\in \mathrm{ch}\, S} w_{S\to C} \int_{\mathcal{Y}} f(\boldsymbol{y}) C_{\boldsymbol{w}}(\boldsymbol{y}) \mathrm{d}\boldsymbol{y} \tag{25}$$

$$= \min_{\boldsymbol{w}\in\mathcal{W}} \sum_{C\in \mathrm{ch}\, S} w_{S\to C} \mathbb{E}_{C\downarrow,\boldsymbol{w}}(f) \tag{26}$$

$$= \min_{\boldsymbol{w}_S} \sum_{C\in \mathrm{ch}\, S} w_{S\to C} \min_{\boldsymbol{w}\in\mathcal{W}} \mathbb{E}_{C\downarrow,\boldsymbol{w}}(f) \tag{27}$$

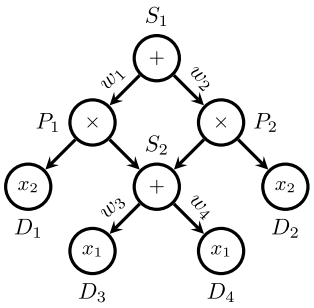


Fig. 2. CSPN with cycles where the algorithm from Section 3 fails.

$$= \min_{\boldsymbol{w}_S} \sum_{C \in \operatorname{ch} S} w_{S \to C} \underline{C}(f) \tag{28}$$

where in Equation (27) we use the hypothesis that children nodes are disjoint, in other words, none of the descendant sum nodes share weights and thus we can "distribute" the min operation. Equation (28) follows from the hypothesis that the algorithm computes the lower expectation for every child node. □

Before continuing with our next result, we show an example where the algorithm from Section 3 fails in obtaining the exact lower and upper expectations.

Example 1. Consider the CSPN whose structure is shown in Fig. 2. Let us assume that the variables X_1 and X_2 are binary variables. For both sum nodes, we use ϵ-contaminated credal sets, that is, for $\epsilon \in [0,1]$; we define

$$\begin{aligned} \mathcal{W}_{S_1} &:= \left\{(1-\epsilon)(w_1,w_2) + \epsilon(v_1,v_2) : v_1 + v_2 = 1, v_i \geq 0, i = 1,2\right\}, \\ \mathcal{W}_{S_2} &:= \left\{(1-\epsilon)(w_3,w_4) + \epsilon(v_3,v_4) : v_3 + v_4 = 1, v_i \geq 0, i = 3,4\right\}. \end{aligned} \tag{29}$$

For this example, we set $w_i = 1/2$ for $i = 1,2,3,4$ and $\epsilon = 0.1$. The distribution nodes are defined as follows.

$$D_1(X_2) = \begin{cases} 1 & \text{if } X_2 = 1, \\ 0 & \text{otherwise;} \end{cases} \qquad D_2(X_2) = \begin{cases} 1 & \text{if } X_2 = 0, \\ 0 & \text{otherwise;} \end{cases}$$

$$D_3(X_1) = \begin{cases} 1 & \text{if } X_1 = 1; \\ 0 & \text{otherwise,} \end{cases} \qquad D_4(X_1) = \begin{cases} 1 & \text{if } X_1 = 0; \\ 0 & \text{otherwise.} \end{cases}$$

Lastly, consider the following factorizing function $f(X_1, X_2) := f_1(X_1) f_2(X_2)$ where

$$f_1(X_1) = \begin{cases} 1 & \text{if } X_1 = 1; \\ 2 & \text{if } X_1 = 0, \end{cases} \qquad f_2(X_2) = \begin{cases} -1 & \text{if } X_2 = 1; \\ 1 & \text{if } X_2 = 0. \end{cases}$$

Using Equation (5) we obtain the following values for the distribution nodes

$$D_1(f) = -1 \qquad D_2(f) = 1 \tag{30}$$

$$D_3(f) = 1 \qquad D_4(f) = 2. \tag{31}$$

From the credal sets in Equation (29) and by Equation (3) we can express the value of the sum node S_2 as

$$S_2(f) = \left[\frac{1}{2}(1-\epsilon) + \epsilon v_3\right] D_3(f) + \left[\frac{1}{2}(1-\epsilon) + \epsilon v_4\right] D_4(f) \tag{32}$$

$$= \frac{3}{2}(1-\epsilon) + \epsilon(v_3 + 2v_4). \tag{33}$$

Then, using Equation (4) the values of the product nodes P_1 and P_2 are given by

$$P_1(f) = D_1(f) S_2(f) = -\frac{3}{2}(1-\epsilon) - \epsilon(v_3 + 2v_4), \tag{34}$$

$$P_2(f) = D_2(f) S_2(f) = \frac{3}{2}(1-\epsilon) + \epsilon(v_3 + 2v_4) \tag{35}$$

$$= -P_1(f). \tag{36}$$

Finally, the value for the sum node S_1 is

$$S_1(f) = \left[\frac{1}{2}(1-\epsilon) + \epsilon v_1\right] P_1(f) + \left[\frac{1}{2}(1-\epsilon) + \epsilon v_2\right] P_2(f) \tag{37}$$

$$= \left[\frac{1}{2}(1-\epsilon) + \epsilon v_1\right] P_1(f) - \left[\frac{1}{2}(1-\epsilon) + \epsilon v_2\right] P_1(f) \tag{38}$$

$$= P_1(f)\left[\epsilon(v_1 - v_2)\right] \tag{39}$$

$$= \left[\frac{3}{2}(1-\epsilon) + \epsilon(v_3 + 2v_4)\right]\left[\epsilon(v_2 - v_1)\right]. \tag{40}$$

When $\epsilon = 0.1$, it is easily seen from Equation (40) that the lower value for S_1 is $\underline{S_1}(f) = -0.155$; this value is attained when $v_2 = v_3 = 0$ and $v_1 = v_4 = 1$. Similarly, the upper value is $\overline{S_1}(f) = 0.155$; attained when $v_1 = v_3 = 0$ and $v_2 = v_4 = 1$.

