Beyond tree-shaped credal sum-product networks

Centen, Tijn

Award date:
2021

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Uncertainty in Artificial Intelligence Research Group

Beyond tree-shaped credal sum-product networks

MSc Thesis

Tijn Centen

Supervisors:
dr. ir. Erik Quaeghebeur
dr. habil. Cassio de Campos
dr. Thomas Krak

Assessment Committee:
dr. ir. Erik Quaeghebeur
dr. habil. Cassio de Campos
dr. Meng Fang

Eindhoven, July 2021

Abstract

Machine learning models applied to critical applications require robust results that can be relied upon. An example of such a critical application is classifying traffic signs in autonomous vehicles. In this example, a robust result means that the output class does not change for small alterations to the traffic sign image. These alterations can either be caused by random noise or by adversarial attacks. Hence, being able to perform a robustness analysis efficiently on these models is an important research topic.

Sum-product networks model a joint probability distribution as a computational graph. The structure of sum-product networks is a directed acyclic graph with additional constraints applied to either sum or product operations. Due to these constraints, exact tractable inference queries can be executed efficiently. A robustness analysis can be performed on sum-product networks, since these networks allow for imprecise inference queries by varying the weight vectors using credal sets.

Up till now, the correctness and efficiency of credal sum-product networks has only been proven for likelihood and conditional expectation queries on tree structures. We present a general imprecise expectation algorithm adapted to computing the imprecise expectation of general factorizing functions, which represent queries. In this thesis, we prove that this algorithm computes the imprecise expectations efficiently on certain directed acyclic graphs. We also prove that any query that complies with the general factorizing function requirements can be computed efficiently on tree structures. If the function factors are non-negative, this also holds for directed acyclic graph structures.

Furthermore, we present two experiments where the imprecise expectation algorithm is used to perform a classification task on a directed acyclic graph structure. The results show that the computed imprecise expectations can be used to perform a robustness analysis and increase the classification accuracy by filtering out unreliable results.

Keywords: Sum-product networks, Tractable probabilistic models, Imprecise inference, Credal classification, Robustness analysis

Preface

This master thesis is the result of my graduation project for the Computer Science and Engineering master's program at Eindhoven University of Technology. It is an internal project at the Uncertainty in Artificial Intelligence Research Group.

I would like to thank Erik Quaeghebeur, Cassio de Campos and Thomas Krak for their support and guidance during my master's project. I have learned a lot during the discussions of the project. I am also grateful to have had the opportunity to experience ISIPTA and to present our poster abstract. I also appreciate the valuable feedback that has been given, even though all communication occurred online due to the pandemic.

Furthermore, I would like to thank my father, mother and sister for their support over the past years.

Tijn Centen

Contents

List of Figures

Chapter 1

Introduction

Probabilistic circuits (PCs) are a type of machine learning models encompassing different types of circuits, which vary on the degree of expressiveness and tractability by applying different constraints on the circuit structure. A sum-product network (SPN) is one type of probabilistic circuit and can be seen as a computational graph that models a joint probability distribution. All nodes in this graph model a probability distribution, with leaf nodes modelling univariate distributions over continuous or discrete variables, like a Gaussian or Bernoulli distribution. Next, product nodes model a factorized distribution over its direct child distributions, increasing the number of variables. Expressiveness is added to the circuit by sum nodes, which model mixture distributions over their child node distributions by computing the weighted sum. The name of sum-product networks is characterized by the two operations performed in the nodes. They are also called "deep mixture models"[32].

Figure 1.1 shows a small example sum-product network over two variables X_0 and X_1. The SPN contains two univariate leaf distributions over variable X_1 which are combined as a mixture distribution in the sum node. The weights of the sum node are assumed to be non-negative and normalized. The resulting mixture distribution of the sum node can form more complex distributions which can increase the expressiveness, so that the real-world distribution can be better approximated. Next, the mixture distribution is combined with the distribution over variable X_0 in a product node, resulting in a multivariate distribution at the root node.

The structural constraints which form a sum-product network from a PC are *smoothness* and *decomposability*. A sum node is smooth, or complete, only when the variables of all child distributions are the same. An SPN is smooth if all sum nodes are smooth. A product node is decomposable, only when the variables of each child distribution are distinct with respect to the other child distributions. An SPN is decomposable if all product nodes are decomposable. These constraints together result in being able to execute exact and tractable inference queries of factorizing functions, like marginalization (MAR) and conditional (CON) queries. The factorizing function can be decomposed into smaller functions up until the leaf nodes, where the function is used to determine the value propagated by the leaf node.

Due to the structural constraints and the sum and product operations described above, the computational graph structure of a sum-product network is constrained to a directed acyclic graph (DAG). This means that the joint probability distribution can be modelled compactly, as node distributions can be reused in different nodes and be merged together by a sum node. The structure of an SPN can either be manually created or constructed using a structure learning algorithm.

SPNs as introduced above are precise models, as the queries compute a single value as the result from observed evidence over the variables. This means that the result of a query could drastically change when the trained weights are only altered slightly. Thus, determining whether a result is reliable or unreliable is important. Adding imprecision to probabilistic models allows for analyzing the robustness of specific tasks performed by the model. Imprecision in probabilistic models can be represented by the computation of lower and upper bounds of the result [41], and can for example be achieved by varying the model parameters. These lower and upper bounds can

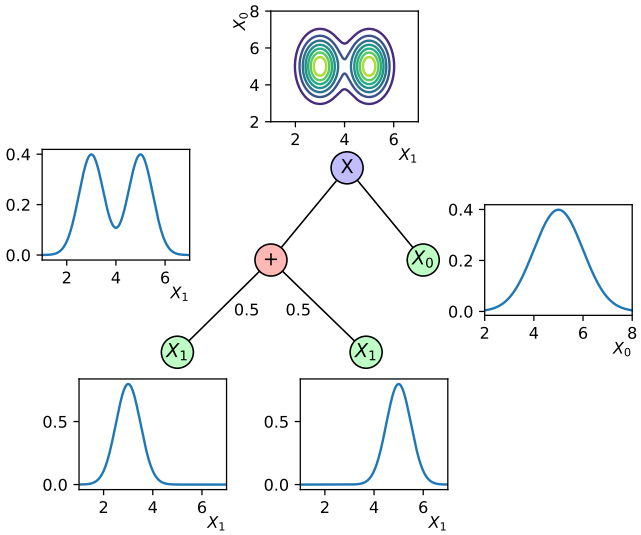


Figure 1.1: Illustration of an SPN structure with accompanying probability distributions over two variables.

be used to analyze the robustness, which is important in for example a classification task, where a model should be able to defer the classification choice to either a human or another model if the result is unreliable and the robustness is low.

A probabilistic model where imprecision is added is a credal network, as introduced by Cozman [8, 9], which extends from Bayesian networks. However, these credal networks do not achieve exact tractable inference in more complex situations as observed by De Campos et al. [11] and Mauá et al. [23, 24]. In more recent work, Mauá et al. [15] have also been able to introduce imprecision in SPNs, creating credal sum-product networks (CSPNs). In these CSPNs the weights of sum nodes are varied in a closed and convex set. The authors prove that efficient imprecise likelihood queries and imprecise conditional expectation queries are possible, under the assumption that the CSPN structure is a tree.

1.1 Problem statement

A limitation of CSPNs as introduced by Mauá et al. [15] is that tractable imprecise conditional inference is only proven for tree structures, as DAG structures require special attention with additional assumptions. Furthermore, the authors have only considered imprecise likelihood queries and imprecise conditional expectation queries in their proofs, using specialized factorizing functions. Hence, the following two problems have been observed:

1. Efficient imprecise conditional expectation queries are only proven for tree structures.
2. The imprecise expectation algorithm is not proven for generalized factorizing functions.

Before discussing the problems in detail, the workings of CSPNs are explained in combination with the possible queries. Imprecision is added to SPNs by varying the sum node weights, which allows for computing the lower and upper bounds of a query and thus being able to perform a robustness analysis. The imprecise queries which Mauá et al. [15] have proven are the likelihood query on DAG structures and the conditional expectation query on tree structures. The imprecise likelihood queries can be computed in a similar way to computing marginal probabilities in SPNs.

The conditional expectation query is used to compute the expectation of some function f over query variable X_q, conditional on some evidence. The function f is a univariate rational-valued

function on a single random variable X_q not in the evidence. In order to compute the conditional expectation, a fixed rational value μ is subtracted from the function result. If the expectation of this new expression is equal to 0, the value μ represents the conditional expectation of f given the evidence. The value for μ can be found efficiently through binary search. This query is proven for tree structures. An application of an imprecise conditional expectation query is credal classification, which is discussed in a later section.

1.1.1 Limited to tree structures

The cause of the first problem is that imprecise conditional expectation queries are solved efficiently by evaluating the SPN structure bottom up while locally solving the imprecise expectation of each node. However, in case the SPN structure is a directed acyclic graph, this local optimization approach can cause conflicts when a node is reused in different parts of the structure. In this case the imprecise expectation of a node cannot be computed locally causing the optimization problem to be delayed and resulting in intractability.

In order to understand which situations cause intractability in imprecise conditional queries, an example CSPN structure is given in Figure 1.2. Imagine that X_1 is the query variable, meaning that the values propagated from this node's function f can be both positive or negative. Consequently, the values propagated by nodes with a distribution over X_0 are always non-negative, as these represent indicator variables. Lets also assume that function f on X_1^a results in a negative value, while function f on X_1^b results in a positive value. Computing the lower bound expectation over the SPN using the imprecise conditional expectation algorithm of Mauá et al. [15] first of all results in computing the lower and upper bound of the bottom sum node, by varying weight w_3 and w_4. In order to compute the lower bound value for the left product node, the upper bound of the bottom sum node is multiplied by the negative value for node X_1^a. However, the lower bound of the right product node is computed using the lower bound of the bottom sum node, multiplied by the positive value for node X_1^b.

As there is a conflict in using both the upper and lower bound values of the bottom sum node, the decision of optimizing weights w_3 and w_4 is delayed to the node which closes the cycle. This affects the algorithm as each node value cannot be determined locally anymore, which in turn impacts the efficiency of the algorithm, especially when the cycles are larger or more frequent. Hence, the existing work focused on tree-structured SPNs.

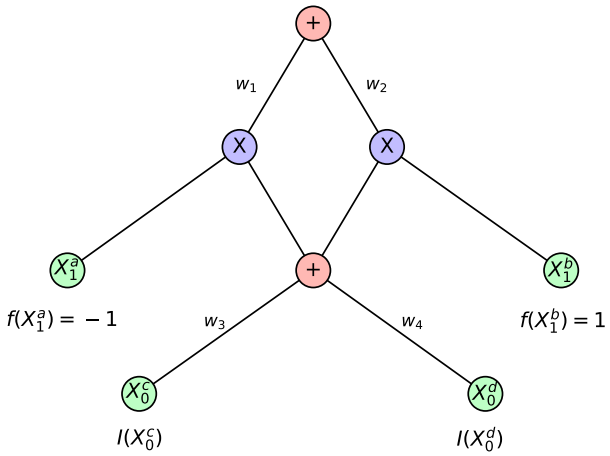


Figure 1.2: Illustration of a problematic CSPN structure.

1.1.2 Limited imprecise query functions

The second observed problem is related to the types of factorizing functions and queries which have been proven to work. The paper of Mauá et al. [15] only allows the following imprecise queries to be evaluated efficiently:

1. Imprecise likelihood query with indicator variables as the factorizing function.
2. Imprecise conditional expectation query with a single query or focus function with all other factors being indicator variables.

Both of these queries are specialized cases of an imprecise expectation query where the factorizing functions are generalized. The factorizing function of which the expectation is computed should factorize over the random variables in order to maintain tractable inference. This means that the function f should decompose into different independent functions over the models' random variables, i.e. $f(X, Y) = f(X)f(Y)$. The indicator variables $I(X)$ can be seen as a standard factor, since a factorizing function with only indicator variables would compute the likelihood. As opposed to indicator variables which are always non-negative, univariate functions $f(X)$ can be both positive or negative and allow for more complex queries. The factorizing function for the example SPN structure of Figure 1.2 would be:

$$f(X_0, X_1) = I(X_0)f(X_1) \tag{1.1}$$

1.2 Research results

The work of this thesis is based on the following research question:

To what extent can we generalize the state-of-the-art credal SPNs while maintaining efficient inference?

In order to answer the research question, the following sub-questions are answered in the thesis:

1. Do credal SPNs allow for efficient inference in DAG-shaped structures and are the results robust?
2. What are the advantages of a non-tree shaped credal SPN with respect to a tree-shaped credal SPN?
3. Can credal SPNs be used to efficiently compute the imprecise expectation of general factorizing functions?
4. Can soft evidence as a factorizing function benefit from imprecise efficient inference?

This thesis extends the work of Mauá et al. [15] on efficient inference of lower and upper expectations for CSPNs in the following three ways:

1. A general factorizing function can be efficiently evaluated with imprecision on tree structures.
2. A factorizing function with all factors non-negative can be efficiently evaluated with imprecision on DAG structures.
3. A general focused function can be efficiently evaluated with imprecision on DAG structures for which no cycle edge is part of the paths from root to focus variable leaves.

The DAG structures which are considered in the third case will avoid the problematic case as discussed in the problem statement, hence making sure that the local optimization of lower and upper expectations can be maintained. An example DAG structure of which no cycle edge is part of the paths from root to focus variable leaves can be seen in Figure 1.3.

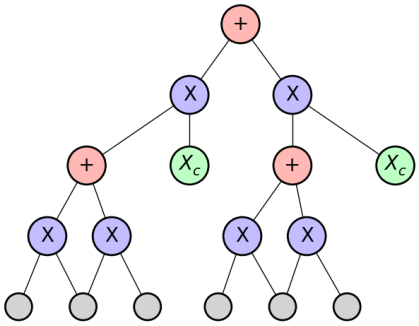


Figure 1.3: Illustration of a DAG structure of which no cycle edge is part of the paths from root to focus variable leaves X_c.

Additionally, the imprecise expectation algorithm of Mauá et al. [15] is generalized to allow any factorizing function which decomposes over the leaf variables, instead of having functions fixed to the indicator functions. The imprecise expectation algorithm on a CSPN structure can be computed efficiently, assuming that the algorithm is proven for the CSPN structure and that the decomposed functions can be computed in constant time.

In order to show that these three cases hold, a proof is constructed for the generalized problem of computing imprecise expectations of any factorizing function. Furthermore, this proof is extended beyond tree-shaped networks towards specialized directed acyclic graph structures.

Next to this, two experiments are performed, of which one compares the performance of a credal classification task on both tree-shaped networks and DAG-shaped networks. The other experiment makes use of the generalized factorizing function to add a form of soft evidence to the credal classification, which is again performed on tree- and DAG-shaped structures.

1.3 Thesis outline

This work is organized as follows. First, a literature review in Chapter 2 will be performed on SPNs, its structure learning algorithms and how imprecision is approached. Next, Chapter 3 will consist of definitions and notations to be used later in the work. Chapter 4 contains the new generalized imprecise expectation algorithm alongside the proof. Experiments with the newly proven algorithm are performed in Chapter 5. Finally, Chapter 6 contains the conclusion and recommendations for future work. The code used to perform the experiments can be found in the repository [4].

Chapter 2

Literature review

In recent years probabilistic circuits have made significant advances in both performance and applying imprecision efficiently. These advances allow for larger models with more complex data to be trained and evaluated, while keeping epistemic uncertainty in check by measuring the robustness. An example of epistemic uncertainty is when the model does not sufficiently cover the given data.

In this chapter, we will first review literature concerning sum-product networks and their recent advances in quality and performance. Furthermore, several structure learning techniques will be reviewed, as the structure of models is of importance in order to compute robustness using credal sets. Finally, literature regarding robustness evaluation in SPNs will be discussed, as well as how our work extends the existing work.

2.1 Sum-product networks

Poon et al. [36] first introduced sum-product networks in 2011 as a rooted directed acyclic graph with either product or weighted sum operations for the internal nodes and boolean variables as the leaves. They try to find general conditions for a model under which the partition function used for inference is tractable. The partition function Z is also called the normalization constant and is used to normalize the probability distribution, i.e. $P(x) = \frac{S(x)}{Z}$ where $Z = \sum_{x' \in X} S(x')$. The reasoning for this, is that exact tractable inference has not been explored for deep architectures. The authors find that the conditions of completeness and consistency together make a valid SPN, while the decomposability condition is a more restricted version of consistency. The completeness constraint applies to sum nodes and ensures that the sum node children act on the same set of variables. Consistency ensures that no variable can appear with different values in the children of product nodes. Such a valid SPN allows for computing the probability given some evidence in time linear in the number of nodes. This allows for the following queries to be evaluated efficiently with valid SPNs:

1. Evidence: probability of evidence for all variables.
2. Marginal: probability of some evidence, summing out all other variables.
3. Conditional: probability of some event, given some evidence.

Poon et al. build on the ideas of Darwiche [10], who discusses how probabilistic queries are computed by differentiating the polynomial efficiently, using a probabilistic circuit.

Gens et al. [17] present the first discriminative training algorithm of SPNs, since previous work only discusses generative methods for training SPNs. The discriminative training approach achieves higher accuracies in classification tasks compared to a generative training approach, while tractability of SPNs is maintained. Generative training is normally achieved by either computing the likelihood gradient and optimizing the sum weights with gradient descent (GD), or by

using Expectation-Maximization [16] (EM) with marginal inference. The discriminative training approach of Gens et al. [17] works by computing the gradient of the conditional log likelihood using backpropagation, applying weight updates using these gradients and normalizing the weights of a sum node. Hence, both generative and discriminative training approaches are options for SPNs, making both generative and discriminative tasks feasible, possibly using the same SPN structure.

Peharz et al. [34] explore some theoretical properties of SPNs that are not yet well understood. They for example show that normalized SPNs, with the weights of each sum node locally normalized, are not a weaker class than non-normalized SPNs. Additionally, they present an algorithm for converting any valid SPN to such a locally normalized SPN. Furthermore, they show that complete and consistent SPNs can be transformed into a complete and decomposable SPN by using a network polynomial in size of the original network, hence maintaining tractable inference. Finally, they extend the inference mechanisms from Darwiche [10] to generalized SPNs, which consider both discrete and continuous random variables. This means that any input distribution can be used for random variables in an SPN, as long as an integral or summation over the input distribution can be computed efficiently, since these integrals and summations are pushed down from the root node, in for example tractable marginalization queries.

A major limitation of probabilistic circuits is scalability and parallelism due to their sparsely connected networks. Peharz et al. [33] recently made some very notable improvements by combining a large number of arithmetic operations in a single einsum-operation, to be parallelized on the GPU. This einsum-operation structure is illustrated in Figure 2.1, where each node is vectorized to size K, meaning that a node represents K unique instances of such a node in a vector. The product nodes compute the outer product of the two child nodes, resulting in a $K \times K$ matrix. The sum nodes convert this matrix to a vector of size K by matrix-multiplication with the normalized weight matrix $\mathbf{W}$ of size $K \times K \times K$. This computation can be expressed in Einstein notation

$$\mathsf{S}_k = \mathbf{W}_{kij} \mathsf{N}_i \mathsf{N}'_j \tag{2.1}$$

where the axes with equal indices get multiplied and indices not used in the result get summed out. These computations are also called Einstein summations or einsum-operations. Additionally, numerical underflow is prevented by performing the computation in the log-domain, which requires the introduction of the log-einsum-exp trick, like the classical log-sum-exp trick [26].

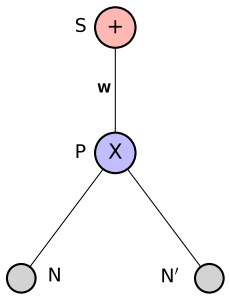


Figure 2.1: The einsum-operation SPN structure.

Furthermore, the authors show that Expectation-Maximization [16] can be simplified for PCs. The authors propose a new structure named Einsum Networks, which can lead to speedups and memory savings of two orders of magnitude. An application of Einsum Networks as a generative image model is also presented on the SVHN and CelebA image datasets, which result in a significant number of input variables in the network with respect to datasets used in other papers, thus showing that Einsum Networks can be used on large datasets.

2.2 SPN Structure learning

As opposed to neural networks, where the structure of a model often needs to be defined by hand, the structure of SPNs is often learned from the given dataset. Poon et al. [36] propose a systematic approach of constructing a structure, tailored for images as input. This structure recursively slices an image on either the horizontal or vertical axes. This is repeated, until a slice exactly represents a pixel. For each rectangular region, all possible slices into subregions are considered, where similar subregions are referenced to the same nodes, making this structure a DAG. The regions are assigned a number of sum nodes, while the nodes between each region are connected through product nodes constructed by a Cartesian product over the subregions. An example structure is shown in Figure 2.2. This structure has also been used in conjunction with Einsum Networks, as proposed by Peharz et al. [33], and is analogous to convolutional neural networks [28] when comparing to traditional neural networks.

Van de Wolfshaar et al. [42] present an extension to the image-tailored structure, where the Deep Generalized Convolutional SPN (DGC-SPN) structures allow for overlap between the image slices and parallelization on GPUs. This is possible by creating a layered network of large sum and product layers. A product layer contains many product nodes, where a set of scopes is split into many subsets which can overlap. Multiple copies of these scope subsets are made, which are called channels. Next, the sum nodes connect to the product nodes over the different channels. The authors argue that DGC-SPNs outperform all other structure learning algorithms, like the image-tailored approach or DCSPNs [2].