On the other hand, applying the algorithm from Section 3 we obtain

$$\underline{D_3}(f) = \overline{D_3}(f) = \mathbb{E}_{D_3}(f_1) = 1, \tag{41}$$

$$\underline{D_4}(f) = \overline{D_4}(f) = \mathbb{E}_{D_4}(f_1) = 2, \tag{42}$$

$$\underline{D_1}(f) = \overline{D_1}(f) = \mathbb{E}_{D_1}(f_2) = -1, \tag{43}$$

$$\underline{D_2}(f) = \overline{D_2}(f) = \mathbb{E}_{D_2}(f_2) = 1. \tag{44}$$

For S_2 we have

$$\underline{S_2}(f) = \min_{v_3,v_4}\Big(\left[(1-\epsilon)w_3 + \epsilon v_3\right]\underline{D_3}(f) + \left[(1-\epsilon)w_4 + \epsilon v_4\right]\underline{D_4}(f)\Big) \tag{45}$$

$$= (1-\epsilon)w_3 + 2(1-\epsilon)w_4 + \min_{v_3,v_4}\left(\epsilon v_3 + 2\epsilon v_4\right) \tag{46}$$

$$= (1-\epsilon)w_3 + 2(1-\epsilon)w_4 + \epsilon \tag{47}$$

$$= 1.45. \tag{48}$$

Similarly

$$\overline{S_2}(f) = (1-\epsilon)w_3 + 2(1-\epsilon)w_4 + \max_{v_3,v_4}\left(\epsilon v_3 + 2\epsilon v_4\right) \tag{49}$$

$$= (1-\epsilon)w_3 + 2(1-\epsilon)w_4 + 2\epsilon \tag{50}$$

$$= 1.55. \tag{51}$$

Therefore for P_1 we define

$$\mathcal{B}_2 := \{\underline{D_1}(f)\underline{S_2}(f), \underline{D_1}(f)\overline{S_2}(f), \overline{D_1}(f)\underline{S_2}(f), \overline{D_1}(f)\overline{S_2}(f)\} \tag{52}$$

and hence

$$\underline{P_1}(f) = \min \mathcal{B}_2 = \min\{-1.45, -1.55, -1.45, -1.55\} = -1.55, \tag{53}$$

$$\overline{P_1}(f) = \max \mathcal{B}_2 = \max\{-1.45, -1.55, -1.45, -1.55\} = -1.45. \tag{54}$$

Similarly, for P_2

$$\mathcal{B}_2 := \{\underline{D_2}(f)\underline{S_2}(f), \underline{D_2}(f)\overline{S_2}(f), \overline{D_2}(f)\underline{S_2}(f), \overline{D_2}(f)\overline{S_2}(f)\} \tag{55}$$

and hence

$$\underline{P_2}(f) = \min \mathcal{B}_2 = \min\{1.45, 1.55, 1.45, 1.55\} = 1.45, \tag{56}$$

$$\overline{P_2}(f) = \min \mathcal{B}_2 = \max\{1.45, 1.55, 1.45, 1.55\} = 1.55. \tag{57}$$

Finally, for S_1 we have

$$\underline{S_1}(f) = \min_{v_1,v_2}\Big(\left[(1-\epsilon)w_1 + \epsilon v_1\right]\underline{P_1}(f) + \left[(1-\epsilon)w_2 + \epsilon v_2\right]\underline{P_2}(f)\Big) \tag{58}$$

$$= (1-\epsilon)w_1\underline{P_1}(f) + (1-\epsilon)w_2\underline{P_2}(f) + \min_{v_1,v_2}\left(\epsilon v_1\underline{P_1}(f) + \epsilon v_2\underline{P_2}(f)\right) \tag{59}$$

$$= -1.55(1-\epsilon)w_1 + 1.45(1-\epsilon)w_2 + \min_{v_1,v_2}\left(-1.55\epsilon v_1 + 1.45\epsilon v_2\right) \tag{60}$$

$$= -1.55(1-\epsilon)w_1 + 1.45(1-\epsilon)w_2 - 1.55\epsilon \tag{61}$$

$$= -0.2, \tag{62}$$

$$\overline{S_1}(f) = \max_{v_1,v_2}\Big(\big[(1-\epsilon)w_1 + \epsilon v_1\big]\,\overline{P_1}(f) + \big[(1-\epsilon)w_2 + \epsilon v_2\big]\,\overline{P_2}(f)\Big) \tag{63}$$

$$= (1-\epsilon)w_1\,\overline{P_1}(f) + (1-\epsilon)w_2\,\overline{P_2}(f) + \max_{v_1,v_2}\big(\epsilon v_1\,\overline{P_1}(f) + \epsilon v_2\,\overline{P_2}(f)\big) \tag{64}$$

$$= -1.45(1-\epsilon)w_1 + 1.55(1-\epsilon)w_2 + \max_{v_1,v_2}\big(-1.45\epsilon v_1 + 1.55\epsilon v_2\big) \tag{65}$$

$$= -1.45(1-\epsilon)w_1 + 1.55(1-\epsilon)w_2 + 1.55\epsilon \tag{66}$$

$$= 0.2. \tag{67}$$

The observed discrepancy arises due to the presence of product nodes in a cycle and the greedy nature of the algorithm from Section 3 which only relies on the values of the child nodes. To compute the exact solution, we have to take a global view and account for potential sign changes resulting from product nodes involved in the cycle.

Notably, when f can take on negative values, one of the product nodes might "propagate up" a lower value, while the other product node propagates an upper value associated to a common child node.

Specifically, from Equations (53) and (54), we see that P_1 attains its minimum and maximum value through $\overline{S_2}(f)$ and $\underline{S_2}(f)$ respectively. Similarly, P_2 attains these values using $\underline{S_2}(f)$ and $\overline{S_2}(f)$ respectively. In other words, $\underline{P_1}(f)$ propagates up the value $\overline{S_2}(f)$ to S_1, whereas $\underline{P_2}(f)$ propagates $\underline{S_2}(f)$; the converse holds for $\overline{P_1}(f)$ and $\overline{P_2}(f)$.

Intuitively, this scenario implies that the top node S_1 receives "distinct messages" from its children P_1 and P_2, both carrying information from the node S_2 (one message containing $\underline{S_2}(f)$ and the other $\overline{S_2}(f)$).

This is a consequence of the inherent greedy nature of the proposed algorithm, which prevents us from attaining the true minimum in such cases. A comprehensive empirical analysis of this situation is available in Section 6.

As we can see from Example 1, when f can take on negative values and the graph of the CSPN contains cycles, our algorithm no longer computes the exact lower and upper expectations. Despite this, we can still compute the correct values if we restrict f to take on non-negative values as shown next.

Lemma 6 *(Sum nodes with overlapping children and non-negative functions). Let S be a sum node from a CSPN and let f be a non-negative factorizing function as in Definition 4. Assume that for every descendant $N \in \mathrm{de}\,S$ we have computed its lower and upper expectation according to Section 3 and that the resulting values coincide with their theoretical counterpart. Further, assume that there is at least a pair of overlapping nodes in $\mathrm{ch}\,S$. Then, using Equations (9) and (10), we have*

$$\underline{S}(f) = \underline{\mathbb{E}}_{S\downarrow,\mathcal{W}}(f) \qquad \textit{and} \qquad \overline{S}(f) = \overline{\mathbb{E}}_{S\downarrow,\mathcal{W}}(f) \tag{68}$$

Proof of Lemma 6. We do the proof only for the lower expectation since similar arguments can be used for the upper one.

The idea for the proof is showing that, when $f \geq 0$, the theoretical lower expectation is attained with an SPN for which every weight vector assigns all of its mass to the child with the lowest (precise) value, then we show that our algorithm does this in a greedy manner. We proceed as follows.

A CSPN can be interpreted as a set of SPNs. The weight vector of each sum node in one of these SPNs is obtained from the mapping $\mathcal{W}$. Hence, once we parameterized an SPN using the mapping $\mathcal{W}$, we can calculate its value using Equations (3) to (5). Furthermore, when $f \geq 0$, the value of each node in an SPN is non-negative. Thus the lower expectation $\underline{\mathbb{E}}_{S\downarrow,\mathcal{W}}(f)$ is attained with an SPN for which each of its weight vectors $\boldsymbol{w}_S$ assigns all of its mass to the child with the lowest (precise) value.

On the other hand, since there is no imprecision on the leaves, when $f \geq 0$, each leaf has a non-negative lower and upper value; consequently, the lower and upper value of each node in a CSPN is non-negative. From this it follows that Equation (9) is minimized when the weight vector assigns all of its mass to the children with the lowest lower value.

Finally, since there is at least a pair of overlapping nodes in $\mathrm{ch}\,S$, we need to make sure that product nodes appearing in this cycle propagate the same value from its children (see Example 1). Corollary 1 assures us that this is the case, and therefore we can conclude that $\underline{S}(f) = \underline{\mathbb{E}}_{S\downarrow,\mathcal{W}}(f)$. □

5. Main results

In this section we use the Lemmas from Section 4 to obtain our main results. These results extend the ones found in Mauá et al. [12] to a larger class of DAG structures for CSPNs and/or more complex functions. First, we present a set of cases under which our algorithm can compute unconditional lower and upper expectations. We argue that these results are the best one can hope for by proving NP-hardness of the more general case. Then, we establish the conditions under which we can compute lower and upper expectations conditioned over a subset of evidence variables. Finally, we show how to use these results in order to compute credal dominance between a pair of values of a random variable.

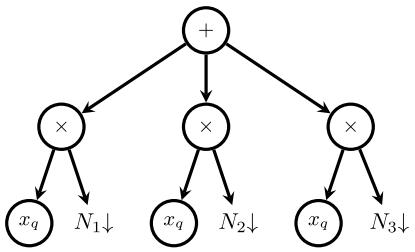


Fig. 3. CSPN with X_q-queried structure. In this figure, the subgraphs $N_1\downarrow$, $N_2\downarrow$ and $N_3\downarrow$ can have cycles but they do not share nodes nor any of its distribution nodes contain X_q.

5.1. Unconditional expectations

We start by establishing our main results by first focusing on unconditional expectations. Unlike Section 4 where we work at the node type level, the following results establish under what type of graph structures and factorizing functions our algorithm from Section 3 computes unconditional lower and upper expectations.

Theorem 1 *(Non-negative factorizing functions). Consider a CSPN with arbitrary graph structure and parameters $\mathcal{W}$. Fix any* non-negative *factorizing function f as in Definition 4. Let R be the root node. Then, using the algorithm from Section 3, we have*

$$\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) = \underline{R}(f) \qquad \text{and} \qquad \overline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) = \overline{R}(f). \tag{69}$$

Moreover, the complexity time for computing these bounds is $O(|\mathcal{N}|T)$ where $|\mathcal{N}|$ is the number of nodes in the CSPN and $O(T)$ is the time to solve the optimization in Equations (9) and (10) (recall that T in the worst case is a polynomial of the input size as the problem is a linear program).

Proof. This Theorem follows from Lemmas 3, 4 and Lemma 5 if every pair of sum nodes in the CSPN is disjoint or Lemma 6 if there are overlapping sum nodes. The complexity time follows by noticing that the worst-case scenario is under the case when the CSPN only contains sum nodes, in this case, for every sum node we have to solve the linear program in Equations (9) or (10). Solving each optimization takes time $O(T)$ and thus the complexity is $O(|\mathcal{N}|T)$. □

Corollary 2. *The algorithm established in Section 3 computes lower and upper likelihoods in polynomial time.*

Proof. This follows from Theorem 1 by taking

$$f(\boldsymbol{x}) = \mathbb{1}_{\{\boldsymbol{y}\}}(\boldsymbol{x}) = \prod_{X\in\boldsymbol{X}} \mathbb{1}_{\{\boldsymbol{y}|_X\}}(\boldsymbol{x}|_X), \tag{70}$$

that is, the product over each model variable X of indicator functions for some realization $\boldsymbol{y}|_X$ of that variable. □

Theorem 2 *(Tree structures). Consider a CSPN with parameters $\mathcal{W}$, and any* general *factorizing function f as in Definition 3. Suppose that the CSPN has a tree-like structure and let R be its root node. Then, using the algorithm from Section 3, we have*

$$\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) = \underline{R}(f) \qquad \text{and} \qquad \overline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) = \overline{R}(f). \tag{71}$$

Similar as in Theorem 1, the complexity time for computing these bounds is $O(|\mathcal{N}|T)$.

Proof. Since the CSPN has tree-like structure, then every pair of sum nodes is disjoint. The theorem follows from Lemmas 3, 4 and 5. The complexity argument is the same as in the proof of Theorem 1. □

We can also use the algorithm from Section 3 to compute lower/upper expectations for the following type of SDAG structures.

Definition 6 *(X_q-queried structure).* Consider a (C)SPN and let $X_q \in \boldsymbol{X}$ be a variable called the query variable. We say that the CSPN has an X_q-queried structure if every path from the root to any distribution node D with scope equal to X_q does not contain edges that are part of a cycle of the (C)SPN.