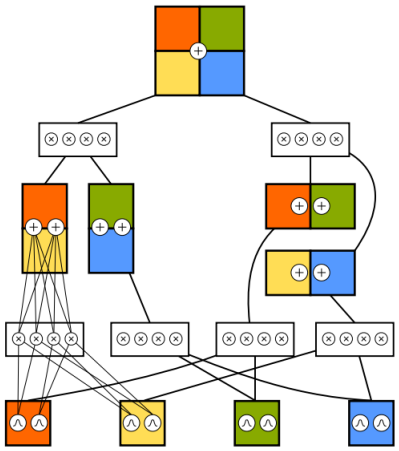


Figure 2.2: Example structure of the image-tailored algorithm. (source: [40], permission given by authors)

Another more general structure learning algorithm is LearnSPN as proposed by Gens et al. [18]. The LearnSPN algorithm takes full advantage of expressiveness of SPNs by recursively attempting to divide the data and variables in approximately independent subsets, creating either sum or product nodes. However, the algorithm does not have fixed steps to achieve this splitting of variables into subsets and can incorporate various clustering methods. First, the variables are clustered with for example a G-test [43]. If this clustering is achieved with approximate independence between the variables, a product node is introduced to split the features. Otherwise, instances are clustered, for example using hard incremental EM [27], and a sum node is introduced to split the dataset instances. The created structure is always a tree, as the structure is built up top-down by greedily clustering the variables. Rooshenas et al. [37] propose improvements to the LearnSPN algorithm by combining Markov networks [19] with LearnSPN, ending up with a best-of-both-worlds algorithm ID-SPN. The algorithm learns both indirect and direct interactions, combining approaches in graphical models and SPNs. The ID-SPN algorithm creates DAG structures, as the

arithmetic circuits are not necessarily trees.

Peharz et al. [35] present a more recent structure learning algorithm, where they propose to generate unspecialized random structures in a scalable manner. They reason that existing structure learners do not scale very well, are tedious to tune and are hard to integrate into existing deep learning frameworks. The RAT-SPN algorithm by Peharz et al. works by first constructing a random region graph over the dataset by splitting the features in a recursive manner, which is done up to a certain depth or when a single variable remains. Additionally, multiple repetitions of splitting the features can be performed which are all connected to the root region, containing all features. Using the region graph, an SPN can easily be constructed by adding sum nodes for every region with the leaf and root nodes getting special attention. Next, these sum nodes are connected through product nodes which are formed by Cartesian products over the sum nodes of two split regions. This SPN structure learning algorithm is very similar to the image-tailored approach and similarly can contain nodes with multiple parents, making this structure also a DAG. The authors conclude that one can get significant results without using sophisticated structure learners like the one described in by Rooshenas et al. [37] and that this simple approach allows wider use of PCs in the deep learning community.

2.3 Robustness in SPNs

Making a machine learning model robust is important for applying such models in the real world. Take for instance a model which is trained to classify traffic signs [20]. If the model would be trained without taking robustness into account, adversarial attacks could be performed by slightly changing the input image, resulting in an incorrect classification output and possibly a catastrophic decision being made, for example a slightly altered stop sign could be detected as a priority sign. Alongside critical real-time examples, knowing the degree of uncertainty of the model for specific data allows for efficiently making informed decisions in processes like fraud detection or other processes where a high uncertainty should result in a manual check. With more and more machine learning models being applied in the real world, the research on robust classification is a relevant research direction.

2.3.1 Credal sets

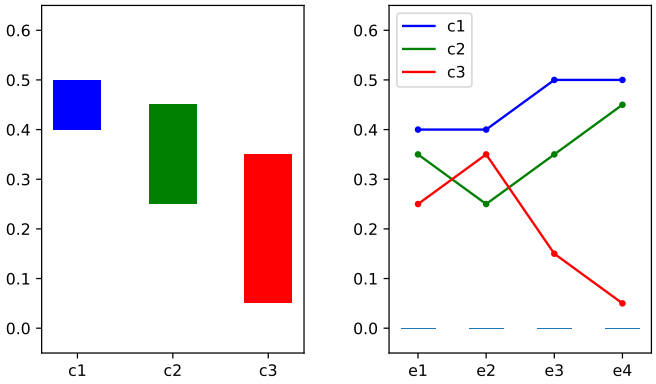
One way of adding imprecision in order to allow for robustness analyses to be performed is by using credal sets. Credal sets were introduced by Levi [21] and are sets of probability distributions. Such a set is closed, if all its limits are contained inside the set. The set is convex if all values between any two elements are part of the set. The credal set can also be seen as an interval where the lower and upper bounds specify the limits of the set. As a robustness analysis is performed on the minimum and maximum values caused by imprecision, only these limit values are considered.

Zaffalon [44] describes how credal sets can be used to extend naive Bayes classifiers (NBCs) and perform credal classification. This is done by assigning a local credal set to each random variable, resulting in model with the name, naive credal classifier (NCC). Zaffalon argues that for any imprecise probabilistic model a credal set can either be characterized by the convex hull of a set of distributions, or by means of linear constraints. The definition of a credal classifier is set as "a function that maps instances of attributes to a set of categories." The output of such a classifier is thus not necessarily a single class, but a set of all likely classes. Next, two methods of classification are presented by the author, being interval dominance and credal dominance.

Interval dominance can be applied by comparing whether the lower bound of one interval is greater than the upper bound of another interval. If this is the case, the first interval dominates the second. If there is any overlap, the intervals cannot be compared. Credal dominance is introduced by Zaffalon [44] and is able to utilize the information in credal sets, which is absent in the intervals. It could for example be the case that the value of a certain class is always greater than another class in all extreme points of the set. With interval dominance, the values of different extreme

points are also compared. This is not the case with credal dominance, as the credal sets are used to perform classification.

Consider the following example, also used by Zaffalon [44], where there are three classes $c1$, $c2$ and $c3$, and four extreme points: $(0.40, 0.35, 0.25)$, $(0.40, 0.25, 0.35)$, $(0.50, 0.35, 0.15)$ and $(0.50, 0.45, 0.05)$. The interval dominance and the credal dominance of this example is illustrated in Figure 2.3. The interval dominance shows that $c1$ dominates $c3$, but $c1$ and $c2$ are overlapping, while the credal dominance shows that in all extreme points $c1$ dominates both $c2$ and $c3$.



(a) Interval dominance (b) Credal dominance

Figure 2.3: Comparing interval dominance to credal dominance

2.3.2 Credal sum-product networks

An approach for making sum-product networks robust is proposed by Mauá et al. [15], where the degree of robustness is determined by making use of closed and convex sets of coefficients to vary the weights of sum nodes. The weights are varied by applying ϵ-contamination, using a value $0 \leq \epsilon \leq 1$ to denote the amount of imprecision added to the weight vector. Contaminating vector w can be expressed as follows:

$$\mathcal{Q}_{w,\epsilon} = \left\{ (1-\epsilon)w + \epsilon v : v_j \geq 0, \sum_j v_j = 1 \right\} \tag{2.2}$$

where $\mathcal{Q}_{w,\epsilon}$ represents the set of possible weight vectors. The ϵ-contamination is performed locally on each sum node weight vector in order to add imprecision. The resulting structure is named a credal sum-product network (CSPN).

The imprecision in the sum nodes allows for computing lower and upper bounds on the node values, which are useful for performing robustness analysis on queries. The combination of a lower and upper bound represent an interval, where the values in this interval all map to a distinct SPN that complies with the contaminated weight vector set. Mauá et al. [15] first of all show that the likelihood interval of a CSPN can be computed efficiently for any valid SPN structure. This is done by propagating both the minimum and maximum likelihoods through the network, where the sum nodes require solving a linear program in order to determine the minimizing and maximizing credal set weights.

Next, computing the upper and lower bounds on the conditional expectation of a function $f(x_q)$ given some evidence e is discussed, i.e. $\min_{\mathbf{w}} \mathbb{E}_{\mathbf{w}}[f|X_\varepsilon = e]$. The authors propose an algorithm which is able to compute said values efficiently, under the assumption that each internal node has at most one parent. Hence, the structure must be a tree. The algorithm can compute the

conditional expectation efficiently, by first creating function $g_\mu(x_q) = f(x_q) - \mu$. The value for μ can be found using binary search in the following equation:

$$\min_{\mathbf{w}} \mathbb{E}_{\mathbf{w}}[g_\mu(x_q)] = 0 \tag{2.3}$$

By searching a value for μ such that the equation equals 0, μ equals the desired conditional expectation value. By choosing a function f over the (bounded) class variable X_q, credal classification by means of credal dominance can be performed over tree-structured SPNs. Credal dominance of class c_1 over class c_2 can be evaluated efficiently by choosing $f(c_1) = 1$, $f(c_2) = -1$ and $f(x_q) = 0$ otherwise. In the remaining part of the paper, the authors show that taking into account robustness by using credal sets results in a better classification accuracy compared to taking the difference in class probabilities.

There already exist a number of papers where CSPNs are utilized successfully. For instance constructing a hierarchy of SPNs using robustness by Conaty et al. [5]. The authors propose a new method for ensemble learning using SPNs by chaining multiple SPNs together. The robustness value is computed using CSPNs and is used to determine whether the result can be used or the next model is consulted, depending on some threshold value. They also prove that this approach can only improve the classification accuracy, which is validated by experiments.

Another paper on robust SPNs is by Correia et al. [6], where the efficiency of the credal set algorithm is addressed. This is done twofold, with the first being the use of a class-selective SPN structure. A class-selective SPN structure has a sum node as the root, with for each unique class a product node as direct child node. Each of these product nodes has an indicator leaf node for the specific class, and a sibling structure for the remaining variables as children. As this structure divides the different unique classes at the root node, credal dominance can be determined by only evaluating the descendant nodes of the two classes being compared. Furthermore, memoisation of results in internal nodes is proposed, which can greatly reduce the number of nodes to be evaluated. The experiments discussed in the paper show that both of these changes have a positive effect on the evaluation times of inference queries.

A different application of robustness is presented by Correia et al. [7] for deep generative forests. The authors are able to extend random forest models towards the new generative forests, which can utilize credal sets to compute robustness values. In experiments, a positive correlation is observed between the accuracy and the robustness value.

Lastly, De Wit et al. [12] show an application of robustness on real-world data, where the data consists of both continuous and categorical variables. The adaptations for CSPNs to handle continuous variables are relatively simple, as the density value in continuous leaf nodes can be computed and propagated. The internal nodes do not need alterations to handle continuous variables. The authors of the paper empirically show that the algorithm proposed in [15] still works in polynomial time and show this on a dataset with the goal of detecting fraudulent orders.

2.4 Summary

In this chapter, a literature review is performed on sum-product networks, structure learning techniques and imprecision and robustness on SPNs. The literature review on sum-product networks describes how SPNs have been created as well as how they have been evolving over time. This literature helps in defining an SPN in Chapter 3. There are many structure learning algorithms for SPNs, which are also relevant to being able to perform a robustness analysis efficiently. Different tree and DAG structures are described, of which the DAG structures are relevant to the proposed research question. These structures are used in the experiments of Chapter 5. Finally, the literature review on robustness in SPNs describes how credal sets work and how interval and credal dominance compare to each other. It also explores the state-of-the-art literature of robustness in credal sum-product networks while also describing applications and adaptations. The described imprecise inference algorithm is generalized in Chapter 4. This thesis builds on top of the state-of-the-art by generalizing and expanding the efficient imprecise inference algorithm.

Chapter 3

Formal definition of credal PCs

In this chapter, notations and formal definitions on Probabilistic Circuits and possible structures are given, along with how credal sets are applied to a Probabilistic Circuit. The definition of a circuit is given in recursive modules where each internal circuit represents a probabilistic distribution constructed by an operation on the child circuit distributions.

3.1 Random variables and inputs

For a formal definition of the problem, we first define some used notations for random variables (RVs). RVs are denoted by a capital letter, i.e. X, Y and Z. The set of possible values of X is denoted by $\mathbf{val}(X)$, with lower-case letters denoting single values, $x \in \mathbf{val}(X)$. A set of RVs is defined as a bold capital letter, e.g. $\mathbf{X} = \{X_1, \cdots, X_n\}$. Now, $\mathbf{val}(\mathbf{X})$ is defined as the Cartesian product of all possible values of $\mathbf{X}$, with $\mathbf{x} \in \mathbf{val}(\mathbf{X})$. A probabilistic circuit is defined over some set of RVs $\mathbf{X} = \{X_1, \cdots, X_n\}$, as will be described in Section 3.3.

3.2 Factorizing functions

Each RV $X_i \in \mathbf{X}$ in a circuit can be assigned a univariate function $f_i(X_i)$ accompanied over its domain. These functions can be combined to create a factorizing function over $\mathbf{X}$ as follows: $f(\mathbf{X}) = \prod_{i=1}^{n} f_i(X_i)$, being a function over the Cartesian product set of $\mathbf{X}$. The function f is thus decomposable up to the individual functions f_i.

We identify the following three types of factorizing functions:

1. General factorizing function
2. Non-negative factorizing function
3. General focused query function

Where a general factorizing function over $\mathbf{X}$ is constructed from a set of univariate functions over $X_i \in \mathbf{X}$ without further limitations. A non-negative factorizing function is a general factorizing function where each function $f_i(X_i)$ is non-negative. Finally, a general focused query function can use any univariate function for the focused query variable $X_q \in \mathbf{X}$ while the other variables require non-negative functions. An example univariate function is the indicator function $I_x(X)$, which can either be the value 1 or 0, depending on the input.

Definition 1 (General factorizing function)**.** Let $\mathbf{X} = \{X_1, \cdots, X_n\}$ be the set of RVs, the general factorizing function is given as

$$f(\mathbf{X}) = \prod_{i=1}^{n} f_i(X_i) \tag{3.1}$$

with f_i as univariate functions.

Definition 2 (Non-negative factorizing function)**.** Let $\mathbf{X} = \{X_1, \cdots, X_n\}$ be the set of RVs, the non-negative factorizing function is given as

$$f(\mathbf{X}) = \prod_{i=1}^{n} f_i(X_i) \text{ with } f_i(X_i) \geq 0 \tag{3.2}$$

with f_i as univariate functions.

Definition 3 (General focused query function)**.** Let $\mathbf{X} = \{X_1, \cdots, X_n, X_q\}$ be the set of RVs with X_q being the focus variable, the general focused query function is given as

$$f(\mathbf{X}) = f_q(X_q) \prod_{i=1}^{n} f_i(X_i) \text{ with } f_i(X_i) \geq 0 \tag{3.3}$$

with f_i and f_q as univariate functions.

Definition 4 (Indicator function)**.** Let X be a random variable with $x \in \mathbf{val}(X)$, the indicator function is defined as

$$I_x(X) = \begin{cases} 1 & \text{X = x} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

3.3 Probabilistic Circuits

Probabilistic Circuits (PCs) can be seen as a recursively built-up network or circuit, representing joint probability distributions. The recursive structure becomes apparent when considering that circuits with an operation are decomposed into smaller circuits and thus simpler joint probability distributions. Possible operations are a summation or product operation. The stopping condition in a probabilistic circuit occurs, when a simple random variable is expressed as some distribution or indicator variable. The recursive circuit structure allows one to reason about each circuit individually, where the direct child circuits' probability distributions are used to create a new probability distribution.

The graphical model consisting of nodes and edges allows for visualizing the computational operations on the distributions as a directed acyclic graph, like in Figure 3.1. Each circuits' probability distribution can be mapped to a subgraph with a single node at its root. This node corresponds to both the circuit and its operation. Connections between circuits and their child circuits are represented by directed edges, with edges below a sum operation node are assigned a normalized non-negative weight value.

Next, a probabilistic circuit will be formalized by starting from the most simple PC and extending the definition to accommodate other recursive features. We will denote individual probabilistic circuits by calligraphic notation $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ or $\mathcal{L}$. Sets of circuits, like the set of direct children, are denoted by capital letters, for example B or D.

In order to define a PC, the following definitions will be used.

Definition 5. $\mathrm{sc}(\mathcal{C})$ denotes the scope of circuit $\mathcal{C}$, being a subset $\mathbf{X}_\mathcal{C}$ of all RVs $\mathbf{X}$.

Definition 6. $\mathrm{ch}(\mathcal{C})$ denotes the direct child circuits of circuit $\mathcal{C}$ and is determined based on the circuit structure. The results of these child circuits are used for computing the result of $\mathcal{C}$.

Definition 7. $\mathrm{desc}(\mathcal{C})$ denotes all descendant circuits connected to circuit $\mathcal{C}$. Thus: $\mathrm{desc}(\mathcal{C}) = \bigcup_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} \mathcal{D} \cup \mathrm{desc}(\mathcal{D})$

Definition 8. $\mathrm{lv}(\mathcal{C})$ denotes all leaf circuits connected to circuit $\mathcal{C}$. Thus: $\mathrm{lv}(\mathcal{C}) = \bigcup_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} \mathrm{lv}(\mathcal{D})$

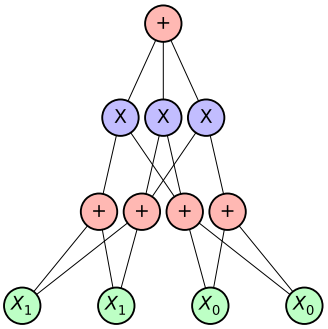


Figure 3.1: A probabilistic circuit over random variables X_0 and X_1, visualized as a directed acyclic graph.

Definition 9. $w_{\mathcal{C}}$ denotes the weight vector for circuit $\mathcal{C}$, with $|w_{\mathcal{C}}| = |\operatorname{ch}(\mathcal{C})|$ and

$$w_{\mathcal{C}} = \langle w_{\mathcal{C}\mathcal{D}} \mid 0 \leq w_{\mathcal{C}\mathcal{D}} \leq 1 \mid \mathcal{D} \in \operatorname{ch}(\mathcal{C}) \rangle, \text{ with } \sum_{\mathcal{D} \in \operatorname{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} = 1 \tag{3.5}$$

I.e. the weight vector has size equal to the number of children of $\mathcal{C}$ and is normalized with non-negative values.

Definition 10. $\mathbf{w}_{\mathcal{C}}$ denotes the weight set of circuit $\mathcal{C}$ and is defined as:

$$\mathbf{w}_{\mathcal{C}} = \{w_{\mathcal{C}}\} \cup \{w_{\mathcal{D}} \mid \mathcal{D} \in \operatorname{desc}(\mathcal{C})\} \tag{3.6}$$

for sum-based circuits and

$$\mathbf{w}_{\mathcal{C}} = \{w_{\mathcal{D}} \mid \mathcal{D} \in \operatorname{desc}(\mathcal{C})\} \tag{3.7}$$

for product-based circuits. The weight set $\mathbf{w}_{\mathcal{C}}$ is a collection of weight vectors under circuit $\mathcal{C}$.

The value of a circuit $\mathcal{C}$, with weight set $\mathbf{w}_{\mathcal{C}}$ and for a given query function $f_{\mathcal{C}}$ over RVs $\mathbf{X}_{\mathcal{C}}$, is equal to the expectation over the circuit, written as $\mathbb{E}_{\mathbf{w}_{\mathcal{C}}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})]$. The value of a circuit is computed using the same recursive structure as described earlier. The function f over a circuit $\mathcal{C}$ is denoted as: $f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = \prod_{\mathcal{D} \in \operatorname{ch}(\mathcal{C})} f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})$, where each $f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})$ can be decomposed up to the leaves. The leaf circuits $\operatorname{lv}(\mathcal{C})$ are each given a function $f_{\mathcal{L}}(\mathbf{X}_{\mathcal{L}})$ with $\mathcal{L} \in \operatorname{lv}(\mathcal{C})$ as input to determine their expectation. Hence, $f_{\mathcal{C}}$ is a factorizing function over all $\mathbf{X}_{\mathcal{C}}$ contained in $\operatorname{lv}(\mathcal{C})$, e.g. $f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = \prod_{\mathcal{L} \in \operatorname{lv}(\mathcal{C})} f_{\mathcal{L}}(\mathbf{X}_{\mathcal{L}})$, as defined in Section 3.2. Since $f_{\mathcal{C}}$ can be factorized, each internal circuit makes use of the child expectation values to compute a new expectation value, possibly using weight parameters from the set $\mathbf{w}_{\mathcal{C}}$. As each circuit represents a probabilistic model, the top circuit can represent a complex model.

With these definitions in place, the different types of circuits can be expressed, starting with the leaf circuits.

3.3.1 Leaf circuits

The most basic probabilistic circuit $\mathcal{C}$ can be defined on a single RV, e.g. the variable $X_{\mathcal{C}}$. This RV models some (discrete or continuous) distribution, for example a Gaussian distribution, of which the expectation of any uniform function $f_{\mathcal{C}}$ on its scope can be computed efficiently. The scope of this probabilistic circuit, $\operatorname{sc}(\mathcal{C})$, is defined as $\{X_{\mathcal{C}}\}$, while $\operatorname{ch}(\mathcal{C}) = \operatorname{desc}(\mathcal{C}) = \emptyset$ and $\operatorname{lv}(\mathcal{C}) = \{\mathcal{C}\}$. Furthermore, the weight vector and weight set are both empty.

The value of any circuit is equal to the expectation of the factorizing functions, hence the leaf circuit $\mathcal{C}$ with a uniform function $f_{\mathcal{C}}$ on variable $X_{\mathcal{C}}$ can efficiently compute its value as $\mathbb{E}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})]$.

A special case for leaf circuits is when the random variable distribution is a degenerate distribution. In this case, a single value $x \in \mathbf{val}(X)$ is assigned to the leaf circuit, where any evidence other than x results in a probability of 0. These types of leaf circuits are for example used in class-selective structures.