Fig. 3 shows an example of a graph with X_q-queried structure.

Theorem 3 *(Case: X_q-queried structures). Consider a CSPN with parameters $\mathcal{W}$. Let $X_q \in \boldsymbol{X}$ and assume that the CSPN has X_q-queried structure. Let f be a general focused query function as in Definition 5. Then, using the algorithm from Section 3, we have*

$$\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) = \underline{R}(f) \qquad \text{and} \qquad \overline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) = \overline{R}(f). \tag{72}$$

Similar as in Theorem 1, the complexity time for computing these bounds is $O(|\mathcal{N}|T)$.

Proof. We argue the case of lower expectations since the same argument can be used for the upper case.

According to Definition 5, f can be written as

$$f(\boldsymbol{x}) = f_q(\boldsymbol{x}|_{X_q}) \prod_{X \in \boldsymbol{X}\setminus\{X_q\}} f_X(\boldsymbol{x}|_X), \tag{73}$$

where f_X is non-negative for $X \in \boldsymbol{X}\setminus\{X_q\}$ whereas $f_q(\boldsymbol{x}|_{X_q})$ can take any real value. Take any path π from the root R to any distribution node, D. Suppose that the scope of D is X_q. We prove that any node on this path computes the correct lower expectation. Let N be any node in π and denote by C_D the children of N that we follow in order to reach D. Consider first the case when N is a product node. In this case, by decomposability, any other child $C \neq C_D$ of N does not contain X_q in its scope. For every $X \neq X_q$ we have $f_X \geq 0$, then it follows from Theorem 1 that

$$\underline{C}(f) = \underline{\mathbb{E}}_{C\downarrow,\mathcal{W}}(f). \tag{74}$$

In other words, for any child node $C \neq C_D$ of N, the algorithm from Section 3 computes the correct lower expectation for the subgraph rooted at C. Now, the X_q-queried structure of the CSPN, together with the decomposability of N ensures that the subgraph rooted at C_D has a tree-like structure, thus using Theorem 2 we have

$$\underline{C_D}(f) = \underline{\mathbb{E}}_{C_D\downarrow,\mathcal{W}}(f) \tag{75}$$

and therefore, by Lemma 4, the lower expectation computed up to the node N is correct.

Next, suppose that N is a sum node. By smoothness assumption, every child $C \in \operatorname{ch} N$ has X_q in its scope and hence the subgraph rooted at C has also X_q-queried structure. Let $\pi_{C,D'}$ be a path from C to a distribution node D' such that $X_q \in \boldsymbol{X}_{D'}$. If this path contains a product node, we can recursively apply the argument above to establish the correctness of the algorithm. On the other hand, if the path contains no product nodes, then from the definition of X_q-queried structured, it follows that the subgraph $C\downarrow$ has tree-like structure and thus Theorem 2 applies. Using the above arguments recursively, we conclude that $\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) = \underline{R}(f)$. The complexity argument is the same as in the proof of Theorem 1. □

5.2. Hardness for non-tree graphs

To a large extent, the previous results are the best for which one can hope. This is because computing general focused query functions in general CSPNs is NP-hard, as the following theorem shows.

Theorem 4 *(Hardness of expectations for non-trees). Consider a CSPN with parameters $\mathcal{W}$. Let $f(\boldsymbol{X})$ be a general focused query function as in Definition 5. Moreover, let X_q be the query variable, that is, f can be expressed as*

$$f(\boldsymbol{x}) = f_q(\boldsymbol{x}|_{X_q}) \prod_{X \in \boldsymbol{X}_e} f_X(\boldsymbol{x}|_X). \tag{76}$$

It then holds that deciding $\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) \leq \alpha$, for a given rational α, is NP-hard.

Proof. We only sketch a possible proof here since the result is derived from the very elaborate and arguably tedious proof in Theorem 2 of Mauá et al. [14]. The idea is that a tree-shaped Bayesian or credal network can be translated into an SPN (albeit with cycles) using a simple well-known construction [2]. After the translation of that network into an SPN, the same query of Theorem 2 in Mauá et al. [14] can be performed using the SPN. Since that theorem proves exactly the result that we seek here, such a translated version is enough to show hardness for non-tree SPN graphs. □

5.3. Conditional expectations

Now we establish the conditions and the way that we can compute conditional expectations using unconditional ones. In this case, we have a subset of variables which are interpreted as the evidence (i.e. these variables are observed and we use their values to condition on) and a single variable (the query variable) from which we want to make different types of inferences.

Theorem 5 *(Conditional expectations for queried structures). Consider a CSPN with an X_q-queried structure and parameters $\mathcal{W}$. Let $\boldsymbol{X}_e \subseteq \boldsymbol{X}\setminus\{X_q\}$ be the variables for which we have evidence. Let $f(\boldsymbol{X})$ be a general* focused query function *with query variable X_q. Specifically f is expressed as*

$$f(\boldsymbol{x}) = f_q(\boldsymbol{x}|_{X_q}) f_e(\boldsymbol{x}|_{X_e}), \tag{77}$$

where $f_e(\boldsymbol{x}|_{X_e})$ is a non-negative factorizing function of the variables in $\boldsymbol{X}_e$. Then $\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f|\boldsymbol{X}_e=\boldsymbol{x}_e)$ can be computed in polynomial time using the algorithm from Section 3. Concretely, $\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f|\boldsymbol{X}_e=\boldsymbol{x}_e)$ can be expressed as

$$\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f|\boldsymbol{X}_e=\boldsymbol{x}_e)=f_e(\boldsymbol{x}|_{X_e})\mu \tag{78}$$

where μ is such that $\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(\mathbb{1}_{\boldsymbol{x}_e}(\boldsymbol{X}_e)\left[f_q(X_q)-\mu\right])=0$.

Proof. Using positive homogeneity of the lower expectation [21] we have

$$\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f|\boldsymbol{X}_e=\boldsymbol{x}_e)=\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f_q(\boldsymbol{x}|_{X_q})f_e(\boldsymbol{x}|_{X_e})|\boldsymbol{X}_e=\boldsymbol{x}_e) \tag{79}$$

$$=f_e(\boldsymbol{x}|_{X_e})\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f_q|\boldsymbol{X}_e=\boldsymbol{x}_e). \tag{80}$$

By the generalized Bayes theorem [21], the term $\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f_q|\boldsymbol{X}_e=\boldsymbol{x}_e)$ is equal to the value of μ such that

$$\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(\mathbb{1}_{\boldsymbol{x}_e}(\boldsymbol{X}_e)\left[f_q(X_q)-\mu\right])=0, \tag{81}$$

where $\mathbb{1}_{\boldsymbol{x}_e}(\boldsymbol{X}_e)=\prod_{X\in\boldsymbol{X}_e}\mathbb{1}_{\boldsymbol{x}_e|_X}(X)$. The function $\mathbb{1}_{\boldsymbol{x}_e}(\boldsymbol{X}_e)\left[f_q(X_q)-\mu\right]$ in Equation (81) is a focused query function, so by Theorem 3 the left-side can be computed in polynomial time when using the algorithm presented in Section 3. Finally, we can use a root finding algorithm (e.g. binary search) to find the value of μ that solves Equation (81). □

Since every tree structure is a particular case of an X_q-queried structure, the following corollary follows immediately from Theorem 5.

Corollary 3 *(Conditional expectations for trees). Consider a CSPN with a tree-like structure and parameters $\mathcal{W}$. Let X_q be the query variable and let $\boldsymbol{X}_e\subseteq\boldsymbol{X}\backslash\{X_q\}$ be some variables for which we have evidence. Let $f(\boldsymbol{X})$ be a general focused query function with query variable X_q. Specifically, f is expressed as*

$$f(\boldsymbol{x})=f_q(\boldsymbol{x}|_{X_q})f_e(\boldsymbol{x}|_{X_e}), \tag{82}$$

where $f_e(\boldsymbol{x}|_{X_e})$ is a non-negative factorizing function of the variables in $\boldsymbol{X}_e$. Then $\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f|\boldsymbol{X}_e=\boldsymbol{x}_e)$ can be computed in polynomial time.

Our previous results can be nicely applied when dealing with a classification problem, here the explanatory variables are the evidence variables and the class variable becomes the query variable.

Under the framework of imprecise probabilities, instead of returning the most probable class as is done in classical probability, we identify a set of non-dominated classes according the criterion of credal dominance [1].

Definition 7 *(Credal dominance).* Let X_q be the class variable, $\boldsymbol{X}_e=\boldsymbol{x}_e$ a set of observed variables and $\mathcal{M}$ a set of probability measures. For c_1,c_2 in $\operatorname{val}X_q$, $c_1\neq c_2$; we say that c_1 credally dominates c_2 if for all $\mathbb{P}\in\mathcal{M}$ we have

$$\mathbb{P}(X_q=c_1|\boldsymbol{X}_e=\boldsymbol{x}_e)>\mathbb{P}(X_q=c_2|\boldsymbol{X}_e=\boldsymbol{x}_e) \tag{83}$$

Using expectations this is equivalent to

$$\mathbb{E}_{\mathbb{P}}\left[\mathbb{1}_{X_q=c_1}(X_q)-\mathbb{1}_{X_q=c_2}(X_q)|\boldsymbol{X}_e=\boldsymbol{x}_e\right]>0 \tag{84}$$

for all $\mathbb{P}\in\mathcal{M}$.

Corollary 4 *(Credal dominance in X_q-queried structures). Consider a CSPN. Let $X_q\in\boldsymbol{X}$ be a variable associated to a class label (the query variable) and let $\boldsymbol{X}_e\subset\boldsymbol{X}$ be a set of variables for which we have evidence ($X_q\notin\boldsymbol{X}_e$). If the CSPN has tree or X_q-queried structure, then credal dominance can be done in polynomial time.*

Proof. In this case the set of probability measures $\mathcal{M}$ in Definition 7 is given by the CSPN, therefore the expression

$$\mathbb{E}_{\mathbb{P}}\left[\mathbb{1}_{X_q=c_1}(X_q)-\mathbb{1}_{X_q=c_2}(X_q)|\boldsymbol{X}_e=\boldsymbol{x}_e\right]>0 \tag{85}$$

for all $\mathbb{P}\in\mathcal{M}$, is equivalent to having a positive lower bound

$$\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}\left[\mathbb{1}_{X_q=c_1}(X_q)-\mathbb{1}_{X_q=c_2}(X_q)|\boldsymbol{X}_e=\boldsymbol{x}_e\right]>0, \tag{86}$$

where R is the root node of the CSPN. The conditional expectation in inequality (86) can be computed using Theorem 5 with $f_q(\boldsymbol{x}|_{X_q})=\mathbb{1}_{X_q=c_1}(X_q)-\mathbb{1}_{X_q=c_2}(X_q)$ and $f_e(\boldsymbol{x}|_{X_e})=1$. □

6. The impact of cycles on inferences

As illustrated in Example 1, when the graph structure of the CSPN has cycles (as in Definition 2) and when the factorizing function f can take on negative values, the algorithm from Section 3 cannot be applied to obtain the value of the theoretical expectations. However, we can show that it computes a lower bound for $\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f)$ and an upper bound for $\overline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f)$, that is, an outer approximation.

We prove first the following lemma.

Lemma 7. *Consider a CSPN and let f be any factorizing function. Let N be any node in the CSPN and suppose that for any child node $C \in \mathrm{ch}\, N$ the following conjugacy relation holds*

$$\underline{C}(f) = -\overline{C}(-f). \tag{87}$$

Then the conjugacy relation also holds for N, that is,

$$\underline{N}(f) = -\overline{N}(-f). \tag{88}$$

Proof. In the case of distribution nodes, the claim follows immediately from Equation (8) and the linearity of (precise) expectations.

Next, let P be a product node of the CSPN with k children. Using Equation (11) we have

$$\begin{aligned}
\underline{P}(f) &= \min\{\underline{B}_{k-1}\underline{C}_k(f), \underline{B}_{k-1}\overline{C_k}(f), \overline{B}_{k-1}\underline{C_k}(f), \overline{B}_{k-1}\overline{C_k}(f)\} && (89)\\
&= \min\{-\underline{B}_{k-1}\overline{C_k}(-f), -\underline{B}_{k-1}\underline{C_k}(-f), -\overline{B}_{k-1}\overline{C_k}(-f), -\overline{B}_{k-1}\underline{C_k}(-f)\} && (90)\\
&= -\max\{\underline{B}_{k-1}\overline{C_k}(-f), \underline{B}_{k-1}\underline{C_k}(-f), \overline{B}_{k-1}\overline{C_k}(-f), \overline{B}_{k-1}\underline{C_k}(-f)\} && (91)\\
&= -\overline{P}(-f) && (92)
\end{aligned}$$

where in Equation (90) we use the assumption that conjugacy holds for the children.