3.3.2 Sum operation

Next, circuits with a sum operation will be defined. A sum-based circuit $\mathcal{C}$ has child circuits connected, which are denoted by $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$. The circuit $\mathcal{C}$ must adhere to the smoothness constraint.

Definition 11 (Smoothness)**.** A sum-based circuit $\mathcal{C}$ is smooth iff $\mathrm{sc}(\mathcal{D}) = \mathrm{sc}(\mathcal{D}')$ for all $\mathcal{D}, \mathcal{D}' \in \mathrm{ch}(\mathcal{C})$ and all descendant circuits are smooth. I.e. the scopes of all direct child circuits are equal.

Furthermore, a weight vector $w_{\mathcal{C}}$ is used by circuit $\mathcal{C}$, where each child circuit $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$ has a non-negative weight $w_{\mathcal{C}\mathcal{D}} \in w_{\mathcal{C}}$ associated, see Definition 9.

The set of weights used in the complete circuit $\mathcal{C}$ is defined as $\mathbf{w}_{\mathcal{C}}$, which thus will also include possible weight vectors for descendant circuits of $\mathcal{C}$, see Definition 10.

The weight vector $w_{\mathcal{C}}$ is then used in combination with the results of the child circuits $\mathrm{ch}(\mathcal{C})$ to represent the result from $\mathcal{C}$ in the following way, using the factorizing function $f_{\mathcal{C}}$:

$$\mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})\right] = \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[\sum_{\mathcal{D}\in\mathrm{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})\right] = \sum_{\mathcal{D}\in\mathrm{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} \mathbb{E}_{\mathbf{w}_{\mathcal{D}}}\left[f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})\right] \tag{3.8}$$

3.3.3 Product operation

Circuits $\mathcal{C}$ with a product operation over the children $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$ will be defined next. As the product-based circuit $\mathcal{C}$ must comply with the following constraint:

Definition 12 (Decomposability)**.** A product-based circuit $\mathcal{C}$ is decomposable iff $\mathrm{sc}(\mathcal{D}) \cap \mathrm{sc}(\mathcal{D}') = \emptyset$ for all $\mathcal{D}, \mathcal{D}' \in \mathrm{ch}(\mathcal{C})$ and all descendant circuits are decomposable. I.e. the scopes of all direct child circuits are distinct.

we can state that $\mathrm{sc}(\mathcal{C}) = \bigcup_{\mathcal{D}\in\mathrm{ch}(\mathcal{C})} \mathrm{sc}(\mathcal{D})$. Furthermore, as the scopes are distinct, the children $\mathrm{ch}(\mathcal{C})$ are also distinct or non-overlapping. The product operation does not make use of any weights itself, so by Definition 10 $\mathbf{w}_{\mathcal{C}}$ is the set of weight vectors of all descendant circuits.

The value of circuit $\mathcal{C}$ is computed using the results of the child circuits $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$ as follows, given that each $f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})$ for $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$ is a factor of $f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})$:

$$\mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})\right] = \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[\prod_{\mathcal{D}\in\mathrm{ch}(\mathcal{C})} f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})\right] = \prod_{\mathcal{D}\in\mathrm{ch}(\mathcal{C})} \mathbb{E}_{\mathbf{w}_{\mathcal{D}}}\left[f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})\right] \tag{3.9}$$

3.3.4 Constraints

With the three different types of circuits explained, alongside the smoothness and decomposability constraints, we can state that a circuit is tractable if these constraints are met. This is called validity:

Definition 13 (Validity)**.** A circuit $\mathcal{C}$ is valid iff it is both smooth and decomposable.

In order to prove the imprecise expectation algorithm later in the work, we need to identify different cases with respect to a circuit and its weights being used more than once in a parent circuit. When this occurs due to the structure of some circuits, we define these circuits to be overlapping. It is important to make this distinction, as the optimization problem cannot be solved directly at the specific circuit and thus must be delayed.

Using Definition 8 we can make a distinction on whether two child circuits are overlapping or not:

Definition 14. For some circuit $\mathcal{C}$, two direct child circuits $\mathcal{D}, \mathcal{D}' \in \text{ch}(\mathcal{C})$ are overlapping iff $\text{lv}(\mathcal{D}) \cap \text{lv}(\mathcal{D}') \neq \emptyset$. I.e. the values of circuits $\mathcal{D}$ and $\mathcal{D}'$ are (partly) based on the same probability distributions.

Lemma 1. *The child circuits* $\text{ch}(\mathcal{C})$ *of a product-based circuit* $\mathcal{C}$ *cannot overlap and are thus disjoint.*

Proof. Assume that some circuit $\mathcal{C}$ is a product-based circuit with $\mathcal{D}, \mathcal{D}' \in \text{ch}(\mathcal{C})$ overlapping. From Definition 14 we know that $\text{lv}(\mathcal{D}) \cap \text{lv}(\mathcal{D}') \neq \emptyset$ should hold for $\mathcal{D}$ and $\mathcal{D}'$ to be overlapping. However, the decomposability constraint of Definition 12 requires that $\text{sc}(\mathcal{D}) \cap \text{sc}(\mathcal{D}') = \emptyset$. Any shared leaf circuit would contradict the decomposability constraint, invalidating the assumption. Thus, a product-based circuit cannot have overlapping children. □

3.3.5 Cycle

In describing the problem in Chapter 1, the notion of a cycle is used to characterize the set of circuits where a problem might occur in computing the imprecise expectation of a factorizing function. A definition for such a cycle, along with a definition for paths, will be given in this section. An example structure containing a cycle is illustrated in Figure 3.2.

Definition 15 (Path)**.** A path is a sequence of circuits from some start circuit $\mathcal{S}$ to some target circuit $\mathcal{T}$, following the direction of the child-parent connections. The path sequence $\{\mathcal{S}, \ldots, \mathcal{T}\}$ can be denoted as a set of circuits P.

Definition 16 (Cycle)**.** A cycle in a circuit $\mathcal{C}$ is defined as the set of circuits which are part of at least one of two distinct paths P and P' between some top circuit $\mathcal{T}$ and some bottom circuit $\mathcal{B}$. The two paths cannot share any circuit other than the top and bottom circuits, $P \cap P' = \{\mathcal{T}, \mathcal{B}\}$, meaning that there exists no circuit $\mathcal{D} \in P \cup P'$ higher than circuit $\mathcal{T}$ in the graph and likewise, that there exists no circuit $\mathcal{D} \in P \cup P'$ lower than circuit $\mathcal{B}$ in the graph.

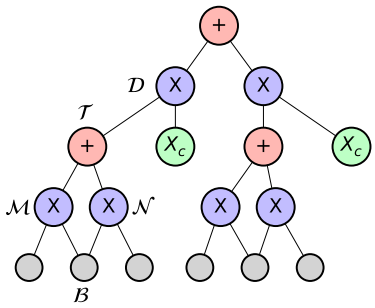


Figure 3.2: Example structure with a cycle $\{\mathcal{T}, \mathcal{M}, \mathcal{N}, \mathcal{B}\}$ constructed from the paths $P = \{\mathcal{T}, \mathcal{M}, \mathcal{B}\}$ and $P' = \{\mathcal{T}, \mathcal{N}, \mathcal{B}\}$. Circuit $\mathcal{D}$ cannot be the top circuit, as otherwise $\mathcal{T}$ is present in both paths.

Lemma 2. *For any valid circuit* $\mathcal{C}$*, the top circuit* $\mathcal{T}$ *of a cycle must always be a sum circuit.*

Proof. Assume that the top circuit $\mathcal{T}$ is a product circuit. We know that $\mathcal{T}$ must have at least two child circuits $\mathcal{D}$ and $\mathcal{D}'$, as there are two paths from some bottom circuit $\mathcal{B}$ towards the top circuit and these two paths do not share any circuits apart from the top and bottom circuits. Thus, $\text{sc}(\mathcal{B}) \subseteq \text{sc}(\mathcal{D})$ and $\text{sc}(\mathcal{B}) \subseteq \text{sc}(\mathcal{D}')$. Since $\text{sc}(\mathcal{D}) \cap \text{sc}(\mathcal{D}')$ at least contains $\text{sc}(\mathcal{B})$, the decomposability constraint is not satisfied. Hence, the assumption that $\mathcal{T}$ is a product circuit is incorrect and $\mathcal{T}$ must be a sum circuit, where the smoothness constraint can be satisfied. □

3.4 Probabilistic circuit structures

In this section, different types of structure learning algorithms are formally defined, which are used in the experiments of Chapter 5. The first structure learning algorithm to explain is the LearnSPN algorithm of Gens et al. [18]. Next, the RAT-SPN algorithm of Peharz et al. [35] is explained. Finally, the class-discriminative structure is defined in Section 3.4.3, which is also called a class-selective structure by Correia et al. [6].

3.4.1 LearnSPN

As already briefly discussed in the literature review (Chapter 2), the LearnSPN algorithm recursively attempts to divide either the data or the features in approximately independent subsets. Dividing of the data results in a sum node with the weights proportional to the partitions, with the scope of each child equal to the current scope. When features are divided, a product node is introduced, modeling independence between the partitioned features. As the algorithm recursively attempts to divide either the data or the features, the resulting structure is a tree which is already trained. An example structure of the LearnSPN algorithm can be seen in Figure 3.3

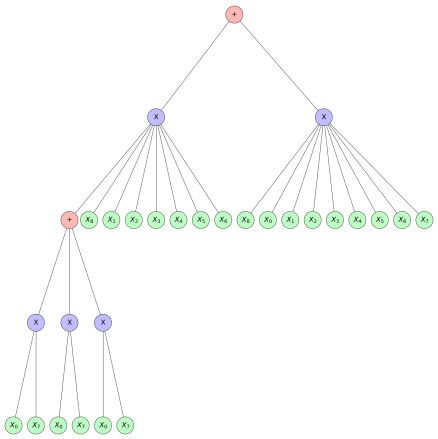


Figure 3.3: Example of a LearnSPN structure

Algorithm 1 shows the general steps taken to generate a LearnSPN structure, with T being the dataset instances and V being the variables. The partitions can be constructed using different clustering algorithms. Gens et al. [18] perform instance clustering using the Expectation-Maximization algorithm of Dempster et al. [13]. EM is possible as either soft EM, where samples can be partly assigned to each cluster, or hard EM, where samples are fully assigned to a single cluster. The authors of the LearnSPN algorithm make use of hard EM to perform clustering.

However, clustering could also be performed by using the Randomized Dependence Coefficient (RDC) of Lopez-Paz et al. [22]. This coefficient is a measure of non-linear dependence between random variables and can be transformed to construct clusters by computing connected components from a RDC adjacency matrix, as done by Molina et al. [25] for the SPFlow library.

K-means clustering is another clustering algorithm which can be used by the LearnSPN algorithm. The K-means clustering algorithm clusters data based on the closest cluster mean and is also implemented in the SPFlow library.

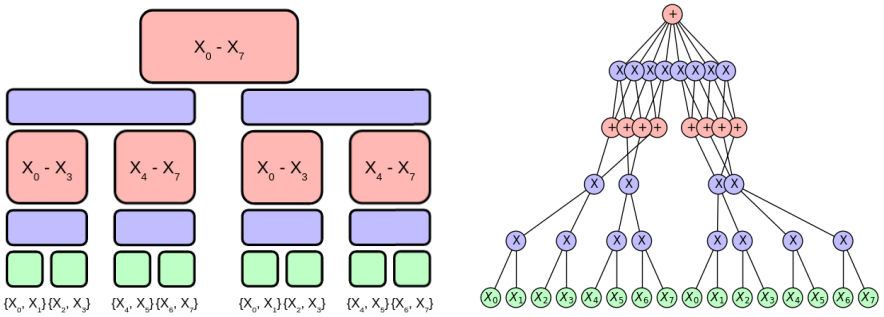
Choosing in which way the partitions over the data and features are taken is done by checking if the feature partitioning is completed successfully. The definition of a success differs between different clustering algorithms but can for example be expressed as a threshold, which is compared to a performance metric of the performed clustering.

Algorithm 1 LearnSPN structure learning algorithm (Gens et al. [18])

```
1: function LearnSPN(T,V)
2:     if |V| = 1 then
3:         return univariate distribution estimated from the variable's values in T
4:     else
5:         Partition V into approximately independent subsets V_j
6:         if success then
7:             return ∏_j LearnSPN(T, V_j)
8:         else
9:             partition T into subsets of similar instances T_i
10:            return Σ_i (|T_i|/|T|) · LearnSPN(T_i, V)
```

3.4.2 RAT-SPN

The RAT-SPN structure learning algorithm differs from other structure learning algorithms, as randomness is used to generate the structure. As discussed in the literature review, Peharz et al. [35] reason that the Random Tensorized SPNs (RAT-SPNs) can scale well and are easy to integrate into existing deep learning frameworks. The structure scaling is accomplished by using several hyperparameters which will be discussed in the next paragraphs, while the algorithm has been implemented in the Tensorflow [1] library by Peharz et al. and in the PyTorch [29] library by the authors of SPFlow [25].



(a) Random Region Graph (b) SPN structure

Figure 3.4: Example of a RAT-SPN structure, with $R = 2$, $D = 2$, $C = 1$, $S = 2$ and $I = 1$

The RAT-SPN algorithm works by first generating a random region graph [14, 30] $\mathcal{R}$. The root of this random region graph starts off with all variables ($\mathbf{X}$) and creates two balanced partitions $\mathbf{R}_1$ and $\mathbf{R}_2$ recursively until either the maximum depth (D) is reached or the partitions are of size 1. This procedure of splitting the partitions is repeated R times at the root, as can also be seen in Algorithm 2. The randomness of the algorithm originates in taking random balanced 2-partitions of the variables. The hyperparameters of this algorithm are the depth D, which can trivially limit the maximum depth of the region graph. Additionally, the number of repetitions R linearly scales the size of the graph, as R instances of total splits are inserted. An example random region graph can be seen in Figure 3.4a.

Algorithm 2 Random Region Graph (Peharz et al. [35])

function RANDOMREGIONGRAPH($\mathbf{X}$, D, R)
  Create an empty region graph $\mathcal{R}$
  Insert $\mathbf{X}$ in $\mathcal{R}$
  for $r = 1 \dots R$ **do**
    Split($\mathcal{R}$, $\mathbf{X}$, D)
function SPLIT($\mathcal{R}$, $\mathbf{X}$, D)
  Draw balanced partition $\mathcal{P} = \{\mathbf{R}_1, \mathbf{R}_2\}$ of $\mathbf{R}$
  Insert $\mathbf{R}_1, \mathbf{R}_2$ in $\mathcal{R}$
  Insert $\mathcal{P}$ in $\mathcal{R}$
  if $D > 1$ **then**
    if $|\mathbf{R}_1| > 1$ **then** Split($\mathcal{R}$, $\mathbf{R}_1$, $D-1$)
    if $|\mathbf{R}_2| > 1$ **then** Split($\mathcal{R}$, $\mathbf{R}_2$, $D-1$)

The random region graph is then converted to an SPN by first creating sum or leaf nodes for all regions $\mathbf{R} \in \mathcal{R}$. If the region $\mathbf{R}$ is not split any further, $\mathbf{R}$ is assigned with I number of leaf nodes. In the event that $\mathbf{R}$ is a root region, C sum nodes are created, where C is the number of classes or root nodes. Finally, all other regions are assigned S regular sum nodes. Next, the regions in $\mathcal{R}$ are joined together by using the partitioning to determine which regions are split. The nodes of both partitioned regions are combined in product nodes using the Cartesian product and are assigned as child nodes of all parent region nodes. Algorithm 3 describes the exact SPN generation code as proposed by Peharz et al. [35]. The hyperparameters C, S and I determine the finer structure, as these are used to indicate the number of nodes at each region. The number of leaf (I) and class (C) nodes are trivial and only are used at the top and bottom of the structure. However, the number of sum nodes S at each region not only determines the number of sum nodes, but also indirectly the number of product nodes due to the Cartesian product. Hence, S^2 product nodes are created above two regions of size S. An example structure of the RAT-SPN algorithm can be seen in Figure 3.4b.

The weights and leaf distribution parameters are initialized randomly and can be trained by the same optimization approaches as used for neural networks, like stochastic gradient descent (SGD) in combination with for example the Adam optimizer. Furthermore, automatic differentiation and GPU-parallelization are possible due to the easy integrations with existing deep learning frameworks.

Algorithm 3 Construct RAT-SPN from Region Graph (Peharz et al. [35])

function CONSTRUCTSPN($\mathcal{R}$, C, S, I)
  Make empty SPN
  for $\mathbf{R} \in \mathcal{R}$ **do**
    if $\mathbf{R}$ is a leaf region **then**
      Equip $\mathbf{R}$ with I distribution nodes
    else if $\mathbf{R}$ is the root region **then**
      Equip $\mathbf{R}$ with C sum nodes
    else
      Equip $\mathbf{R}$ with S sum nodes
  for $\mathcal{P} = \{\mathbf{R}_1, \mathbf{R}_2\} \in \mathcal{R}$ **do**
    Let $\mathbf{N}_{\mathbf{R}}$ be the node for region $\mathbf{R}$
    for $\mathsf{N}_1 \in \mathbf{N}_{\mathbf{R}_1}, \mathsf{N}_2 \in \mathbf{N}_{\mathbf{R}_2}$ **do**
      Introduce product $\mathsf{P} = \mathsf{N}_1 \times \mathsf{N}_2$
      Let P be a child for each $\mathsf{N} \in \mathbf{N}_{\mathbf{R}_1 \cup \mathbf{R}_2}$
  return SPN

3.4.3 Class-discriminative structure

In addition to the LearnSPN and RAT-SPN structure learning algorithms which generate a complete network, class-discriminative structures will generate the top circuit structure by making use of the unique classes and require another structure learning algorithm to complete the class-specific structure. Class-discriminative structures have also been named class-selective structures and defined by Correia et al. [6], as with a selective SPN [31], there is only a single sub-network evaluated for a given class value. However, with the generalized imprecise expectation algorithm multiple class-specific sub-networks could be active at the same time. Hence, the name class-discriminative is used in this work. The definition of class-discriminative structures is given below, with an example structure in Figure 3.5.

Definition 17. Let X_c denote the class variable with discrete values. A class-discriminative structure has a sum-based circuit $\mathcal{C}$ as the root, with $|X_c|$ product circuits as its children. Each of these class-specific product circuits has an indicator leaf circuit for the values $x_c \in \mathbf{val}(X_c)$ and a class-specific sub-network which is generated over the samples with the respective class using any structure learning algorithm. Furthermore, the class-specific product circuits are non-overlapping, as per Definition 14.

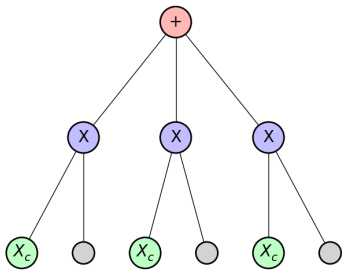


Figure 3.5: An example of a class-discriminative structure with 3 classes, with the sub-networks left out.

3.5 Credal probabilistic circuits

In the work of Mauá et al. [15], robustness of the network is investigated by perturbing the weights $\mathbf{w}$ of a circuit, as defined in Section 3.3.2. This is done by making use of ϵ-contamination on the weights, for each sum operation individually. The resulting model is named a credal sum-product network (CSPN).

A weight w for some sum-based circuit $\mathcal{C}$ is perturbed using ϵ-contamination as follows, where the weight vector w is normalized and $0 \leq \epsilon \leq 1$:

$$\mathcal{Q}_{w,\epsilon} = \left\{ (1-\epsilon)w + \epsilon v : v_j \geq 0, \sum_j v_j = 1 \right\} \tag{3.10}$$

Hence, $\mathcal{Q}_{w,\epsilon}$ resembles the set of all possible perturbations on the weight vector w using ϵ. A CSPN can be created by independently perturbing the weights of all sum-based circuits, contained in $\mathcal{Q}$, which is defined as $\{\mathcal{C}_{\mathbf{w}} : \mathbf{w} \in \mathcal{Q}\}$, containing all possible circuits $\mathcal{C}$ that can be created from varying the weights by ϵ. A variation for ϵ of 0 results in the original weights $\mathbf{w}$ being used and a variation of 1 results in all possible weight vectors to be included.

Chapter 4

General imprecise inference algorithm

In this chapter, the algorithm for computing the imprecise expectation of a factorizing function f over an SPN is introduced. Furthermore, a proof is given, showing that the algorithm is both correct and efficient. The proof is subdivided into many smaller proofs, which, when combined, can prove the correctness of many different probabilistic circuit structures and factorizing functions.

4.1 Imprecise expectation algorithm

From the definitions of Chapter 3, the value of a circuit $\mathcal{C}$ is equal to the expectation of a function $f_{\mathcal{C}}$ over the random variables $\mathbf{X}_{\mathcal{C}}$. In order to perform robustness analysis on this value, we will be computing the minimum and maximum values, $\min_{\mathbf{w}_{\mathcal{C}}} \mathbb{E}_{\mathbf{w}_{\mathcal{C}}} f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})]$ and $\max_{\mathbf{w}_{\mathcal{C}}} \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})]$, over all weight sets $\mathbf{w}_{\mathcal{C}} \in \mathcal{Q}$.

The minimum and maximum value can be computed using the algorithm described below, given some assumptions on the structure of the circuit. The allowed structures will be discussed in a later section. The algorithm is evaluated in reverse topological order, making use of the already computed values for the direct child circuits. Furthermore, a linear program is used for each sum circuit, while a greedy algorithm is used for each product circuit. $\underline{V}^{\mathcal{C}}$ will be the minimum value for circuit $\mathcal{C}$, while $\overline{V}^{\mathcal{C}}$ will be the maximum value.

- If $\mathcal{C}$ is a leaf circuit associated with some probabilistic distribution over $X_{\mathcal{C}}$, then:

$$\underline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \overline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \mathbb{E}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] \tag{4.1}$$

- If $\mathcal{C}$ is a product operation circuit with $k = |\operatorname{ch}(\mathcal{C})|$, then:

$$\underline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \underline{B}^{\mathcal{C}}_{k} \tag{4.2}$$

$$\overline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \overline{B}^{\mathcal{C}}_{k} \tag{4.3}$$

  Where $\underline{B}^{\mathcal{C}}_{k}$ and $\overline{B}^{\mathcal{C}}_{k}$ are computed using the following greedy algorithm, which computes the minimum and maximum values for $\underline{B}^{\mathcal{C}}_{j}$ and $\overline{B}^{\mathcal{C}}_{j}$ with $j = 1, \ldots, k$ over all possible combinations of:

  1. The lower and upper bound values of the last iteration ($\underline{B}^{\mathcal{C}}_{j-1}$ and $\overline{B}^{\mathcal{C}}_{j-1}$)
  2. The lower and upper bound values of child circuit $\mathcal{D}_j \in \operatorname{ch}(\mathcal{C})$ ($\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j})$ and $\overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j})$)

Initialize $\underline{B}_1^{\mathcal{C}} = \underline{V}^{\mathcal{D}_1}(f_{\mathcal{D}_1})$ and $\overline{B}_1^{\mathcal{C}} = \overline{V}^{\mathcal{D}_1}(f_{\mathcal{D}_1})$. Then, for all $j = 2, \ldots, k$:

$$\underline{B}_j^{\mathcal{C}} = \min\{\underline{B}_{j-1}^{\mathcal{C}}\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \underline{B}_{j-1}^{\mathcal{C}}\overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \overline{B}_{j-1}^{\mathcal{C}}\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \overline{B}_{j-1}^{\mathcal{C}}\overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j})\} \tag{4.4}$$

$$\overline{B}_j^{\mathcal{C}} = \max\{\underline{B}_{j-1}^{\mathcal{C}}\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \underline{B}_{j-1}^{\mathcal{C}}\overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \overline{B}_{j-1}^{\mathcal{C}}\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \overline{B}_{j-1}^{\mathcal{C}}\overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j})\} \tag{4.5}$$

- If $\mathcal{C}$ is a sum operation circuit, then:

$$\underline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \min_{w_{\mathcal{C}} \in \mathcal{Q}} \sum_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}}\underline{V}^{\mathcal{D}}(f_{\mathcal{D}}) \tag{4.6}$$

$$\overline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \max_{w_{\mathcal{C}} \in \mathcal{Q}} \sum_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}}\overline{V}^{\mathcal{D}}(f_{\mathcal{D}}) \tag{4.7}$$

4.2 Proof setup

As with the definition of PCs, we use $\mathcal{C}$ to denote the current circuit to be proven and $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$ to denote the direct child circuits. For the proofs, we use induction and assume that the child circuit values have already been computed efficiently. We also assume that the circuit adheres to the structural constraints of smoothness and decomposability and is thus valid. Finally, $f_{\mathcal{C}}$ is used to denote the factorizing function over $\mathcal{C}$, which factorizes into $\prod_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})$ for the direct children, and eventually into the functions $f_{\mathcal{L}}$ for $\mathcal{L} \in \mathrm{lv}(\mathcal{C})$.

The division of the different proofs is made on the type of circuit, the sign of child values and the structure of the child circuits, resulting in the following cases:

1. Leaf circuit
2. Product-based circuit
3. Non-negative product-based circuit
4. Sum-based circuit with disjoint children
5. Sum-based circuit with overlapping children

The proofs for these cases can be found in the next sections.

The objective of each proof is to show that the following theorem holds for each different case and can be computed efficiently, given some case-specific assumptions:

Theorem 3. *Assume that $\mathcal{C}$ is a valid circuit, $f_{\mathcal{C}}$ is a factorizing function and that $\underline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ and $\overline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ for $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$ have already been computed. Under the right assumptions for each case, it holds that:*

$$\begin{aligned} \underline{V}^{\mathcal{C}}(f_{\mathcal{C}}) &= \min_{\mathbf{w}_{\mathcal{C}} \in \mathcal{Q}} \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] = \underline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] \\ \overline{V}^{\mathcal{C}}(f_{\mathcal{C}}) &= \max_{\mathbf{w}_{\mathcal{C}} \in \mathcal{Q}} \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] = \overline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] \end{aligned} \tag{4.8}$$

and that the computation for circuit $\mathcal{C}$ is efficient.

If each case can be proven to be valid and efficient, traversing the probabilistic circuit from the leaves towards the root and computing the values of $\underline{V}^{\mathcal{C}}$ and $\overline{V}^{\mathcal{C}}$ for each circuit $\mathcal{C}$ makes the complete circuit valid and efficient.

4.3 Base case proof

Theorem 4. *Let $\mathcal{C}$ be a valid leaf circuit and $f_{\mathcal{C}}$ be a factorizing function which is efficient to evaluate. It holds that $\underline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \underline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})]$ and $\overline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \overline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})]$ for circuit $\mathcal{C}$ and can be computed in polynomial time.*

Proof. When $\mathcal{C}$ is a leaf circuit, there are no child circuits connected to $\mathcal{C}$. The function $f_{\mathcal{C}}$ is factorized to cover the scope of $\mathcal{C}$ and is assumed to be efficient to evaluate. Furthermore, the parameters used for resembling a distribution in this leaf are not shared with any other leaf circuits. Hence, the value computed for this type of circuit is completely self-contained.

$$\underline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] = \min_{\mathbf{w}_{\mathcal{C}}} \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] \tag{4.9}$$

$$= \mathbb{E}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] \tag{4.10}$$

$$= \underline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \overline{V}^{\mathcal{C}}(f_{\mathcal{C}}) \tag{4.11}$$

Since a leaf circuit has no weights, Equation 4.8 can already be simplified by removing the minimum operator. The expectation of the function $f_{\mathcal{C}}$ is equal to the lower and upper bounds of the algorithm. The upper bound is analogous.

Assuming that the value of the function $f_{\mathcal{C}}$ is already computed or can be computed in polynomial time, the value for $\mathcal{C}$ can be computed in polynomial time. □

4.4 Product operation proof

Theorem 5. *Let $\mathcal{C}$ be a valid product-based circuit and $f_{\mathcal{C}}$ be a factorizing function. Assume that Theorem 3 holds for circuits $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$. I.e. the values $\underline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ and $\overline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ have already been computed. It holds that $\underline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \underline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})]$ and $\overline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \overline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})]$ for product-based circuit $\mathcal{C}$ and can be computed in polynomial time.*

Proof. As $\mathcal{C}$ is a valid product-based circuit, Lemma 1 states that the children of a product circuit cannot overlap and are thus disjoint. Therefore we can state that all children and their weights are disconnected, i.e. $\mathbf{w}_{\mathcal{D}} \cap \mathbf{w}_{\mathcal{D}'} = \emptyset$ for $\mathcal{D}, \mathcal{D}' \in \mathrm{ch}(\mathcal{C})$, since otherwise scopes would overlap and not be disjoint. Thus, there cannot be any weight set $\mathbf{w}$ which has conflicting constraints on minimizing or maximizing the result. From this reasoning, we can conclude that the only possible structure for a product circuit and its direct children is a tree. The proof will only be given for $\underline{V}^{\mathcal{C}}(f_{\mathcal{C}})$, as $\overline{V}^{\mathcal{C}}(f_{\mathcal{C}})$ is analogous.

The value $\underline{V}^{\mathcal{C}}(f_{\mathcal{C}})$ of the algorithm is equal to $\underline{B}_k^{\mathcal{C}}$ for product circuits, as explained in Section 4.1. Due to the product circuit definition:

$$\underline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] = \min_{\mathbf{w}_{\mathcal{C}}} \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] = \min_{\mathbf{w}_{\mathcal{C}}} \mathbb{E}_{\mathbf{w}_{\mathcal{C}}} \left[\prod_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}}) \right] \tag{4.12}$$

By proving that

$$\underline{B}_j^{\mathcal{C}} = \underline{\mathbb{E}} \left[\prod_{i=1}^{j} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i}) \right] \text{ with } \mathcal{D}_i \in \mathrm{ch}(\mathcal{C}) \tag{4.13}$$

using induction over all $j = 1, \ldots, k$ and $k = |\mathrm{ch}(\mathcal{C})|$, we will have proven that Theorem 5 holds using Lemma 6. [1]

[1] Using a proof provided by dr. Thomas Krak

Base case ($j = 1$):
Since $\underline{B}_1^{\mathcal{C}} = \underline{V}^{\mathcal{D}_1}(f_{\mathcal{D}_1}) = \underline{\mathbb{E}}\left[f_{\mathcal{D}_1}(\mathbf{X}_{\mathcal{D}_1})\right]$ by using the definition of $\underline{B}_j^{\mathcal{C}}$ of Section 4.1 and Theorem 3, the base case holds.

Induction hypothesis ($j-1$):

$$\underline{B}_{j-1}^{\mathcal{C}} = \underline{\mathbb{E}}\left[\prod_{i=1}^{j-1} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})\right] \text{ with } \mathcal{D}_i \in \mathrm{ch}(\mathcal{C}) \tag{4.14}$$

Induction step (j):
From Lemma 6, it follows that:

$$\underline{B}_j^{\mathcal{C}} \geq \underline{\mathbb{E}}\left[\prod_{i=1}^{j} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})\right] \tag{4.15}$$

Next, the inequality in the other direction will be proven. Lets assume that

$$\underline{B}_j^{\mathcal{C}} > \underline{\mathbb{E}}\left[\prod_{i=1}^{j} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})\right] \tag{4.16}$$

There must be some $\mathbf{w}_{\mathcal{C}} \in \mathcal{Q}$ such that

$$\underline{B}_j^{\mathcal{C}} > \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[\prod_{i=1}^{j} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})\right] \tag{4.17}$$

Because $\mathbf{w}_{\mathcal{C}} \in \mathcal{Q}$ and the definition of the lower and upper expectations, it follows that

$$\underline{\mathbb{E}}[f_{\mathcal{D}_j}(\mathbf{X}_{\mathcal{D}_j})] \leq \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}[f_{\mathcal{D}_j}(\mathbf{X}_{\mathcal{D}_j})] \leq \overline{\mathbb{E}}[f_{\mathcal{D}_j}(\mathbf{X}_{\mathcal{D}_j})] \tag{4.18}$$

and by using the induction hypothesis:

$$\underline{B}_{j-1}^{\mathcal{C}} \leq \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[\prod_{i=1}^{j-1} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})\right] \leq \overline{B}_{j-1}^{\mathcal{C}} \tag{4.19}$$

From the independence of $f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})$ for $i = 1, \ldots, j$ under $\mathbf{w}_{\mathcal{C}}$, it follows that

$$\mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[\prod_{i=1}^{j} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})\right] = \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[\prod_{i=1}^{j-1} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})\right] \cdot \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}[f_{\mathcal{D}_j}(\mathbf{X}_{\mathcal{D}_j})] \tag{4.20}$$

By combining the above equations with the algorithm to compute $\underline{B}_j^{\mathcal{C}}$ and $\overline{B}_j^{\mathcal{C}}$, we find that

$$\mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[\prod_{i=1}^{j} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})\right] \in [\underline{B}_j^{\mathcal{C}}, \overline{B}_j^{\mathcal{C}}] \tag{4.21}$$

which contradicts the assumption that $\underline{B}_j^{\mathcal{C}}$ is greater than the lower expectation. Hence, the opposite of the assumption must be true, proving that $\underline{B}_j^{\mathcal{C}} = \underline{\mathbb{E}}\left[\prod_{i=1}^{j} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})\right]$ and consequently proving Theorem 5.

The product operation can be performed in polynomial time, as $\underline{B}_k^{\mathcal{C}}$ with $k = |\,\mathrm{ch}(\mathcal{C})|$ is computed greedily with a running time of $O(k)$. □

Lemma 6. *For all $j = 1, \ldots, k$, there are $\mathbf{w}, \mathbf{w}' \in \mathcal{Q}$ such that $\mathbb{E}_{\mathbf{w}}[\prod_{i=1}^{j} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})] = \underline{B}_j^{\mathcal{C}}$ and $\mathbb{E}_{\mathbf{w}'}[\prod_{i=1}^{j} f_{\mathcal{D}_i}(\mathbf{X}_{\mathcal{D}_i})] = \overline{B}_j^{\mathcal{C}}$*

Proof. We only consider the lower bound proof, since the upper bound is analogous. Assume $\mathcal{C}$ is a product-based circuit with $\mathcal{D}_i \in \mathrm{ch}(\mathcal{C})$ for $i = 1, \ldots, k$ with $k = |\mathrm{ch}(\mathcal{C})|$. Let $\mathbf{w}_{\mathcal{D}_i}$ denote the weight set for circuit $\mathcal{D}_i \in \mathrm{ch}(\mathcal{C})$.

The value of $\underline{B}_j^{\mathcal{C}}$ is computed using either the lower or upper bound values of circuit $\mathcal{D}_i$, i.e. $\underline{V}^{\mathcal{D}_i}(f_{\mathcal{D}_i})$ or $\overline{V}^{\mathcal{D}_i}(f_{\mathcal{D}_i})$, for $i = 1, \ldots, j$. These values are computed using weight sets $\mathbf{w}_{\mathcal{D}_i}$. From Lemma 1 we know that the child circuits of a product-based circuit are non-overlapping or disjoint. Therefore, the used weights are also disjoint and we can construct the weight set for product-based circuit $\mathcal{C}$ as follows:

$$\mathbf{w}_{\mathcal{C}} = \bigcup_{i=1}^{j} \mathbf{w}_{\mathcal{D}_i} \tag{4.22}$$

□

4.4.1 Non-negative product

Theorem 7. *Let $\mathcal{C}$ be a valid product-based circuit and $f_{\mathcal{C}}$ be a non-negative factorizing function. Assume that Theorem 3 holds for circuits $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$. I.e. the values $\underline{V}^{\mathcal{D}}(f_{\mathcal{D}}) \geq 0$ and $\overline{V}^{\mathcal{D}}(f_{\mathcal{D}}) \geq 0$ have already been computed. It holds that $\underline{V}^{\mathcal{C}}(f_{\mathcal{C}})$ only makes use of child lower bounds $\underline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ and $\overline{V}^{\mathcal{C}}(f_{\mathcal{C}})$ only makes use of child upper bounds $\overline{V}^{\mathcal{D}}(f_{\mathcal{D}})$.*

Proof. Since Theorem 5 is proven in the previous section, we only need to prove that computing $\underline{B}_j^{\mathcal{C}}$ and $\overline{B}_j^{\mathcal{C}}$ for $j = 1, \ldots, k$ can be done using only the lower bounds or upper bounds respectively.

First of all, we assume that for all $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$ the values $\underline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ and $\overline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ are non-negative. The initial step is trivial, since $\underline{B}_1^{\mathcal{C}} = \underline{V}^{\mathcal{D}_1}(f_{\mathcal{D}_1})$ and $\overline{B}_1^{\mathcal{C}} = \overline{V}^{\mathcal{D}_1}(f_{\mathcal{D}_1})$ and thus only lower bound values and upper bound values are used respectively, also being non-negative.

Next, we know that $\underline{B}_j^{\mathcal{C}}$ and $\overline{B}_j^{\mathcal{C}}$ are computed as follows in the algorithm:

$$\underline{B}_j^{\mathcal{C}} = \min\{\underline{B}_{j-1}^{\mathcal{C}}\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \underline{B}_{j-1}^{\mathcal{C}}\overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \overline{B}_{j-1}^{\mathcal{C}}\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \overline{B}_{j-1}^{\mathcal{C}}\overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j})\} \tag{4.23}$$

$$\overline{B}_j^{\mathcal{C}} = \max\{\underline{B}_{j-1}^{\mathcal{C}}\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \underline{B}_{j-1}^{\mathcal{C}}\overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \overline{B}_{j-1}^{\mathcal{C}}\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}),\ \overline{B}_{j-1}^{\mathcal{C}}\overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j})\} \tag{4.24}$$

We know that each value in the min and max operators is non-negative, while each of the four multiplications also results in a non-negative value. Furthermore, we know:

$$\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}) \leq \overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}) \qquad \text{and} \qquad \underline{B}_{j-1}^{\mathcal{C}} \leq \overline{B}_{j-1}^{\mathcal{C}} \tag{4.25}$$

Combining these equalities with the equation above results in a simplified version to computing $\underline{B}_j^{\mathcal{C}}$ and $\overline{B}_j^{\mathcal{C}}$:

$$\underline{B}_j^{\mathcal{C}} = \underline{B}_{j-1}^{\mathcal{C}}\underline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}) \tag{4.26}$$

$$\overline{B}_j^{\mathcal{C}} = \overline{B}_{j-1}^{\mathcal{C}}\overline{V}^{\mathcal{D}_j}(f_{\mathcal{D}_j}) \tag{4.27}$$

This proves that $\underline{V}^{\mathcal{C}}(f_{\mathcal{C}})$ only makes use of child lower bounds $\underline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ and $\overline{V}^{\mathcal{C}}(f_{\mathcal{C}})$ only makes use of child upper bounds $\overline{V}^{\mathcal{D}}(f_{\mathcal{D}})$. □

4.5 Sum operation proof

The sum operation $\sum_{\mathcal{D}\in\text{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}}\mathcal{D}$ of a circuit $\mathcal{C}$ over the direct children $\text{ch}(\mathcal{C})$ will be proven in this section. Again, we assume that Theorem 3 holds for all child circuits. The sum operation requires a weight vector with size equal to the number of children in $\text{ch}(\mathcal{C})$. This weight vector is defined as $w_{\mathcal{C}}$ and is contained in the weight set $\mathbf{w}_{\mathcal{C}}$. We also assume that circuit $\mathcal{C}$ adheres to the smoothness constraint defined in Definition 11.

As Equation 4.8 minimizes or maximizes over the weights and the smoothness constraint holds for the circuit, we can make a distinction on whether all children of $\mathcal{C}$ are disjoint or some are overlapping, as explained in Definition 14.

4.5.1 Disjoint children

Theorem 8. *Let $\mathcal{C}$ be a valid disjoint sum-based circuit and $f_{\mathcal{C}}$ be a factorizing function. Assume that Theorem 3 holds for circuits $\mathcal{D} \in \text{ch}(\mathcal{C})$. I.e. the values $\underline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ and $\overline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ have already been computed. It holds that $\underline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \underline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})]$ and $\overline{V}^{\mathcal{C}}(f_{\mathcal{C}}) = \overline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})]$ for disjoint sum-based circuit $\mathcal{C}$ and can be computed in polynomial time.*

Proof. If the child circuits of $\mathcal{C}$ are disjoint, we know that each $\mathcal{D} \in \text{ch}(\mathcal{C})$ has a distinct non-overlapping set of weights $\mathbf{w}_{\mathcal{D}} \in \mathbf{w}_{\mathcal{C}}$. Furthermore, $w_{\mathcal{C}}$ is defined as the weight vector for circuit $\mathcal{C}$. Thus, the set of weights for circuit $\mathcal{C}$ is defined as follows: $\mathbf{w}_{\mathcal{C}} = \{w_{\mathcal{C}}\} \cup \{w_{\mathcal{S}} \mid \mathcal{S} \in \text{desc}(\mathcal{C})\}$ with $\mathbf{w}_{\mathcal{D}} \cap \mathbf{w}_{\mathcal{D}'} = \emptyset$ for each $\mathcal{D}, \mathcal{D}' \in \text{ch}(\mathcal{C})$ with $\mathcal{D} \neq \mathcal{D}'$.

$$\underline{\mathbb{E}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] = \min_{\mathbf{w}_{\mathcal{C}}} \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}[f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})] = \min_{\mathbf{w}_{\mathcal{C}}} \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[\sum_{\mathcal{D}\in\text{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})\right] \tag{4.28}$$

$$= \min_{\mathbf{w}_{\mathcal{C}}} \sum_{\mathcal{D}\in\text{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} \mathbb{E}_{\mathbf{w}_{\mathcal{D}}}[f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})] \tag{4.29}$$

$$= \min_{w_{\mathcal{C}}} \sum_{\mathcal{D}\in\text{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} \min_{\mathbf{w}_{\mathcal{D}}} \mathbb{E}_{\mathbf{w}_{\mathcal{D}}}[f_{\mathcal{D}}(\mathbf{X}_{\mathcal{D}})] \tag{4.30}$$

$$= \min_{w_{\mathcal{C}}} \sum_{\mathcal{D}\in\text{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} \underline{V}^{\mathcal{D}}(f_{\mathcal{D}}) \tag{4.31}$$

$$= \underline{V}^{\mathcal{C}}(f_{\mathcal{C}}) \tag{4.32}$$

Due to the linearity of expectation, we know that the summation over the weighted functions $f_{\mathcal{D}}$ for each $\mathcal{D} \in \text{ch}(\mathcal{C})$ can be taken outside of the expectation operation (4.29). Furthermore, as the assumption is made that all child circuits are disconnected, the set of weights for all children is disjoint as argued earlier. When the set of weights to be optimized is disjoint, we can divide the minimum operation over all individual child circuits (4.30). The assumption that Theorem 3 holds for child circuits $\mathcal{D} \in \text{ch}(\mathcal{C})$ is used to convert the expectation notation to the value notation (4.31). Using the fact that all child circuits are assumed to be valid and efficiently computable, we can prove Theorem 8. The proof for the maximum value is analogous.