Finally, let S be a sum node. According to Equation (9) we have

$$\begin{aligned}
\underline{S}(f) &= \min_{\boldsymbol{w}_S} \sum_{C\in\mathrm{ch}\,S} w_{S\to C}\underline{C}(f) && (93)\\
&= \min_{\boldsymbol{w}_S} -\sum_{C\in\mathrm{ch}\,S} w_{S\to C}\overline{C}(-f) && (94)\\
&= -\max_{\boldsymbol{w}_S} \sum_{C\in\mathrm{ch}\,S} w_{S\to C}\overline{C}(-f) && (95)\\
&= -\overline{S}(-f) \quad \square && (96)
\end{aligned}$$

The following proposition establishes that the algorithm from Section 3 computes an outer approximation of the true lower/upper expectations. The intuition of this result is that the algorithm essentially assumes a tree structure where the cycles are removed (unrolled), obtaining in this way a relaxed optimization problem, resulting in outer bounds.

Proposition 1. *Consider a CSPN with root node R. Let f be a general factorizing function as in Definition 3. Then we have*

$$\underline{R}(f) \le \underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) \qquad \text{and} \qquad \overline{R}(f) \ge \overline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f). \tag{97}$$

Proof. If the CSPN is a tree, then using Theorem 2 we have that the equality holds. Suppose now that the CSPN has at least one cycle. In this case, we can unroll every cycle in the CSPN (copying nodes and the corresponding parameters, that is, $\mathcal{W}$) as shown in Fig. 4. Let R' denote the root of the SDAG created by unrolling every cycle in the original SDAG with root R. Since the unrolled SDAG has a tree-like structure, by Theorem 2 we know that $\underline{R'}(f) = \underline{\mathbb{E}}_{R'\downarrow,\mathcal{W}}(f)$. Since we copy the parameters from the original CSPN, we also have

$$\underline{\mathbb{E}}_{R'\downarrow,\mathcal{W}}(f) = \underline{R'}(f) = \underline{R}(f). \tag{98}$$

On the other hand, since the unrolled SDAG is a proper super-model (relaxation) of the original SDAG (results would match those of the original only if all copied/unrolled cycles would take the same choices during inference), it follows that

$$\underline{\mathbb{E}}_{R'\downarrow,\mathcal{W}}(f) \le \underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f), \tag{99}$$

and thus $\underline{R}(f) \le \underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f)$.

To prove the other inequality we use the conjugacy relation from Lemma 7. We have shown that the inequality $\underline{R}(f) \le \underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f)$ holds for any general factorizing function f, in particular, it holds for $-f$; hence we have

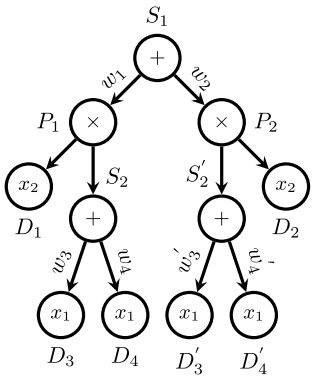


Fig. 4. Unrolling the cycle from Fig. 2. As we can notice, the structure below the bottom node S_2 in Fig. 2 gets duplicated, creating the subgraph rooted at node S_2'.

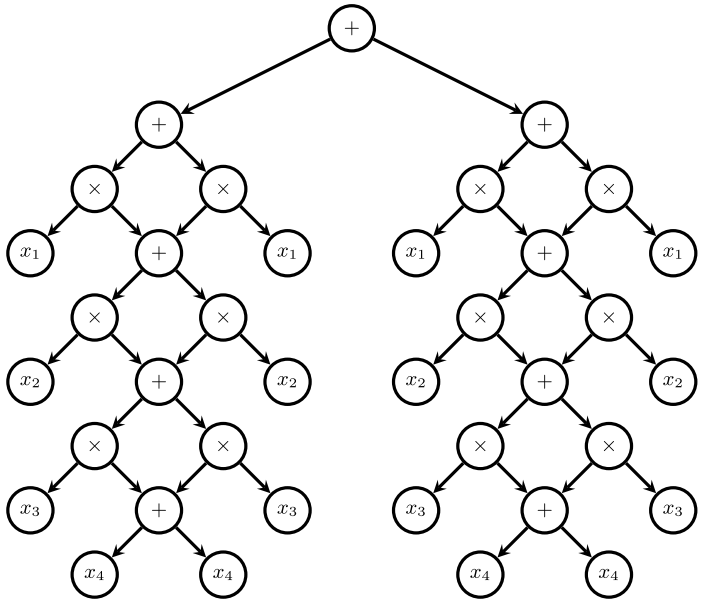


Fig. 5. CSPN with 6 cycles.

$$\underline{R}(-f) \leq \underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(-f) \Leftrightarrow -\underline{R}(-f) \geq -\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(-f) \tag{100}$$

$$\Leftrightarrow \overline{R}(f) \geq \overline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f), \tag{101}$$

where the last equivalence follows from conjugacy. □

Finally, the next proposition shows the relation between lower and upper errors; in particular, it suggests that the errors might not be the same for a given function. Yet, it is enough to study the quality of the errors on one side to get an idea of the quality of the errors on the other side because of their relation.

Proposition 2. *Consider a CSPN with root node R. Let f be a general factorizing function as in Definition 3. Define the lower and upper errors, respectively, as*

$$\underline{\delta}_f := \underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) - \underline{R}(f), \tag{102}$$

$$\overline{\delta}_f := \overline{R}(f) - \overline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f), \tag{103}$$

then we have $\underline{\delta}_f = \overline{\delta}_{-f}$.

Proof. The result follows from the conjugacy relation established in Lemma 7. We have

$$\underline{\delta}_f = \underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f) - \underline{R}(f) \tag{104}$$

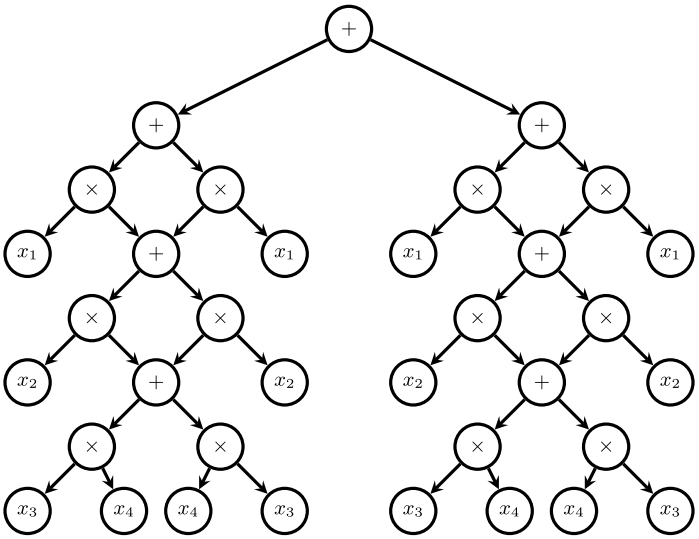


Fig. 6. CSPN with 4 cycles.

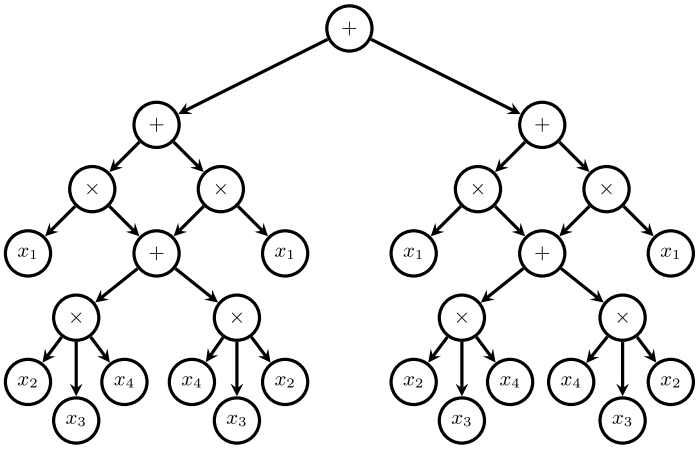


Fig. 7. CSPN with 2 cycles.

$$= -\left(\overline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(-f) - \overline{R}(-f)\right) \tag{105}$$

$$= -\left(-\overline{\delta}_{-f}\right) \tag{106}$$

$$= \overline{\delta}_{-f}. \quad \square \tag{107}$$

6.1. Empirical assessment of the impact of cycles

Empirically, we assess the effect that cycles have on the computed inferences when f takes on negative values. We do this as follows.

First, we start with the graph shown in Fig. 5. Iteratively, we remove cycles from each branch as seen in Figs. 6 and 7, using this procedure we end up having a total of 6 different graphs. For each of the resulting graphs, we sample random local parameters of an SPN and then apply ϵ-contamination to each sum node as shown in Example 1.

For every value of ϵ and every graph structure, we run 50 iterations. Each iteration has a different function f which can take on negative values. For each iteration, we obtain the exact lower expectation $\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f)$ as well as the approximation $\underline{R}(f)$; we use these values to compute the absolute and log-relative errors.

Fig. 8 reports our results. From this figure we see that when f can take on negative values, the error is an increasing function of the number of cycles in the graph, thus making the bound on $\underline{\mathbb{E}}_{R\downarrow,\mathcal{W}}(f)$ looser.

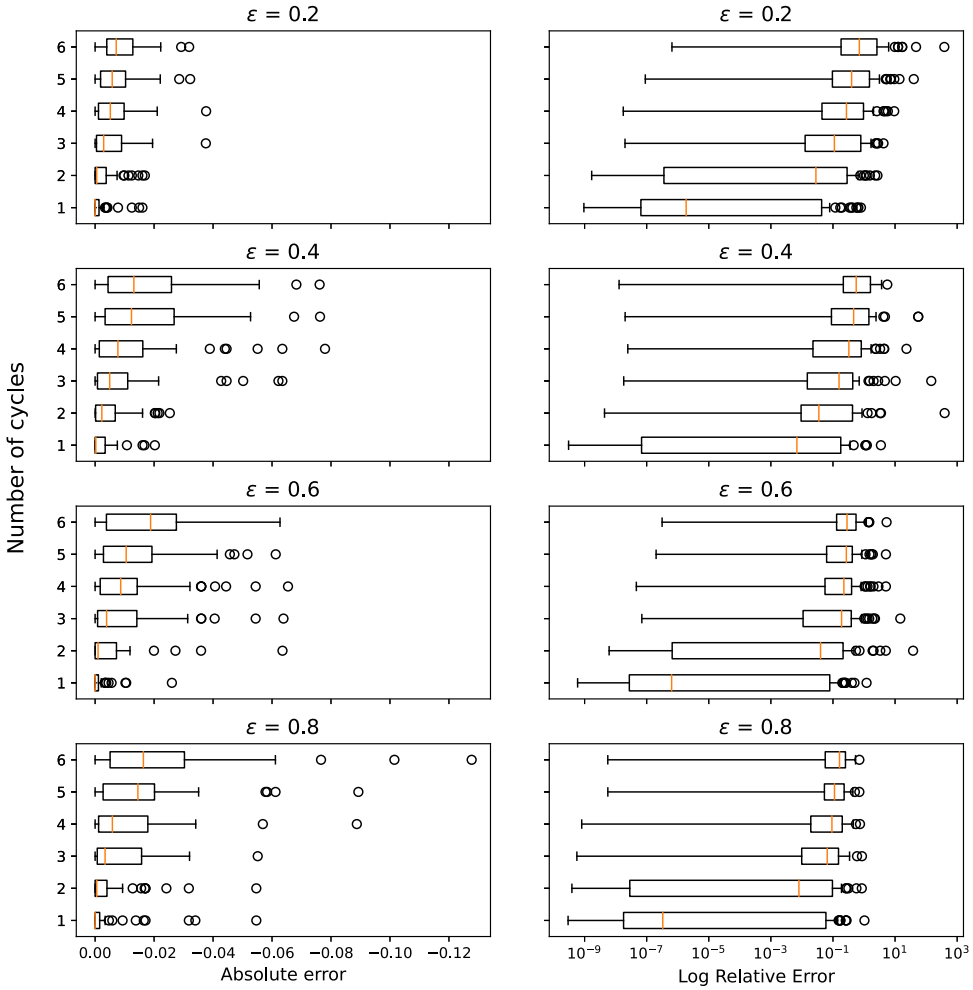


Fig. 8. Error as function of the number of cycles.