The sum operation can be performed in polynomial time, as the minimum of a weighted sum can be modelled as a linear program, which are solvable in polynomial time. □

4.5.2 Positive overlapping children

Lemma 9. *Given a valid circuit $\mathcal{C}$ and a non-negative factorizing function $f_{\mathcal{C}}$ for which function $f_i \geq 0$ is a factor of $f_{\mathcal{C}}$, $\underline{V}^{\mathcal{C}}(f_{\mathcal{C}})$ and $\overline{V}^{\mathcal{C}}(f_{\mathcal{C}})$ are positively valued and only makes use of the descendants' lower bound values $\underline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ and $\overline{V}^{\mathcal{D}}(f_{\mathcal{D}})$ respectively, with $\mathcal{D} \in \text{desc}(\mathcal{C})$.*

Proof. Since all functions f_i in $f_\mathcal{C}$ are non-negative, the leaf circuits in $\mathcal{C}$ are positively valued without having descendants, thus proving Lemma 9 for leaf circuits.

Sum-based circuits already only make use of the lower bound values, as seen in Equation 4.6 of the imprecise expectation algorithm of Section 4.1. Since the weights of sum-based circuits are also non-negative, the weighted values for each child circuit are positive. Furthermore, the summation over these positive values is again positive, proving the lemma for sum-based circuits.

Product-based circuits can normally use both lower and upper bound values, but given that the child circuits are all positive, Theorem 7 states that positive product-based circuits only use lower bound child values for computing the lower bound. A product of positive values always results in a positive value, thus proving the lemma for product-based circuits and consequently proving it for all valid circuits $\mathcal{C}$.

The case for the upper bound values is analogous. □

Theorem 10. *Let $\mathcal{C}$ be a valid sum-based circuit and $f_\mathcal{C}$ be a non-negative factorizing function. Assume that Theorem 3 holds for circuits $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$. I.e. the values $\underline{V}^\mathcal{D}(f_\mathcal{D}) \geq 0$ and $\overline{V}^\mathcal{D}(f_\mathcal{D}) \geq 0$ are valid and known. It holds that $\underline{V}^\mathcal{C}(f_\mathcal{C}) = \underline{\mathbb{E}}[f_\mathcal{C}(\mathbf{X}_\mathcal{C})]$ and $\overline{V}^\mathcal{C}(f_\mathcal{C}) = \overline{\mathbb{E}}[f_\mathcal{C}(\mathbf{X}_\mathcal{C})]$ for positive overlapping sum-based circuit $\mathcal{C}$ and can be computed in polynomial time.*

Proof. When the child circuits of a sum-based circuit $\mathcal{C}$ are overlapping, as in Definition 14, we need further assumptions to prove different cases. As otherwise the example from Figure 1.2 is a valid counterexample for the proof. An assumption to make is that all individual functions f_i in $f_\mathcal{C}$ result in a positive value. The proof will be given for the lower bound value, as the upper bound is analogous. We already know that the sum operation can be performed in polynomial time, as this has been shown in the previous proof.

First, the sets of weights to be optimized will be defined. Again, $w_\mathcal{C}$ is defined as the weight vector for circuit $\mathcal{C}$ and $\mathbf{w}_\mathcal{D}$ denotes the set of weights for child circuit $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$. $\mathbf{w}_\mathcal{C}$ is thus again defined as $\mathbf{w}_\mathcal{C} = \{w_\mathcal{C}\} \cup \{w_\mathcal{S} \mid \mathcal{S} \in \mathrm{desc}(\mathcal{C})\}$. However, as there are child circuits $\mathcal{D}, \mathcal{D}' \in \mathrm{ch}(\mathcal{C})$ with $\mathcal{D} \neq \mathcal{D}'$ that do overlap, we can define the set of circuits B causing this overlap as $B = \mathrm{desc}(\mathcal{D}) \cap \mathrm{desc}(\mathcal{D}')$. Consequently, the weight sets for circuits $\mathcal{B} \in B$ are defined as $\mathbf{w}_\mathcal{B} = \mathbf{w}_\mathcal{D} \cap \mathbf{w}_{\mathcal{D}'}$.

We can prove Theorem 10 by proving the following:

$$\underline{\mathbb{E}}[f_\mathcal{C}(\mathbf{X}_\mathcal{C})] = \min_{\mathbf{w}_\mathcal{C}} \sum_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} \mathbb{E}_{\mathbf{w}_\mathcal{D}}\left[f_\mathcal{D}(\mathbf{X}_\mathcal{D})\right] = \min_{w_\mathcal{C}} \sum_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} \min_{\mathbf{w}_\mathcal{D}} \mathbb{E}_{\mathbf{w}_\mathcal{D}}\left[f_\mathcal{D}(\mathbf{X}_\mathcal{D})\right] = \underline{V}^\mathcal{C}(f_\mathcal{C}) \tag{4.33}$$

Where the weights used in $\mathbf{w}_\mathcal{C}$ for the lower bound expectation should be the same weights as in $w_\mathcal{C}$ and $\mathbf{w}_\mathcal{D}$ for $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$ for the lower bound value.

Since we know that $\mathcal{B} \in \mathrm{desc}(\mathcal{D})$ and $\mathcal{B} \in \mathrm{desc}(\mathcal{D}')$, we can conclude that $\mathcal{B} \in \mathrm{desc}(\mathcal{C})$. Furthermore, as we assumed that $f_\mathcal{C}$ is a non-negative factorizing function with $f_i \geq 0$, we can use Lemma 9 to determine only lower bound values $\underline{V}^\mathcal{S}(f_\mathcal{S})$ are used in the descendant circuits $\mathcal{S} \in \mathrm{desc}(\mathcal{C})$. Since $\mathcal{B}$ is a descendant of $\mathcal{D}$, $\mathcal{D}'$ and $\mathcal{C}$, we know all three circuits must use the lower bound value $\underline{V}^\mathcal{B}(f_\mathcal{B})$. Hence, circuits $\mathcal{D}$, $\mathcal{D}'$ and $\mathcal{C}$ use the same value of $\mathcal{B}$ using the same weight set $\mathbf{w}_\mathcal{B}$.

Since the above holds for all circuits $\mathcal{B} \in B$, and since circuits not in an overlap set have no overlapping weights and cannot have conflicts, we can conclude that the weight values in the set $\mathbf{w}_\mathcal{D}$ are contained in $\mathbf{w}_\mathcal{C}$ for each $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$. Hence, the weight set $\mathbf{w}_\mathcal{C}$ can be divided over the minimum operator into $w_\mathcal{C}$ and $\mathbf{w}_\mathcal{D}$ for $\mathcal{D} \in \mathrm{ch}(\mathcal{C})$.

Thus, it follows that Theorem 10 holds for positive overlapping sum-based circuits $\mathcal{C}$. □

4.6 Combining proofs

In this section, the theorems of the previous sections will be combined to prove various applications on different structures and factorizing functions, making use of the imprecise expectation algorithm of Section 4.1. The first section will prove the non-negative factorizing function on any

circuit structure, as used for an imprecise likelihood computation. The imprecise conditional expectation with a single query variable is proven for any general focused query function. Next, credal classification is proven by making use of the imprecise conditional expectation proof. There are different applications possible which utilize the fact that the values at the leaf nodes are determined using factorizing functions. One of such applications is soft evidence, which will be discussed in Section 4.6.5. Finally, conditional expectation with multiple query variables is proven in the final section.

4.6.1 Non-negative factorizing function on DAG structures

The correctness of the upper and lower bound likelihood computation is already given by Mauá et al. [15], but in this section the proof will be given more generally by making use of the non-negative factorizing functions of Definition 2 and the modular theorems of the previous sections.

Theorem 11. *Let $\mathcal{C}$ be any valid circuit and $f_{\mathcal{C}}$ be a non-negative factorizing function with function $f_i \geq 0$ as a factor of $f_{\mathcal{C}}$, where $f_{\mathcal{C}}$ is computed in constant time. It holds that the algorithm described in Section 4.1 computes the lower and upper expectations over function $f_{\mathcal{C}}$, in $O(|\mathcal{C}|L)$ time, where $|\mathcal{C}|$ is the number of nodes under circuit $\mathcal{C}$, and L is an upper bound on the cost of solving a linear program of the form $\min_{w_{\mathcal{C}} \in \mathcal{Q}} \sum_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} \underline{V}^{\mathcal{D}}(f_{\mathcal{D}})$, polynomial in the number of children $|\mathrm{ch}(\mathcal{C})|$.*

Proof. In order to prove Theorem 11, we will consider the lower bound, as the upper bound is analogous.

Using Theorem 4 on leaf circuits, we know that $\underline{\mathbb{E}}[f_{\mathcal{L}}(\mathbf{X}_{\mathcal{L}})] = \underline{V}^{\mathcal{L}}(f_{\mathcal{L}})$ for leaf circuit $\mathcal{L}$. Since $f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = \prod_{\mathcal{L} \in \mathrm{lv}(\mathcal{C})} f_{\mathcal{L}}(X_{\mathcal{L}})$ with $f_{\mathcal{L}}(X_{\mathcal{L}}) \geq 0$ we know that the lower bound expectation can be computed efficiently in leaf circuits and is positive.

As we know that all functions f_i in $f_{\mathcal{C}}$ are positive, we can make use of Theorems 5, 7 and 10 to show that the lower bound expectations in any product or sum circuit can be computed correctly, given correct child circuit values. Theorems 5 and 7 for the product circuits do not impose any restrictions on the structure of the circuit. The same holds for sum circuits, as long as the child values are positive, using Theorem 10.

By combining Theorem 7, Lemma 9 and the assumption that all f_i are positive, we know that the value of each circuit in $\mathcal{C}$ is positive. Hence, no restrictions on the structure are imposed, apart from it being valid, making the imprecise expectation algorithm result in correct values.

For a product operation, the computation explained in Section 4.1 can be simplified to a product over the lower bound child values, for the lower bound product value. The time required is proportional to the number of child circuits $|\mathrm{ch}(\mathcal{C})|$. The sum operation solves a linear program over all its child circuits $\mathrm{ch}(\mathcal{C})$, taking $O(L)$ time. Since the value of each circuit needs to be computed at most once, at most $O(|\mathcal{C}|)$ computations of $O(L)$ are performed, resulting in a running time of $O(|\mathcal{C}|L)$. □

As already proven by Mauá et al. [15], the linear program of the sum operation can be solved in $O(k \log k)$ time, where k is the number of child circuits. Since $k \leq |\mathcal{C}|$, the overall running time becomes $O(|\mathcal{C}|^2 \log |\mathcal{C}|)$ with $|\mathcal{C}|$ being the number of nodes in circuit $\mathcal{C}$.

Imprecise likelihood

Theorem 12. *Let $\mathcal{C}$ be any valid circuit and $f_{\mathcal{C}}$ be a factorizing function with function $f_i(X_i) = I_{x_i}(X_i)$ as a factor of $f_{\mathcal{C}}$. It holds that the algorithm described in Section 4.1 computes the lower and upper bound likelihood values over function $f_{\mathcal{C}}$, in $O(|\mathcal{C}|L)$ time, where $|\mathcal{C}|$ is the number of nodes under circuit $\mathcal{C}$, and L is an upper bound on the cost of solving a linear program of the form $\min_{w_{\mathcal{C}} \in \mathcal{Q}} \sum_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} \underline{V}^{\mathcal{D}}(f_{\mathcal{D}})$, polynomial in the number of children $|\mathrm{ch}(\mathcal{C})|$.*

Proof. Since the factorizing function is defined as the function over the indicator functions, $f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = \prod_{i=1}^{|\mathrm{lv}(\mathcal{C})|} I_{x_i}(X_i)$ and each indicator function is a non-negative value $I_{x_i}(X_i) \geq 0$, the factorizing function is non-negative as per Definition 2.

Using Theorem 11, the lower and upper bound likelihood values are correct and computed efficiently. □

4.6.2 General factorizing functions on tree structures

In this section, the efficient computation of the imprecise expectation of a general factorizing function on tree structures is proven. Definition 1 describes how general factorizing functions are constructed.

Theorem 13. *Let $\mathcal{C}$ be any valid tree-structured circuit and $f_{\mathcal{C}}$ be a general factorizing function over variables $\mathbf{X}_{\mathcal{C}} = \{X_1, \ldots, X_n\}$, where $f_{\mathcal{C}}$ is computed in constant time. It holds that the algorithm described in Section 4.1 computes the lower and upper expectations over function $f_{\mathcal{C}}$, in $O(|\mathcal{C}|L)$ time, where $|\mathcal{C}|$ is the number of nodes under circuit $\mathcal{C}$, and L is an upper bound on the cost of solving a linear program of the form $\min_{w_{\mathcal{C}} \in \mathcal{Q}} \sum_{\mathcal{D} \in \mathrm{ch}(\mathcal{C})} w_{\mathcal{C}\mathcal{D}} \underline{V}^{\mathcal{D}}(f_{\mathcal{D}})$, polynomial in the number of children $|\,\mathrm{ch}(\mathcal{C})|$.*

Proof. In this proof, only the lower bound is considered, as the upper bound is analogous.

Let $f_{\mathcal{C}}$ be a general factorizing function of the form

$$f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = \prod_{i=1}^{n} f_i(X_i) \tag{4.34}$$

The proof structure for this theorem is similar to the previous proof, as it makes use of the modular theorems described in this chapter. Trivially, the values computed in the leaf circuits are correct and computed efficiently, as shown in Theorem 4. These can be both positive or negative.

Next, the computations in product-based circuits are correct and efficient, as shown in Theorem 5. This theorem only requires that the child circuit values can be computed efficiently, so any valid general factorizing function will work for a product-based circuit.

As we know that the structure of circuit $\mathcal{C}$ is a tree, sum-based circuits cannot have overlapping child circuits. Therefore, Theorem 8 on sum-based circuits with disjoint children covers all possible cases for sum circuits in tree structures. Again, this theorem only requires that the child circuit values can be computed efficiently.

So, by combining Theorems 4, 5 and 8 with the recursive definition of tree-structured circuits, the imprecise expectation of a general factorizing function can be computed correctly on tree structures, using the algorithm described in Section 4.1.

The time required for the greedy algorithm of product-based circuits is linear in the number of child circuits k, assuming all child values have already been computed. As the time required for solving a linear program in sum-based circuits is already $O(k \log k)$ in the number of children k, the greedy algorithm for product-based circuits does not increase the overall time complexity. Hence, the time required is again equal to $O(|C|L)$. □

4.6.3 Imprecision on general focused query functions

Next, the correctness of the conditional expectation algorithm will be given. For tree structures, Mauá et al. have already proven the correctness [15], but for certain DAG structures the algorithm should still hold. The DAG structure which will be included in the proof is one where no cycle edge is part of the paths from root to query variable leaves. A cycle edge is an edge between two cycle nodes, as defined in Definition 16. An example of such a structure is a class-discriminative structure explained in Section 3.4.3.

In order to prove correctness, two theorems are created for the following structures:

1. Tree structures

2. DAG structures, where no cycle edge is part of the paths from root to query variable leaves

Additionally, the following lemma is defined for computing the imprecise conditional expectation using binary search:

Lemma 14. *Let $\mathcal{C}$ be any valid circuit and $f_{\mathcal{C}}$ be a general focused query function over focus variable X_q and evidence variables $\mathbf{X}_e = \{X_1, \ldots, X_n\}$. I.e. $f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = f_q(X_q) \prod_{i=1}^{n} f_i(X_i)$ with $f_i(X_i) \geq 0$. Furthermore, assume that the algorithm described in Section 4.1 computes the correct lower and upper bound expectations of any general focused query function efficiently. It then holds that the lower and upper conditional expectation of f_q can be computed efficiently.*

Proof. The lower bound of the conditional expectation of $f_q(X_q)$ given some evidence $\mathbf{x}_e \in \mathbf{val}(\mathbf{X}_e)$ can be written as follows:

$$\underline{\mathbb{E}}\left[f_q(X_q) \mid \mathbf{x}_e\right] = \min_{\mathbf{w}_{\mathcal{C}}} \mathbb{E}_{\mathbf{w}_{\mathcal{C}}}\left[f_q(X_q) \mid \mathbf{x}_e\right] \tag{4.35}$$

which can also be written as:

$$\underline{\mathbb{E}}\left[[f_q(X_q) - \mu] \prod_{i=1}^{n} f_i(X_i)\right] = 0 \tag{4.36}$$

where μ equals the lower bound conditional expectation of $f_q(X_q)$, as expressed in Equation 4.35, when the above equation equals 0. Let $g_q(X_q) = f_q(X_q) - \mu$, resulting in the following factorizing function:

$$g_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = g_q(X_q) \prod_{i=1}^{n} f_i(X_i) \tag{4.37}$$

Assuming that all values are rational and that the lower bound expectation of $g_{\mathcal{C}}$ can be computed efficiently allows for μ to be computed efficiently using binary search in the interval $[\min_{x_q \in X_q} f_q(x_q), \max_{x_q \in X_q} f_q(x_q)]$.

We can now compute the lower bound expectation of f_q using μ in the binary search step to find the correct value for μ.

The case for the upper bound is analogous. □

Tree structures

Theorem 15. *Let $\mathcal{C}$ be any valid tree-structured circuit and $f_{\mathcal{C}}$ be a general focused query function with focus variable X_q and evidence variables $\{X_1, \ldots, X_n\}$. I.e. $f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = f_q(X_q) \prod_{i=1}^{n} f_i(X_i)$ with $f_i(X_i) \geq 0$. It holds that the algorithm described in Section 4.1 can be used to compute the lower and upper bounds of the imprecise conditional expectation of function $f_q(X_q)$ given non-negative factors for all other variables, in polynomial time.*

Proof. We will again only consider the lower bound, as the upper bound is analogous.

Let $g_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})$ denote a general focused query function constructed from $f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})$ with $g_q(X_q) = f_q(X_q) - \mu$ as follows:

$$g_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = g_q(X_q) \prod_{i=1}^{n} f_i(X_i) \tag{4.38}$$

Assume that the structure of $\mathcal{C}$ is a tree structure and let $\underline{V}^{\mathcal{C}}(g_{\mathcal{C}})$ and $\overline{V}^{\mathcal{C}}(g_{\mathcal{C}})$ denote the expectations of $g_{\mathcal{C}}$ at circuit $\mathcal{C}$ for this proof. Using Theorem 4 the leaf circuits already are assigned the correct expectation value, being either $\mathbb{E}[f_i(X_i)]$ or $\mathbb{E}[g_q(X_q)]$.

As we assumed that the structure is a tree, we can combine Theorems 5 and 8 to prove correctness of the lower bound expectation for a tree structure.

Theorem 5 is applicable in this situation, since it does not impose any constraints on the structure or value of the leaf circuits. Theorem 8 can be used, since it only requires the child circuits of a sum-based circuit to be disjoint, which is the case in tree structures.

Using Lemma 14, the lower bound expectation over $g_{\mathcal{C}}$ can be used to find the correct μ using binary search. μ equals the lower conditional expectation of f_q.

Hence, by the recursive definition of the circuits and by the modular theorems, conditional expectation can be computed correctly.

The running time of the algorithm is polynomial, since the computations of all circuits can be performed bottom up, where all values are propagated to the parent circuits. Furthermore, the running time for the computation in each circuit is polynomial, as discussed in each proof individually. □

DAG structures

In order to prove the theorem for DAG structures, we first need to prove the following lemma about the paths from root to query variable leaves.

Lemma 16. *Given a circuit $\mathcal{C}$ with some query variable X_q. It holds that the set of circuits in* $\text{desc}(\mathcal{C})$ *that include the query variable in their scope is exactly equal to the set of circuits on the paths from root to query variable leaves. I.e. $\{\mathcal{C}' \mid \mathcal{C}' \in \text{desc}(\mathcal{C}) \wedge X_q \in \text{sc}(\mathcal{C}')\} = \{\mathcal{C}' \mid \mathcal{C}' \in P_q\}$ with P_q being the set of circuits on any path from the root circuit $\mathcal{C}$ to query variable leaf circuits.*

Proof. In order to prove that both sets are equal, we will prove that both sets are a subset of each other. Let us first prove the following:

$$\{\mathcal{C}' \mid \mathcal{C}' \in \text{desc}(\mathcal{C}) \wedge X_q \in \text{sc}(\mathcal{C}')\} \subseteq \{\mathcal{C}' \mid \mathcal{C}' \in P_q\}, \tag{4.39}$$

which is proven by contradiction, by assuming the opposite:

$$\{\mathcal{C}' \mid \mathcal{C}' \in \text{desc}(\mathcal{C}) \wedge X_q \in \text{sc}(\mathcal{C}')\} \supset \{\mathcal{C}' \mid \mathcal{C}' \in P_q\} \tag{4.40}$$

In order for this to be true, there must be a circuit $\mathcal{B} \in \text{desc}(\mathcal{C})$ with $X_q \in \text{sc}(\mathcal{B})$, which is not in P_q. However, in order for $\mathcal{B}$ to include X_q in its scope, one of its children has to also include X_q in their scope. This can be repeated until a leaf circuit $\mathcal{L}$ is encountered that includes X_q in its scope. As $\mathcal{B} \in \text{desc}(\mathcal{C})$, circuit $\mathcal{B}$ must be connected to the root circuit $\mathcal{C}$. Hence, there exists a path from root circuit $\mathcal{C}$ to the query variable leaf circuit $\mathcal{L}$ through circuit $\mathcal{B}$. Hence, $\mathcal{B}$ must be included in P_q which contradicts the assumption made above, proving one direction of the equation.