7. Conclusions

We extend previous results [12] to compute inferences using credal sum-product networks (CSPNs) by analyzing a broader class of graph structures as well as types of factorizing functions over which tractable algorithms are possible and we design such algorithms. We prove that our proposed algorithms have time complexity polynomial (often linear or almost linear) in the number of nodes. Furthermore, we prove that our algorithm computes tight bounds for lower and upper expectations and we perform an empirical analysis to understand under what situations these bounds can become arbitrarily loose (over graphs that do not satisfy the conditions of the algorithms). We also prove that one cannot solve inferences with CSPNs in unrestricted graphs in polynomial time (unless P = NP). There are multiple directions for future research. One can consider imprecision over the parameters of leaf distribution nodes. One can consider further special cases of graphs with cycles and/or different types of query functions that might admit polynomial-time algorithms under appropriate restrictions. One may attempt to smartly compute and cache computations to speed up calculations even further. More broadly and challenging, one could try to bring credal inferences to continuous mixtures [5]. There are plenty of ideas to grow this area still to be explored.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

The authors thank the support from the Eindhoven Artificial Intelligence Systems Institute and the Department of Mathematics and Computer Science of TU Eindhoven. Cassio de Campos thanks the support of EU European Defence Fund Project KOIOS (EDF-2021-DIGIT-R-FL-KOIOS) and Dutch NWO Perspectief 2021-2022 Project PersOn (P21-03).

References

[1] T. Augustin, F.P.A. Coolen, G. De Cooman, M.C.M. Troffaes (Eds.), Introduction to Imprecise Probabilities, Wiley Series in Probability and Statistics, 2014.
[2] C. Butz, J.S. Oliveira, R. Peharz, Sum-product network decompilation, in: M. Jaeger, T.D. Nielsen (Eds.), Proceedings of the 10th International Conference on Probabilistic Graphical Models, PMLR, 2020, pp. 53–64, https://proceedings.mlr.press/v138/butz20a.html.
[3] T. Centen, Beyond tree-shaped credal sum-product networks, Master's thesis, Technische Universiteit Eindhoven, 2021.
[4] F. Cerutti, L.M. Kaplan, A. Kimmig, M. Şensoy, Handling epistemic and aleatory uncertainties in probabilistic circuits, Mach. Learn. 111 (2022) 1259–1301, https://doi.org/10.1007/s10994-021-06086-4.
[5] A.H. Correia, G. Gala, E. Quaeghebeur, C. de Campos, R. Peharz, Continuous mixtures of tractable probabilistic models, Proc. AAAI Conf. Artif. Intell. 37 (2023) 7244–7252, https://doi.org/10.1609/aaai.v37i6.25883.
[6] C.P. De Campos, New complexity results for MAP in Bayesian networks, in: International Joint Conference on Artificial Intelligence (IJCAI), AAAI Press, 2011, pp. 2100–2106.
[7] C.P. De Campos, F.G. Cozman, The inferential complexity of Bayesian and credal networks, in: International Joint Conference on Artificial Intelligence (IJCAI), 2005, pp. 1313–1318.
[8] L. Jaulin, M. Kieffer, O. Didrit, Éric Walter, Applied Interval Analysis, 1 ed., Springer, London, 2001.
[9] P. Khosravi, Y. Choi, Y. Liang, A. Vergari, G. Van den Broeck, On tractable computation of expected predictions, in: NeurIPS, 2019, pp. 11167–11178.
[10] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, The MIT Press, Cambridge, Massachusetts, 2009.
[11] A. Levray, V. Belle, Learning credal sum product networks, in: Automated Knowledge Base Construction, 2020, https://openreview.net/forum?id=3-Tc21z1Ub.
[12] D.D. Mauá, D. Conaty, F.G. Cozman, K. Poppenhaeger, C.P. De Campos, Robustifying sum-product networks, Int. J. Approx. Reason. 101 (2018) 163–180, https://doi.org/10.1016/j.ijar.2018.07.003.
[13] D.D. Mauá, F.G. Cozman, D. Conaty, C.P. Campos, Credal sum-product networks, in: A. Antonucci, G. Corani, I. Couso, S. Destercke (Eds.), Proceedings of the Tenth International Symposium on Imprecise Probability: Theories and Applications, PMLR, 2017, pp. 205–216, https://proceedings.mlr.press/v62/mau%C3%A117a.html.
[14] D.D. Mauá, C.P. De Campos, A. Benavoli, A. Antonucci, Probabilistic inference in credal networks: new complexity results, J. Artif. Intell. Res. 50 (2014) 603–637, https://doi.org/10.1613/jair.4355.
[15] R. Peharz, R. Gens, F. Pernkopf, P. Domingos, On the latent variable interpretation in sum-product networks, IEEE Trans. Pattern Anal. Mach. Intell. 39 (2017) 2030–2044, https://doi.org/10.1109/TPAMI.2016.2618381.
[16] R. Peharz, S. Tschiatschek, F. Pernkopf, P. Domingos, On theoretical properties of sum-product networks, in: G. Lebanon, S.V.N. Vishwanathan (Eds.), Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, PMLR, San Diego, California, USA, 2015, pp. 744–752, https://proceedings.mlr.press/v38/peharz15.html.
[17] H. Poon, P. Domingos, Sum-product networks: a new deep architecture, in: 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), 2011, pp. 689–690.
[18] D. Roth, On the hardness of approximate reasoning, Artif. Intell. 82 (1996) 273–302, https://doi.org/10.1016/0004-3702(94)00092-1.
[19] R. Sánchez-Cauce, I. París, F.J. Díez, Sum-product networks: a survey, IEEE Trans. Pattern Anal. Mach. Intell. 44 (7) (2022) 3821–3839, https://doi.org/10.1109/TPAMI.2021.3061898.
[20] A. Vergari, Y. Choi, A. Liu, S. Teso, G. Van den Broeck, A compositional atlas of tractable circuit operations for probabilistic inference, in: A. Beygelzimer, Y. Dauphin, P. Liang, J.W. Vaughan (Eds.), Advances in Neural Information Processing Systems, 2021, https://openreview.net/forum?id=9SD2Rb3NiWu.
[21] P. Walley, Statistical Reasoning with Imprecise Probabilities, Chapman and Hall, 1991.
[22] H. Zhao, M. Melibari, P. Poupart, On the relationship between sum-product networks and Bayesian networks, in: F. Bach, D. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning, PMLR Lille, France, 2015, pp. 116–124, https://proceedings.mlr.press/v37/zhaoc15.html.