The opposite direction of the equation is written down as follows:

$$\{\mathcal{C}' \mid \mathcal{C}' \in \text{desc}(\mathcal{C}) \wedge X_q \in \text{sc}(\mathcal{C}')\} \supseteq \{\mathcal{C}' \mid \mathcal{C}' \in P_q\}, \tag{4.41}$$

which we will prove again by a contradiction and assuming the opposite:

$$\{\mathcal{C}' \mid \mathcal{C}' \in \text{desc}(\mathcal{C}) \wedge X_q \in \text{sc}(\mathcal{C}')\} \subset \{\mathcal{C}' \mid \mathcal{C}' \in P_q\} \tag{4.42}$$

The assumed equation is true when there exists at least one circuit $\mathcal{B}$ on some path from the root circuit $\mathcal{C}$ to some query variable leaf circuit $\mathcal{L}$, where $\mathcal{B}$ is not an element of the set of circuits containing X_q in their scope.

This can however not be the case, as $X_q \in \text{sc}(\mathcal{L})$ and the scope of $\mathcal{L}$ is contained in the scope of every parent circuit. Hence, also being circuit $\mathcal{B}$, resulting in $X_q \in \text{sc}(\mathcal{B})$. Furthermore, we know that $\mathcal{B} \in \text{desc}(C)$, as there exists a path between the two circuits. Thus, this contradicts the assumption made above and combined with the earlier statement, Lemma 16 is proven to hold. □

Using Lemma 16 together with the Theorems 4, 5, 8 and 10, we can prove the DAG structure case of the following theorem:

Theorem 17. *Let $\mathcal{C}$ be a valid DAG-structured circuit where no cycle edge is part of the paths from root to query variable leaves and $f_{\mathcal{C}}$ be a general focused query function with focus variable X_q and evidence variables $\{X_1, \ldots, X_n\}$. I.e. $f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = f_q(X_q) \prod_{i=1}^{n} f_i(X_i)$ with $f_i(X_i) \geq 0$. It holds that the algorithm described in Section 4.1 can be used to compute the lower and upper bounds of the imprecise conditional expectation of function $f_q(X_q)$ given non-negative factors for all other variables, in polynomial time.*

Proof. Again, only the lower bound computation is considered, as the upper bound is analogous.

Let $g_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})$ denote a general focused query function constructed from $f_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}})$ with $g_q(X_q) = f_q(X_q) - \mu$ as follows:

$$g_{\mathcal{C}}(\mathbf{X}_{\mathcal{C}}) = g_q(X_q) \prod_{i=1}^{n} f_i(X_i) \tag{4.43}$$

Assume that the structure of $\mathcal{C}$ is a DAG structure, where no cycle edge is part of the paths from root to query variable leaves. $\underline{V}^{\mathcal{C}}(g_{\mathcal{C}})$ and $\overline{V}^{\mathcal{C}}(g_{\mathcal{C}})$ again denote the expectations of $g_{\mathcal{C}}$ at circuit $\mathcal{C}$ for this proof. The leaf circuits can again be proven by Theorem 4.

In order to prove the internal structure of the network, we will make a case distinction on the following two cases:

1. Circuits $\mathcal{C}' \in \text{desc}(\mathcal{C})$ that include the query variable in their scope ($X_q \in \text{sc}(\mathcal{C}')$)
2. Circuits $\mathcal{C}' \in \text{desc}(\mathcal{C})$ that do not include the query variable in their scope ($X_q \notin \text{sc}(\mathcal{C}')$)

Case 1:
In this case, we prove Theorem 17 for circuits that include the query variable in their scope. As we assumed that the paths (P_q) from root to query variable leaf circuits do not pass any cycle edges, we know that no circuit on these paths can be part of a cycle. Furthermore, we can use Lemma 16 to determine that the set of circuits on the paths P_q is exactly equal to the set of circuits in $\text{desc}(\mathcal{C})$ that include the query variable in their scope. Hence, we know that any circuit that includes the query variable in their scope cannot be part of a cycle and has no overlap in its child circuits.

Because of this, Theorem 8 which proves the sum operation for disjoint children is enough for this case, which does not impose any requirements on the sign of the circuit values. Theorem 5 is still applicable for the product circuits.

Case 2:
For all the remaining circuits without the query variable X_q in their scope, we know that all f_i must be positive, as assumed in the theorem. Hence, Theorem 10 for positive overlapping children of sum circuits can be used. Again, Theorem 5 is used to prove the product circuits without X_q in their scope.

Using Lemma 14, the lower bound expectation over $g_{\mathcal{C}}$ can be used to find the correct μ using binary search. μ equals the lower conditional expectation of f_q.

As each circuit in $\text{desc}(\mathcal{C})$ in the structure is covered by a proof, they together prove Theorem 17 recursively for the case of DAG structures, where no cycle edge is part of the paths from root to query variable leaves.

The running time for this structure is, like for the tree structures, polynomial in the number of nodes, since each circuit is evaluated bottom up. □

The Theorems 15 and 17 together show that imprecise conditional expectations can be computed efficiently for both tree structures and certain DAG structures.

The DAG structure proof covers the class-discriminative structure with any DAG structure as the class-specific structure, since this structure keeps the query variable leaves outside of the class-specific structure. Thus having no cycle edge part of the paths from root to query variable leaf circuits.

More exotic configurations of possible networks could possibly be constructed and proven, but the tree and limited DAG structures already cover many structure learning algorithms.

4.6.4 Credal classification

The next application to be proven for a credal probabilistic circuit is credal classification, as we might want to know the degree of robustness of classification results using a probabilistic circuit.

The degree of robustness can be measured by the ϵ value used in computing the weight vector perturbations $\mathcal{Q}_{w,\epsilon}$. The higher the ϵ-robustness value, the more variation in all weight vectors is allowed. The result of a CSPN is not a single value, but a lower and upper bound representing a probability interval.

As also discussed in the literature review, Chapter 2, there are different methods to classify data using these intervals [44]. The first is interval dominance, where one class dominates another class if the intervals do not overlap. Another method is credal dominance, where the credal sets are used to perform classification.

Say we have a circuit $\mathcal{C}$ with a class variable X_q and evidence variables $\mathbf{X}_e = \{X_1, \ldots, X_n\}$. The factorizing function $f_\mathcal{C}$ for credal classification is a general focused query function, expressed for classes $c_1, c_2 \in \mathbf{val}(X_q)$ as follows:

$$f_\mathcal{C}(\mathbf{X}_\mathcal{C}) = f_q(X_q) \prod_{i=1}^{n} f_i(X_i) \text{ with } f_i(X_i) = I_{x_i}(X_i) \text{ and } f_q(X_q) = I_{c_1}(X_q) - I_{c_2}(X_q) \tag{4.44}$$

with the non-class variable functions equal to the indicator functions and the class variable function equal to the difference between the class indicators. This means that $f_q(c_1) = 1$, $f_q(c_2) = -1$ and $f_q(X_q) = 0$ otherwise. Furthermore, $\mathcal{Q}$ is equal to the set of all possible perturbations on the circuit weight vectors. We say that class c_1 credally dominates class c_2 if:

$$\min_{\mathbf{w}_\mathcal{C} \in \mathcal{Q}} \mathbb{E}_{\mathbf{w}_\mathcal{C}}[f_\mathcal{C}(\mathbf{X}_\mathcal{C})] > 0 \tag{4.45}$$

as proposed by Zaffalon [44, Def. 4.1].

In general, deciding if class c_1 credally dominates another class c_2 is coNP-hard, as proven by Mauá et al. [15], if no further assumptions are made. Hence, Mauá et al. [15] require the assumptions that the number of classes is bounded. Another assumption by the authors is that the circuit structure is a tree. Next, a theorem and proof is given which relaxes the structural assumption.

Theorem 18. *Let $\mathcal{C}$ be any valid DAG-structured circuit where no cycle edge is part of the paths from root to query variable leaves, hence including tree structures. Let $f_\mathcal{C}$ be a general focused query function on query variable X_q and evidence variables $\mathbf{X}_e = \{X_1, \ldots, X_n\}$. Furthermore, assume that the number of classes is bounded. It holds that credal classification can be performed in polynomial time.*

Proof. Equation 4.45 can be converted to the computation of a conditional expectation with a single query variable, which is already proven in Theorems 15 and 17. These theorems cover the computation of the lower bound imprecise conditional expectation of a general focused query function on both tree and limited DAG structures respectively, in polynomial time. Hence, credal classification can be performed in polynomial time. □

The maximum ϵ-robustness value, for which the weight vector perturbations $\mathcal{Q}_{w,\epsilon}$ still result in Equation 4.45 being positive and a single class to be found, can be found by performing a binary search on the ϵ value in the interval $[0, 1]$.

4.6.5 Soft evidence

An application of the generalized imprecise expectation algorithm with factorized functions is to apply soft evidence at each leaf circuit function. This soft evidence can be used to model a given input with a tendency towards a specific value without being precise. Soft evidence can be applied on discrete variables by redistributing the likelihood of the observed value to all other values. An example is given in Figure 4.1. The case without soft evidence would result in leaves with value 2 getting a full likelihood of 1, while all leaves with other values than 2 would get a likelihood of 0.

In the experiments performed in Chapter 5, the DAG implementation in PyTorch is limited to only accept continuous variables as normal distributions for the features, along with a categorical

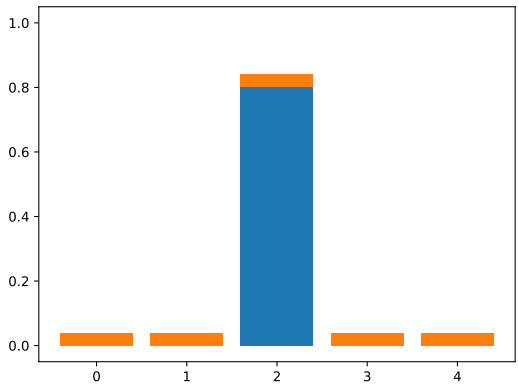


Figure 4.1: Soft evidence on a categorical distribution with a redistributing factor of 0.2. For an observed class of 2, the soft evidence distributes a likelihood of 0.2 over all classes.

variable for the unique classes. Therefore, a variation on soft evidence is used which can be applied on continuous variables. As opposed to using the likelihood value of a single observation, computed using the probability density function $\text{pdf}(obs, \mu, \sigma)$, an interval around the observation will be used. This interval is based on the standard deviation of the normal distribution multiplied by some soft evidence factor and results in having a minimum and maximum observation value.

The soft evidence factor is used to define the size of the soft evidence area as a factor of the standard deviation for both the lower and upper observation value.

The soft evidence value is then computed as the area under the normal distribution, between the minimum and maximum observations, which can be computed with the cumulative distribution function as follows:

$$v = \text{cdf}(obs_{min}, \mu, \sigma) - \text{cdf}(obs_{max}, \mu, \sigma) \tag{4.46}$$

By computing the factorized functions at the leaf circuits in this way, the issue of exact continuous observations having probability 0 is also solved. A visual representation of soft evidence on continuous variables is shown in Figure 4.2 where the value without soft evidence is illustrated with the line at observation 0.4 and with likelihood around 2.5 and the value with soft evidence is visualized with the area around the observation.

Corollary 19. *Let $\mathcal{C}$ be a valid circuit according to Theorem 18. Making use of soft evidence instead of the indicator functions for each function f_i in $f_\mathcal{C}$, credal classification can still be performed efficiently.*

Proof. Soft evidence can be computed for each function f_i in $f_\mathcal{C}$, as the soft evidence computation is univariate for each variable used in the circuit $\mathcal{C}$. Furthermore, the probability value as the result of the soft evidence can be used in the sum and product operations, as it resembles the expected value. □

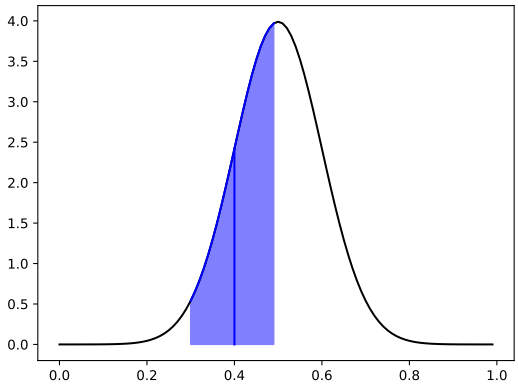


Figure 4.2: Continuous soft evidence on a normal distribution with mean 0.5, standard deviation 0.1 and soft evidence factor 1. For an observed value of 0.4, the soft evidence probability is the area under the curve from 0.3 to 0.5, which is roughly equal to 0.477.

Chapter 5

Experiments

This chapter will describe the performed experiments where the imprecise expectation algorithm of Section 4.1 is applied on certain DAG structures. In Section 5.1 class-discriminative DAG structures are compared to tree structures for a classification task, making use of the credal classification approach. Thus, showing with empirical data that computing robustness for certain DAG structures is possible. Section 5.2 contains experiments utilizing the generalization of the factorizing functions. For each leaf distribution, a variation of soft evidence on continuous variables is applied. Hence, this experiment shows a possible application of being able to compute the imprecise expectation of more general factorizing functions in a probabilistic circuit.

5.1 Comparing credal tree to credal DAG structures

5.1.1 Approach

In order to show a practical example of the imprecise expectation algorithm on certain DAG structures a comparison with tree structures is made. This experiment compares the number of correctly classified samples with the total number of samples, also called the accuracy, for both the credal tree and credal DAG structures. The accuracy is computed over different ϵ-robustness values, as introduced in Section 4.6.4, where samples that classify as a set of classes under the ϵ-robustness value are filtered out and not used in computing the accuracy. The described comparison is made on different samples from the dataset as used for training the structures in order to keep the network unbiased. Training is performed using the training set, while evaluating the experiment is done using the test set.

The task of the networks is to perform credal classification as described in Section 4.6.4. This task is performed on different datasets, where the robustness value is used to filter out any samples with a robustness value lower than some threshold. I.e. The classification results of the filtered out samples are determined to be unreliable. The accuracy at some threshold is computed using the remaining samples. The procedure used to perform credal classification makes use of binary search in order to determine, for each data sample, the largest robustness value which still results in a single class. The exact procedure is described in Section 4.1.

The tree and DAG structures which have been used for this experiment are constructed using the LearnSPN and RAT-SPN algorithms described in Sections 3.4.1 and 3.4.2, in combination with a class-discriminative top structure described in Section 3.4.3. The following structures are constructed:

1. Tree structure constructed using the LearnSPN algorithm.

2. Class-discriminative tree structure constructed using the class-discriminative structure, making use of the LearnSPN algorithm for each class-specific circuit structure.

3. Class-discriminative DAG structure constructed using the class-discriminative structure, making use of the RAT-SPN algorithm for each class-specific circuit structure.

As discussed in Section 4.6.3, these types of structures should allow for efficient imprecise conditional expectation evaluation.

For the comparison to be fair, the size of the structures is changed in such a way that the number of parameters of the three networks is about equal. This way, the ϵ-contamination is performed equally often between the networks. All three networks make use of Gaussian distributions for the feature leaf nodes and categorical leaf nodes for the class variable.

Comparing the accuracy of the networks on their own filtered set of data samples should already show whether the algorithm works for DAG structures. However, in order to compare the performance of each network even better, the accuracies of the three structures can be evaluated using one and the same filtered sample set of one of the networks. I.e. the robustness value of each sample is computed by a single network, while the accuracy of the three networks is computed over the non-credal classification results of these samples filtered on the robustness value. By doing this, we can verify whether the type of structure is relevant for evaluating data or for filtering the dataset samples. We will also see whether the filtered sample set of one network can even be used to evaluate the performance of another network.

The datasets which have been used to compare the structures are a subset of the OpenML-CC18 benchmark [39] and can be found on their website. The datasets have a varying number of samples and variables and consist of continuous values for all variables except the class. Table 5.1 lists the used datasets, along with some statistics. Each dataset is randomly shuffled and split in a training and test set using a 70/30 ratio. If the dataset is not normalized yet, z-score normalization is used to normalize the continuous data. The training set is used to generate and train the networks, while the test set is used to perform the experiments.

Dataset	Training set	Testing set	Nr. variables	Nr. classes
Authent	960	412	5	2
Diabetes	537	231	9	2
Gesture	6911	2962	33	5
Robot	3819	1637	25	4
Texture	3849	1651	41	11

Table 5.1: The used OpenML-CC18 datasets for the experiment

5.1.2 Implementation

The experiment has been performed using the SPFlow library [25] which provides an SPN framework with various existing algorithms implemented, like LearnSPN. In order to allow for credal classification, the robustness evaluation used by Correia et al. [7] is transferred to the SPFlow library. This enables for robust inference and robust classification to be performed. The robustness value is searched for using binary search with each iteration evaluating the SPN bottom up.

Tree structure generation

Both of the tree structures can be generated using the LearnSPN algorithm of SPFlow, discussed in Section 3.4.1. The LearnSPN algorithm used makes use of randomized dependence coefficients [22] (RDC) in order to determine whether the features can be clustered or whether the dataset needs to be clustered. The latter will be clustered using K-means clustering. RDC requires a threshold to determine the type of clustering. For the performed experiments, the default threshold of 0.3 is used. As a last tuning parameter, the minimum number of data samples required for clustering can be set. The default value for this parameter is 200, but depending on the dataset this value might be varied, as each dataset has a different number of samples. The class-discriminative LearnSPN

structure has no extra parameters and might even be generated by the standard LearnSPN algorithm in some cases.

DAG structure generation

The class-discriminative DAG structure is generated and trained in PyTorch using a layerwise RAT-SPN implementation present in SPFlow [25] and explained in Section 3.4.2. As the PyTorch implementation is completely separate from the object-oriented SPN implementation used for robustness evaluation, a conversion script has been created for copying over the structure, weights and leaf variables after training the network. The RAT-SPN parameters are trained using gradient descent with the Adam optimizer, run for 60 epochs and a batch size of 100 samples. These hyperparameters are chosen by manually comparing results of different settings and choosing the best performing ones. The loss function used for training and evaluation is the cross-entropy loss function on the weighted child values of the root node, suitable for classification. The learning rate is set to 0.01 for most datasets, except Robot, which uses a learning rate of 0.1 as 0.01 resulted in limited improvements over the epochs.

In order to approach the number of parameters of the tree structures, several hyperparameters of the RAT-SPN algorithm can be tuned. For most datasets, the following hyperparameters are used:

- Repetitions: 1
- Depth: 2
- Number of leaf distributions: 1
- Number of sum nodes per layer: 2

this resulted in DAG structures with roughly the same number of parameters as the tree structures and a good graph representation of a RAT-SPN.

The Gesture dataset is larger, which results in more parameters in the tree structures. To match the number of parameters for this dataset, the number of leaf distributions is increased to 2 and the number of sum nodes per layer is set to 3. On the other hand, the Texture dataset already achieves a high accuracy with all hyperparameters set to 1, resulting in a tree structure.

5.1.3 Results

The results of the experiments for the different datasets are discussed in this section. For each dataset, the accuracy of the networks is evaluated over the ϵ-robustness values from 0 to 0.5. The accuracy is computed using only samples with a robustness value greater than the ϵ-robustness threshold. The accuracy over the ϵ-robustness threshold is plotted in a figure, where the accuracy is represented on the y-axis and the robustness threshold on the x-axis. The accuracy of the three different structures is plotted in the same figure using solid lines of different colors, in order to compare the performance. Alongside the accuracy, the fraction of used test samples is plotted using a dashed line, again using the same colors. Correia et al. [7] also use a similar figure to plot the relation between the accuracy and ϵ-robustness threshold.

As the datasets have different characteristics with respect to the number of features and the number of samples, the generated structures for the three approaches have different statistics. The relevant statistics like the number of nodes and number of parameters to be used with imprecision are shown in Table 5.2. As can be seen, the number of nodes in the DAG structure are lower compared to the tree structures, as the number of parameters is matched. Furthermore, the tree structures consist of much more leaf nodes, due to the nature of not being able to merge branches in the graph. In contrast, the DAG structure is able to reuse a single leaf node in multiple internal nodes.

Dataset	Structure	Nodes	Sum	Product	Leaf	Params	Eval. time (s)
Authent	Tree	33	3	8	22	8	21
	CD-Tree	40	5	11	24	11	27
	CD-DAG	43	11	22	10	18	25
Diabetes	Tree	29	2	5	22	5	11
	CD-Tree	39	3	7	29	6	14
	CD-DAG	51	11	22	18	18	18
Gesture	Tree	779	57	180	542	180	3478
	CD-Tree	924	55	162	707	162	4096
	CD-DAG	491	36	130	325	170	2017
Robot	Tree	752	5	34	713	34	2060
	CD-Tree	763	5	34	724	34	2145
	CD-DAG	165	21	44	100	36	440
Texture	Tree	935	12	33	890	33	2557
	CD-Tree	936	12	33	891	33	2550
	CD-DAG	507	12	44	451	22	1506

Table 5.2: Generated structure statistics for all datasets

Robot

Figure 5.1 shows a visualization as described above for the Robot dataset. With 25 variables and 1637 test samples it is one of the larger datasets used. Both the tree structures have 34 parameters and the DAG structure has 36 parameters. The starting accuracy at a threshold of 0 of the DAG structure is already better when compared to the tree structures. When the robustness threshold increases, the accuracy also improves, always staying above the tree structure accuracies. However, the DAG structure does filter out more samples, as seen by the dashed lines.

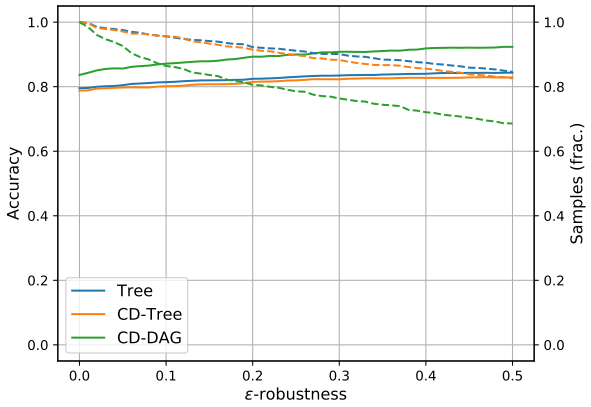
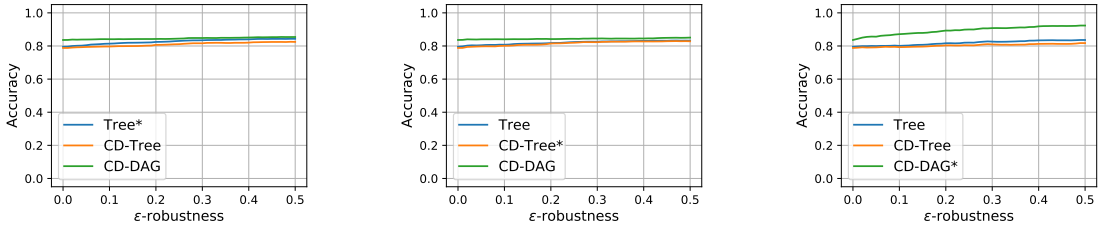


Figure 5.1: Structure comparison on Robot dataset

As explained in the experiment approach, Section 5.1.1, the networks are also compared using the fixed sample sets of each network. The accuracy of each network for the different sample sets is shown in Figure 5.2. These figures already show that the robustness values of the samples can be transferred from one network to the predictions of another network, while maintaining accuracy. With regards to the Robot dataset, the sample set of the DAG network does not increase the accuracy of the tree networks, Figure 5.2c, while the increased accuracy of the DAG network gets diminished by the tree structure sample sets, Figure 5.2a and 5.2b.



(a) Tree set (b) CD-Tree set (c) CD-DAG set

Figure 5.2: Robot dataset accuracies compared using the same filtered sample sets (The * denotes which filtered sample set is used)

Gesture

The Gesture dataset networks have the most number of parameters in their structures due to the dataset having 33 variables and 6911 training samples. From Figure 5.3 it can be seen that the starting accuracy is already higher for the DAG structure, but classification is still often incorrect. Increasing the robustness threshold shows that the higher accuracy of the DAG structure is maintained in comparison with the tree structures. However, the fraction of used samples for the DAG network quickly drops to a low value, resulting in using only a small subset (roughly 2.5%) of the 2962 train samples for the accuracy computation. Besides that filtering out many samples might make the network seem doubtful, the trust in the computed accuracy also drops.

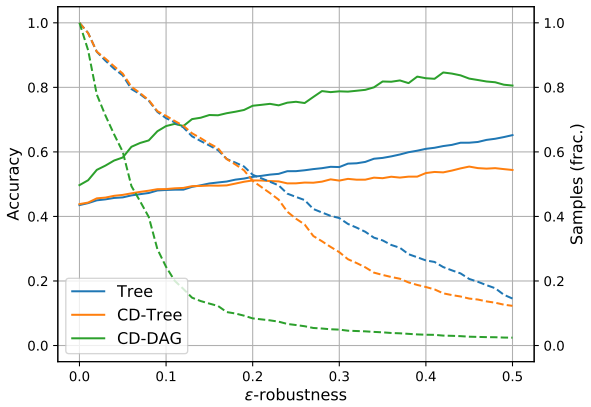
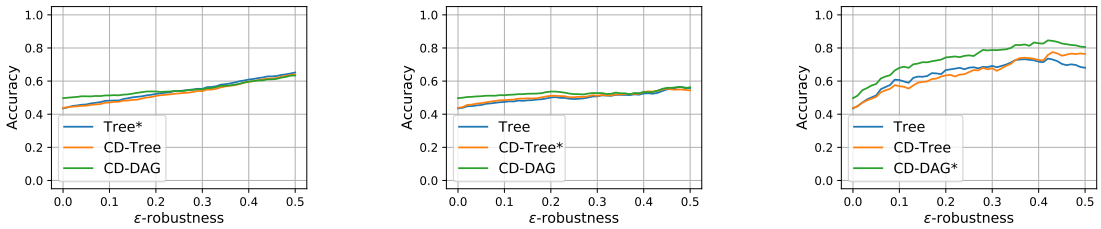


Figure 5.3: Structure comparison on Gesture dataset

Comparing the performance of the three networks using the three sample sets in Figure 5.4 shows that the starting accuracy is still different for the three networks, since a threshold of 0 acts like a regular SPN and is not influenced. As the robustness threshold increases, the networks do follow the trend of the native network. However, in Figure 5.4c it can be seen that the DAG network still performs best on the native DAG sample set.

Diabetes

Next, the results of the Diabetes dataset in Figure 5.5 are discussed. This dataset has 9 variables and a small test set size of 231 samples. The number of parameters for the tree, class-discriminative tree and DAG structures are 5, 6 and 18 respectively. Comparing the DAG structure to both tree structures shows that both the accuracy and the proportion of filtered samples is about equal. The



(a) Tree set (b) CD-Tree set (c) CD-DAG set

Figure 5.4: Gesture dataset accuracies compared using the same filtered sample sets

comparison on the three sample sets again shows a very similar performance between the three structures, which can be seen in Figure A.1 in the Appendix.

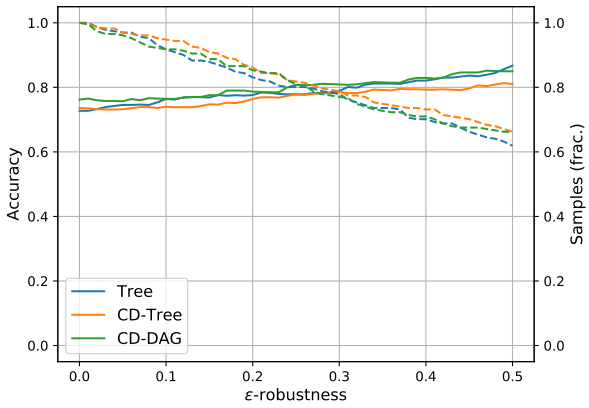


Figure 5.5: Structure comparison on Diabetes dataset

Authent

The Authent dataset is another small dataset with only 5 variables and a test set size of 412 samples. The generated structures thus are very small. Nevertheless, the accuracy of the three networks is very high, with an optimal accuracy for the DAG structure and a slight lower accuracy for both tree structures. Furthermore, the DAG structure only filters out a handful of samples, while the tree structures need to filter out more samples to increase their accuracy, as seen in Figure 5.6.

The individual results on the filtered sample sets from Figure 5.7 show that the optimal performance of the DAG structure is present for all sample sets. This can be explained, since the accuracy of the network with a robustness threshold of 0 is already 1. With the limited filtering of the DAG sample set, both tree structures do not achieve any improved accuracy, as seen in Figure 5.7c.

Texture

The results described for the Authent dataset are not necessarily achieved because of the small dataset size, since generated structures for the Texture dataset show similar results. The Texture dataset consists of 1651 test samples and 41 variables with 11 classes. Due to the large dataset size, only the RAT-SPN with the smallest structural hyperparameters came close to the tree structures.

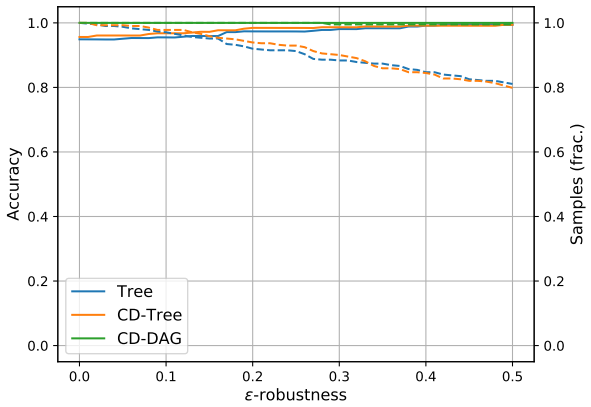
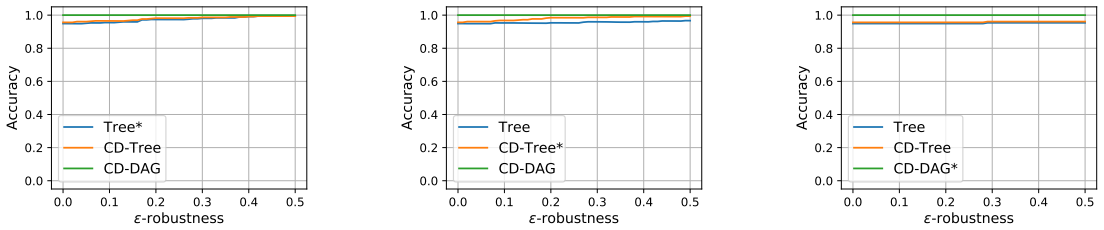


Figure 5.6: Structure comparison on Authent dataset



(a) Tree set (b) CD-Tree set (c) CD-DAG set

Figure 5.7: Authent dataset accuracies compared using the same filtered sample sets

This means that the generated structure using the RAT-SPN algorithm is a tree and not a DAG. nevertheless, the performance of this structure is optimal, while that of the tree structures is not. As the results are similar to the Authent dataset, they can be found in Appendix A.1 (Figure A.2 and A.3).

Evaluation time

A consequence of matching the number of parameters in the tree and DAG networks is that the DAG networks require fewer nodes compared to the tree networks. This is due to the network being able to reuse nodes as long as the structural requirements are met. In Table 5.2 the number of nodes for each structure is shown for every dataset. As already discussed in Chapter 4, the running time of the credal classification algorithm scales with the number of nodes in the network, as every node requires a computation which can be cached and propagated to all parent nodes. The last column in the table contains the robustness evaluation time, in seconds, for the different structures on the test samples. Especially the latter three larger datasets show a clear correlation between the number of nodes and the evaluation times.

5.1.4 Conclusion

In this section, conclusions of the performed experiments are made. First of all, the results empirically show that performing credal classification on class-discriminative DAG structures efficiently is possible, as also claimed by the earlier proof. All results show a positive correlation between the accuracy and the robustness threshold, meaning that the robustness value does indicate some form of confidence in the classification. With credal classification possible, it is not guaranteed

that results are comparable with tree structures. Therefore, comparisons with two tree structures have been made, showing that the accuracy of a DAG structure might exceed that of a tree structure, depending on the context and dataset. Both the Robot and Gesture datasets show that a higher accuracy is achievable at the cost of filtering out more samples. In contrast, the Diabetes dataset shows that the accuracy of the DAG structure can also be very similar to that of the tree structures. Lastly, the Authent and Texture dataset show that a DAG structure might be more frugal in filtering samples, when the accuracy is optimal. These results do show that there is a place for directed acyclic graphs in credal classification. However, further research on different tree and DAG structures and more datasets is needed to make these claims more concrete.

Another conclusion that can be drawn from the results is that the filtered sample set of one network can be used to evaluate the performance of another network, while maintaining a similar or higher accuracy. This observation is made between the different datasets, with the improved accuracy mostly occurring for the DAG structure. A possible application of this observation, to be researched in future work, is to use a filtered sample set constructed with some structure by credal classification on a different more complicated network that is incapable of performing robust classification.

Finally, the results show that the speed of the robustness evaluation can probably be improved by using DAG structures, as the number of nodes is reduced by reusing them for different parents. This makes the DAG structures run more efficient than tree structures with a comparable number of parameters. The reduced evaluation time can also be utilized to be able to create larger DAG structures with more parameters compared to tree structures.

5.2 Applying soft evidence

5.2.1 Approach

A possible application of the imprecise expectation algorithm with general factorizing functions is applying soft evidence, as explained in Section 4.6.5. These factorizing functions are more general, since a function is introduced for every random variable, compared to how a robustness analysis is performed by Mauá et al. [15] using a factorizing function with indicator functions for non-query variables. We have tested whether applying soft evidence on continuous variables helps with determining the degree of robustness of samples with the credal classification algorithm. In order to verify this hypothesis, we have compared the accuracy of the test set as a function over the ϵ-robustness values, as for the previous experiment. However, for this experiment different soft evidence factors or interval widths are compared with each other on a single structure. The algorithm for credal classification is the same as for the previous experiment, except for the leaf circuit values. These values are computed using the area under the normal distribution in the specified interval.

The structures for which this experiment has been conducted are again the two tree structures constructed with the LearnSPN algorithm in conjunction with the class-discriminative top structure. The DAG structure is constructed using the RAT-SPN algorithm, again in combination with the class-discriminative top structure. The structures are set-up using the same hyperparameters as for the experiment in Section 5.1, with all three structures making use of Gaussian distributions for the feature leaf nodes and categorical leaf nodes for the class variable.

The datasets used in this experiment are the same subset of the OpenML-CC18 [39] benchmark as for the previous experiment (See Table 5.1). Training and test sets are created using a 70/30 randomly shuffled split, with normalization of the continuous variables performed using z-score. The networks are generated and trained using the training set and evaluated using the test set.

5.2.2 Implementation

This experiment has again been performed by using the SPFlow library [25] in combination with the transferred robustness evaluation code used by Correia et al. [7]. In addition to this, the

functions evaluated at the leaf nodes incorporate an implementation of soft evidence for continuous variables as explained in Section 4.6.5. The value at these normally distributed leaf nodes is computed by first determining the minimum and maximum observation value using the standard deviation multiplied with the soft evidence factor. Next, the cumulative distribution function (cdf) is computed at the minimum and maximum observation using the leaf node mean and standard deviation. Finally, the minimum cdf value is subtracted from the maximum cdf value and the logarithm is computed.

The three structures to be evaluated individually have been generated using the same implementation and hyperparameters as explained in Section 5.1.2, so these will not be repeated here. The used soft evidence factors in most experiments is 0 for no soft evidence and 1, 2 and 3 as an offset factor multiplied with the standard deviation of the distribution. The interval size relative to the standard deviation between the minimum and maximum observation is double that of the factors described above, being 2, 4 and 6.

5.2.3 Results

The results of the performed experiments are discussed in this section, where for each dataset and structure, the accuracy of the test set is evaluated as a function over the ϵ-robustness values from 0 to 0.5. The computation of the accuracy using the robustness threshold and the meaning of the plots has been explained in Section 5.1.3, so this will not be repeated here. Similarly, the statistics of the generated structures are equal to Table 5.2 of the previous experiment. The way that the visualizations are altered, is that the accuracy is evaluated over the different interval sizes for soft evidence. This is done using the same trained network, so the only difference between the different curves is the interval size.

Diabetes

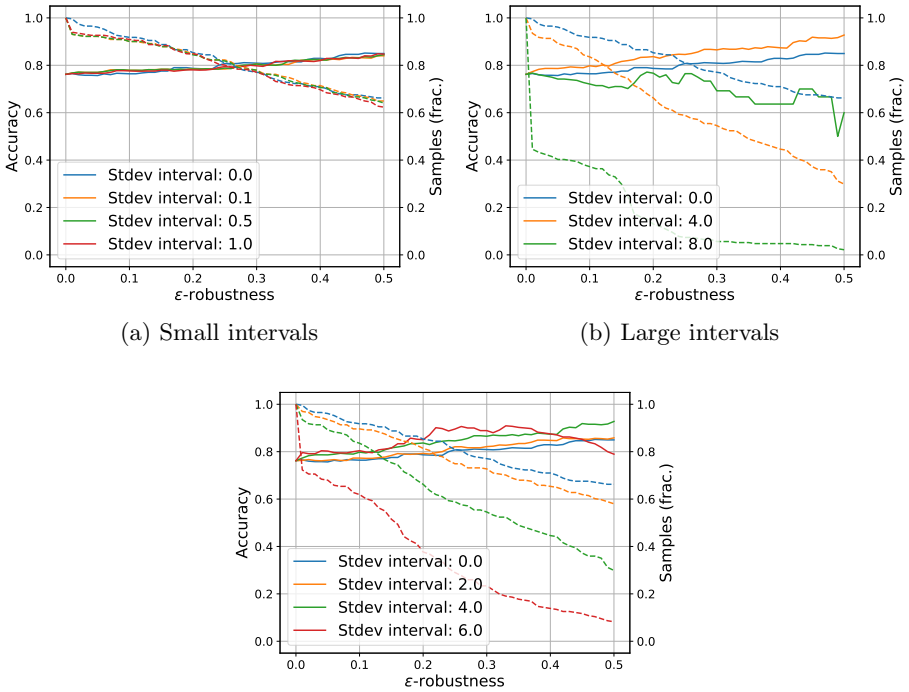
The first dataset for which the results will be discussed is the Diabetes dataset. For this dataset, we ran multiple instances of the experiment with different soft evidence interval sizes, ranging from 0.1 to 8.0. Figure 5.8 shows several plots with different intervals on the DAG structure. Figure 5.8a shows no significant changes in accuracy over the different smaller interval changes, while Figure 5.8b shows that an interval of 8 results in a lot of samples to be filtered out, making the accuracy unreliable. This result can be explained by the fact that the interval factor is multiplied with the standard deviation, as 4 standard deviations in both directions covers the distribution for most observations, there is little difference between two samples. Hence, for most datasets the intervals of 2, 4 and 6 are used, as shown in Figure 5.8c for the diabetes dataset.

Looking at the accuracies of the different intervals in Figure 5.8c, we can see that they are slightly higher for interval ranges 4 and 6 compared to the case without soft evidence, with interval range 6 dropping down as the robustness threshold get higher. We can also see that, as the soft evidence interval gets larger, more samples are filtered out.

Figures 5.9a and 5.9b show the results for the tree and class-discriminative tree structures. Both figures show that the initial accuracy for all soft evidence plots is higher compared to the regular accuracy, due to filtering out many samples. These samples must all have a low robustness value right above 0, as the accuracies for a robustness threshold of 0 are equal. The figure of the tree structure results shows that the interval accuracies quickly drop below the regular accuracy, while the figure of the class-discriminative tree structure results shows a higher accuracy maintained until a robustness threshold around 0.25, after which the accuracy also drops. For both structures, a lot more samples are filtered out compared to the regular accuracy.

Robot

The results of the soft evidence over the three structures on the robot dataset are shown in Figure 5.10. Both the tree and class-discriminative tree structures do show an almost fixed improvement in accuracy over the different robustness thresholds. Most notably, the increase in

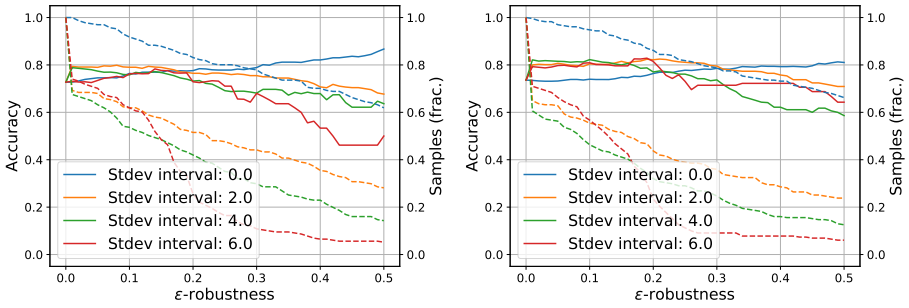


(a) Small intervals

(b) Large intervals

(c) Regular intervals

Figure 5.8: Comparing different choices in soft evidence intervals on the Diabetes dataset with a CD-DAG structure



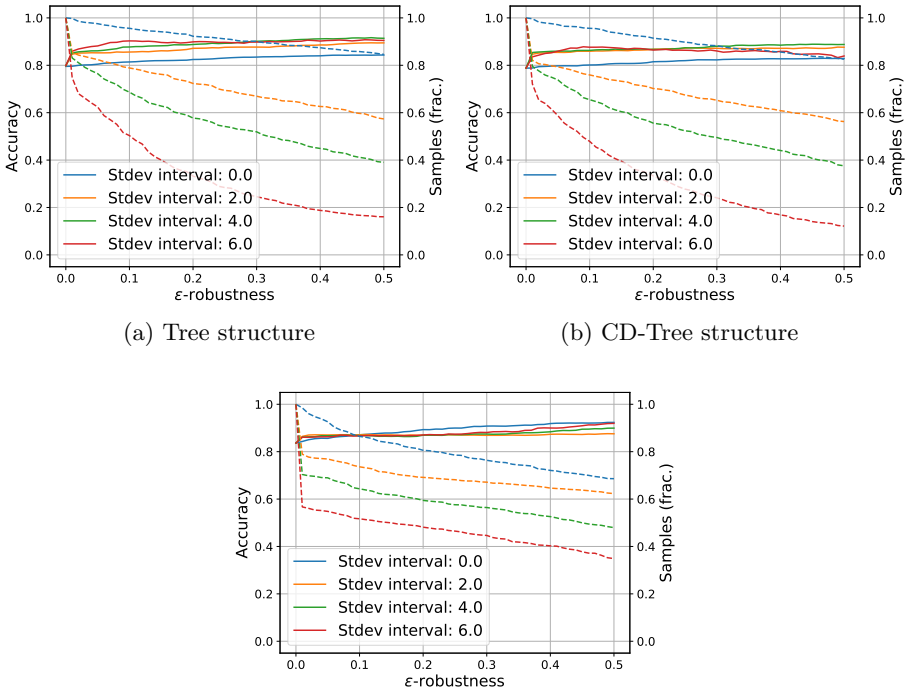
(a) Tree structure

(b) CD-Tree structure

Figure 5.9: Soft evidence results on the Diabetes dataset for the Tree and CD-Tree structures

accuracy does not seem to change a lot over the different soft evidence intervals, while the number of filtered samples does increase as the interval increases. The removed samples are thus not necessarily incorrectly classified.

However, for the class-discriminative DAG structure from Figure 5.10c we see no improvement in accuracy compared to the regular accuracy. The results for this DAG structure do again show that the removed samples with the higher interval ranges are not necessarily incorrectly classified.



(a) Tree structure

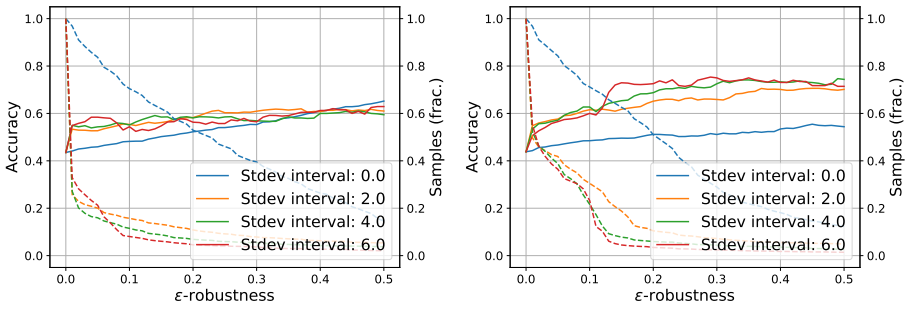
(b) CD-Tree structure

(c) CD-DAG

Figure 5.10: Soft evidence results on the Robot dataset for the Tree, CD-Tree and CD-DAG structures

Gesture

As for the Gesture dataset, we can see in Figure 5.11 that class-discriminative tree structure achieves an improvement in accuracy with soft evidence applied, while the regular tree structure only shows an initial improvement in accuracy which reduces as the robustness threshold increases. For both structures, the number of filtered out samples is rather high in comparison with the regular number of samples filtered.



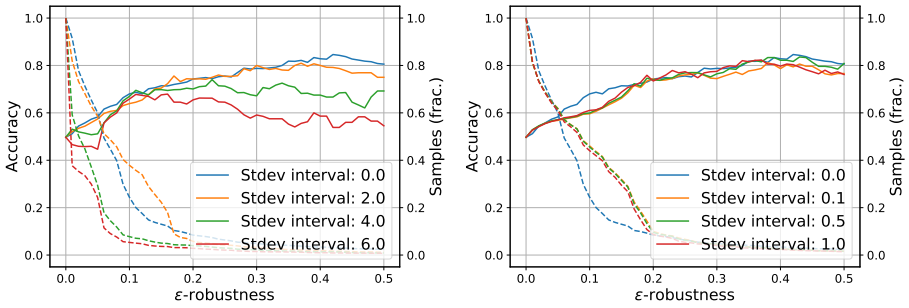
(a) Tree structure

(b) CD-Tree structure

Figure 5.11: Soft evidence results on the Gesture dataset for the Tree and CD-Tree structures

The class-discriminative DAG structure does not see any improvement to the accuracy and even has a lower accuracy where the fraction of filtered samples is approaching 0, as seen in Figure 5.12a. An interesting observation for the combination between the Gesture dataset and

the DAG structure can be seen in Figure 5.12b, where the results for the small soft evidence intervals are shown. Between the robustness thresholds of 0.05 and 0.2 the fraction of filtered samples for soft evidence experiments is greater than that of the regular experiment. Due to this, the accuracy of the soft evidence experiments also drops down. In this case, more incorrect than correct samples are kept in computing the accuracy. Possibly this is due to the connected nature of the DAG structure, but no definitive conclusion can currently be made.



(a) Regular intervals

(b) Small intervals

Figure 5.12: Soft evidence results on the Gesture dataset for the CD-DAG structure

Texture

The results for the Texture dataset from Figure 5.13 show that the soft evidence intervals do improve the accuracy for both tree structures, while filtering out more samples. In this case, clear improvements in the accuracy between the different intervals can be seen, where the larger intervals achieve a higher accuracy by filtering out more samples.

As the generated DAG structure for the Texture dataset achieves an optimal accuracy with minimal sample filtering, the results of the soft evidence experiment on the DAG structure show the same optimal accuracy with more filtering applied on the samples. In this case, soft evidence does not help in the classification goal.

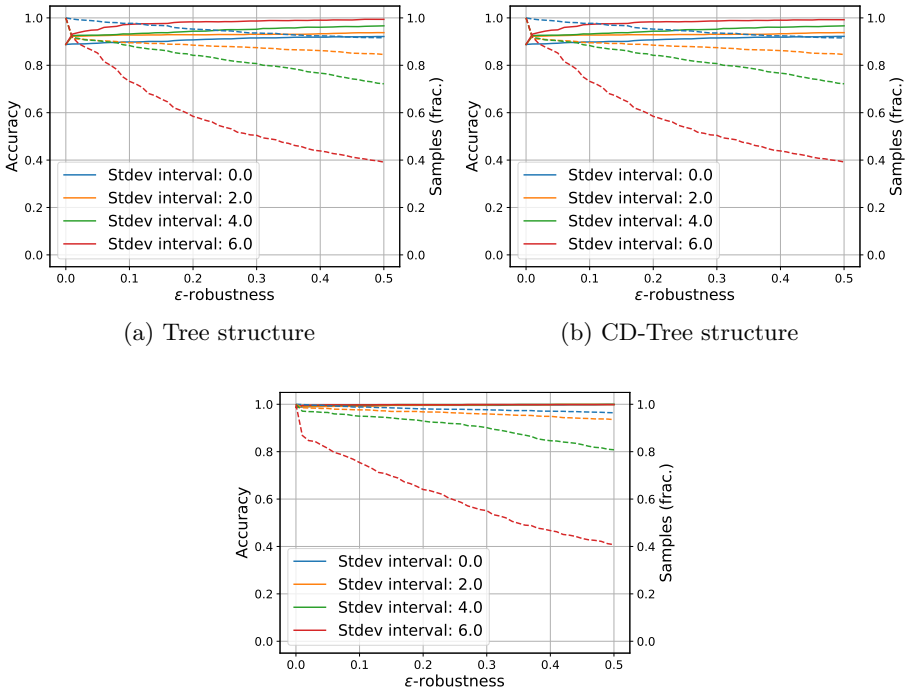
Authent

The Authent dataset performed very similar to the Texture dataset in the previous experiment, which is again the case for this experiment. Therefore, the results for this dataset are moved to Appendix A.2, Figure A.4.

5.2.4 Conclusion

This section will describe which conclusions are made from the experiment results. First of all, the results show that applying continuous soft evidence on the generalized credal classification algorithm is possible, showing one of possibly many applications making use of the general factorizing functions of which the imprecise expectation is to be computed. However, when comparing the soft evidence credal classification results with the regular credal classification results we can conclude that soft evidence might improve the accuracy, but it might also be the case that the accuracy is not altered or even deteriorates, all while filtering out more samples than the regular approach.

Another conclusion from the experiment results is that the continuous soft evidence implementation causes more samples to get a robustness value just above 0. These samples seem to be incorrectly classified in most cases, as multiple figures of different datasets show an increase in the accuracy after these samples are filtered out. Thus, it is possible that the soft evidence



(a) Tree structure

(b) CD-Tree structure

(c) CD-DAG

Figure 5.13: Soft evidence results on the Texture dataset for the Tree, CD-Tree and CD-DAG structures

causes incorrect samples to move towards lower robustness values. Noteworthy is that for a robustness threshold of 0, the accuracy of the soft evidence credal classification is equal to the regular credal classification accuracy. This means that the soft evidence only has an effect on the accuracy when the imprecision of the ϵ-contamination is applied. A possible cause might be that the values computed at the leaf distributions for both the singular observation likelihood and the interval observation probability are positively correlated, meaning if one increases the other one will increase as well. Without any imprecision by the ϵ-contamination, the result is always equal for both cases.

A possible reason why large intervals of size 6 can still perform relatively well in some cases is twofold. One reason is that even though an interval size of 6 results in an offset of 3 times the standard deviation in both directions, which results in a value of 0.997 with an observation equal to the mean, any observation further from the mean still results in a significant difference in value. This difference in value still resembles the difference in probability between the different observations.

Furthermore, when two leaf nodes have normal distributions over separate parts of the same domain and scope, then some observation could result in a high soft evidence value at one node distribution, while the other distribution results in a low soft evidence value. This could for example be the case when a dataset variable can be clearly split in distinct sets, for instance in classification tasks. Further research would be required to make further conclusions on this topic.

Chapter 6

Conclusions and recommendations

6.1 Conclusions

This thesis extends the work of Mauá et al. [15] on efficient inference of lower and upper expectations on tree-shaped CPNs. This is achieved by relaxing the structural constraints from trees to certain DAG structures. Furthermore, generalized factorizing functions can be dealt with the introduced imprecise expectation algorithm.

The next section presents an overview of the thesis conclusion. Furthermore, the main research question is answered. Section 6.1.2 discusses the conclusions regarding the general imprecise inference algorithm. Finally, Section 6.1.3 presents the conclusions gathered from the experiments.

6.1.1 Overview

Machine learning models are being used for an increasing number of, sometimes critical, tasks. Furthermore, sum-product networks have the advantage over other probabilistic models of being able to perform various exact inference queries efficiently. Hence, being able to use efficient imprecise queries on as many structures as possible allows for robustness analyses to come at almost no extra cost, when an SPN is already being used. Furthermore, the general factorizing functions allow for computing many different imprecise inference queries efficiently.

The answer to the main research question: *To what extent can we generalize the state-of-the-art credal SPNs while maintaining efficient inference?* can be summarized as follows:

- A general factorizing function can be efficiently evaluated with imprecision on tree-structured SPNs.
- A non-negative factorizing function can be efficiently evaluated with imprecision on DAG-structured SPNs.
- A general focused function can be efficiently evaluated with imprecision on DAG-structured SPNs, where no cycle edge is part of the paths from root to focus variable leaves.

6.1.2 Conclusions of the general imprecise inference algorithm

This thesis presents a proof on the correctness and efficiency of the introduced imprecise expectation algorithm in a modular way, where each different circuit operation and possible variations are proven independently. The considered cases result in the following conclusions:

- Leaf circuits can be computed efficiently, assuming that the expectation of individual functions of the factorizing function can be computed in constant time. As shown in Section 4.3.

- Product-based circuits compute correct imprecise expectations efficiently, without assumptions on the structure or the factorizing function. We do assume that the child circuit values have already been computed. This holds, because the greedy algorithm for product operations is both correct and efficient, as shown in Section 4.4.
- Sum-based circuits with disjoint child circuits compute correct imprecise expectations efficiently. This holds, because the weight sets in the child circuits do not overlap and because the linear program used for determining weight values can be solved in polynomial time, as shown in Section 4.5.1.
- Sum-based circuits with positive overlapping children and non-negative factorizing functions also compute correct imprecise expectations efficiently, because the same value of an overlapping circuit is used for the different parent circuits, due to the non-negative factorizing functions. This is discussed in Section 4.5.2.

We can also conclude that the imprecise expectation algorithm of Section 4.1 evaluates various imprecise queries using different types of factorizing functions efficiently. Consider the following imprecise queries:

- Imprecise expectations of general factorizing functions on tree structures.
- Imprecise expectations of non-negative factorizing functions on DAG structures. For example, imprecise likelihood queries.
- Imprecise conditional expectations of general focused query functions on tree structures and certain DAG structures. For example, credal classification.

The proposed algorithm is correct and efficient for these queries, because the modular theorems on the different circuit cases are combined with the recursive definition of probabilistic circuits. These proofs can be found in Section 4.6.

6.1.3 Conclusions of the experiments

Additionally, this thesis contains two experiments showing practical applications for the general factorizing functions. In the first experiment, credal tree structures were compared to credal DAG structures on the task of credal classification. The findings confirmed that:

- The results of the DAG-shaped structures are robust, since a positive correlation between the accuracy and the robustness value has been observed for DAG structures.
- Furthermore, the comparisons with the tree structures suggest that DAG structures can achieve a higher or similar accuracy compared to tree structures, with possibly more samples being filtered out.
- Finally, since the number of parameters is matched between the different structures, the evaluation time of the DAG structures is lower than that of the tree structures, due to the low number of nodes.

The second experiment is similar to the first experiment, as credal classification is performed. However, the factorizing function is adapted to compute the (continuous) soft evidence for all variables other than the class variable. This experiment shows a possible application of how the general factorizing functions can be used. We observed the following:

- The results for the effectiveness of soft evidence are mixed, with some improvements in the accuracy at the cost of filtering out many samples due to a low robustness.
- A second observation is that more samples have a robustness close to 0 while using soft evidence, of which a larger share is incorrectly classified due to the increased accuracies. Further experiments on using different factorizing functions are needed to make more concrete conclusions.

6.2 Recommendations for future work

In the previous section, a conclusion for the thesis has been presented, describing how the state-of-the-art credal SPNs have been generalized and summarizing which observations have been made from the experiment results. However, there are still some limitations and ideas that require further research. These are described in the following sections.

Section 6.2.1 discusses recommendations for future work regarding the general imprecise inference algorithm, while future applications of the algorithm are discussed in Section 6.2.2.

6.2.1 General imprecise inference algorithm

The proposed general imprecise inference algorithm covers many structures which are generated using structure learning algorithms, possibly combined with a class-discriminative structure. However, more exotic configurations of possible networks could be constructed, which are not contained in the provided proofs. These include structures built up from the modular theorems provided in this thesis, or structures built up from additional circuit cases, which require proving correct and efficient evaluation of the algorithm. These circuits would have to be sum-based circuits, as product-based circuits are already proven without any assumptions on the structure.

An example of such a structure is a directed acyclic graph, where the class variable is represented in a single leaf circuit with possibly multiple parent circuits. The value propagated by this leaf circuit is either positive or negative, but never conflicts with the value of another leaf circuit. This structure should also allow for efficient computation of imprecise conditional expectations on general focused query functions.

6.2.2 Experiments

The experiments executed in this thesis verify the robustness values on DAG structures and show possible applications of the general factorizing functions. Nevertheless, the datasets used have relatively few features. Further research could focus on performing experiments on datasets with more features, resulting in needing to use deeper structures. For example, a robustness analysis experiment on classification of an image dataset could be performed. The structure for this classification task could be the image-tailored structure, in combination with the class-discriminative top structure to maintain efficient imprecise inference.

Additionally, more experiments for the three types of factorizing functions and corresponding circuits structures could be performed:

- This could be to apply soft evidence to the feature variables on more datasets, possibly by using soft evidence on discrete features.
- A different experiment, which can be performed with the general factorizing function on tree structures, is where multiple functions over different variables of the factorizing function can be negative.
- Finally, a possible experiment for further research could be to make use of some other probabilistic model to represent each leaf circuit. In this case, the variables could be coupled in sets at the leaf circuits, this set of variables could then be passed to a probabilistic model which is trained on this smaller set. This probabilistic model could also allow for robust evaluations over the multivariate distribution it models. An example of such a probabilistic model is a variational autoencoder (VAE) [38, 3].

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 19

[2] Cory J Butz, Jhonatan S Oliveira, André E dos Santos, and André L Teixeira. Deep convolutional sum-product networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3248–3255, 2019. 9

[3] Alexander Camuto, Matthew Willetts, Stephen Roberts, Chris Holmes, and Tom Rainforth. Towards a theoretical understanding of the robustness of variational autoencoders. In *International Conference on Artificial Intelligence and Statistics*, pages 3565–3573. PMLR, 2021. 55

[4] Tijn Centen. GitHub repository: Beyond tree-shaped credal sum-product networks. `https://github.com/tijncenten/cspn`. 5

[5] Diarmaid Conaty, Jesús Martínez del Rincon, and Cassio P de Campos. A hierarchy of sum-product networks using robustness. *International Journal of Approximate Reasoning*, 113:245–255, 2019. 12

[6] Alvaro HC Correia and Cassio P de Campos. Towards scalable and robust sum-product networks. In *International Conference on Scalable Uncertainty Management*, pages 409–422. Springer, 2019. 12, 18, 21

[7] Alvaro HC Correia, Robert Peharz, and Cassio P de Campos. Towards robust classification with deep generative forests. *arXiv preprint arXiv:2007.05721*, 2020. 12, 40, 41, 46

[8] Fabio Gagliardi Cozman. Credal networks. *Artificial intelligence*, 120(2):199–233, 2000. 2

[9] Fabio Gagliardi Cozman. Graphical models for imprecise probabilities. *International Journal of Approximate Reasoning*, 39(2-3):167–184, 2005. 2

[10] Adnan Darwiche. A differential approach to inference in Bayesian networks. *J. ACM*, 50(3):280–305, May 2003. 7, 8

[11] Cassio P De Campos and Fabio Gagliardi Cozman. Inference in credal networks through integer programming. In *Proceedings of the Fifth International Symposium on Imprecise Probability: Theories and Applications*, 2007. 2

[12] Rob de Wit, Cassio P de Campos, Diarmaid Conaty, and Jesus Martinez del Rincon. Robustness in sum-product networks with continuous and categorical data. In *International Symposium on Imprecise Probabilities: Theories and Applications*, pages 156–158. PMLR, 2019. 12

[13] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. 18

[14] Aaron Dennis and Dan Ventura. Learning the architecture of sum-product networks using clustering on variables. In *Advances in Neural Information Processing Systems*, pages 2033–2041. Citeseer, 2012. 19

[15] Denis Deratani Mauá, Diarmaid Conaty, Fabio Gagliardi Cozman, Katja Poppenhaeger, and Cassio P de Campos. Robustifying sum-product networks. *International Journal of Approximate Reasoning*, 101:163–180, 2018. 2, 3, 4, 5, 11, 12, 21, 30, 31, 35, 46, 53

[16] Mattia Desana and Christoph Schnörr. Expectation maximization for sum-product networks as exponential family mixture models. *arXiv preprint arXiv:1604.07243*, 2016. 8

[17] Robert Gens and Pedro Domingos. Discriminative learning of sum-product networks. *Advances in Neural Information Processing Systems*, 25:3239–3247, 2012. 7, 8

[18] Robert Gens and Domingos Pedro. Learning the structure of sum-product networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 873–880, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. 9, 18, 19

[19] Vibhav Gogate, William Webb, and Pedro Domingos. Learning efficient Markov networks. *Advances in neural information processing systems*, 23:748–756, 2010. 9

[20] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, 2018. 10

[21] Isaac Levi. *The Enterprise of Knowledge: An Essay on Knowledge, Credal Probability, and Chance*. MIT press, 1983. 10

[22] David Lopez-Paz, Philipp Hennig, and Bernhard Schölkopf. The randomized dependence coefficient. *Advances in neural information processing systems*, 26:1–9, 2013. 18, 40

[23] Denis D Mauá, Cassio P De Campos, and Marco Zaffalon. Updating credal networks is approximable in polynomial time. *International Journal of Approximate Reasoning*, 53(8):1183–1199, 2012. 2

[24] Denis Deratani Mauá, Cassio P De Campos, Alessio Benavoli, and Alessandro Antonucci. Probabilistic inference in credal networks: New complexity results. *Journal of Artificial Intelligence Research*, 50:603–637, 2014. 2

[25] Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Pranav Subramani, Nicola Di Mauro, Pascal Poupart, and Kristian Kersting. SPFlow: An easy and extensible library for deep probabilistic learning using sum-product networks, 2019. 18, 19, 40, 41, 46

[26] Kevin P Murphy et al. Naive Bayes classifiers. *University of British Columbia*, 18(60):1–8, 2006. 8

[27] Radford M Neal and Geoffrey E Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998. 9

[28] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015. 9

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019. 19

[30] Robert Peharz, Bernhard C Geiger, and Franz Pernkopf. Greedy part-wise learning of sum-product networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 612–627. Springer, 2013. 19

[31] Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *LTPM workshop*, volume 32, 2014. 21

[32] Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044, 2016. 1

[33] Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van Den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7563–7574, Virtual, 13–18 Jul 2020. PMLR. 8, 9

[34] Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 744–752, San Diego, California, USA, 09–12 May 2015. PMLR. 8

[35] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 334–344, Tel Aviv, Israel, 22–25 Jul 2020. PMLR. 10, 18, 19, 20

[36] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690, 2011. 7, 9

[37] Amirmohammad Rooshenas and Daniel Lowd. Learning sum-product networks with direct and indirect variable interactions. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 710–718, Bejing, China, 22–24 Jun 2014. PMLR. 9, 10

[38] Ping Liang Tan and Robert Peharz. Hierarchical decompositional mixtures of variational autoencoders. In *International Conference on Machine Learning*, pages 6115–6124. PMLR, 2019. 55

[39] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, June 2014. OpenML website: `https://www.openml.org/s/99/data`. 40, 46

[40] A. Vergari, R. Peharz, Y. Choi, and G. Van den Broeck. Probabilistic circuits: Representations, inference, learning and theory. *ECML-PKDD*, 2020. 9

[41] Peter Walley. Statistical reasoning with imprecise probabilities. 1991. 1

[42] Jos Wolfshaar and Andrzej Pronobis. Deep generalized convolutional sum-product networks. In *International Conference on Probabilistic Graphical Models*, pages 533–544. PMLR, 2020. 9

[43] Barnet Woolf. The log likelihood ratio test (the G-test). *Annals of human genetics*, 21(4):397–409, 1957. 9

[44] Marco Zaffalon. The naive credal classifier. *Journal of statistical planning and inference*, 105(1):5–21, 2002. 10, 11, 35

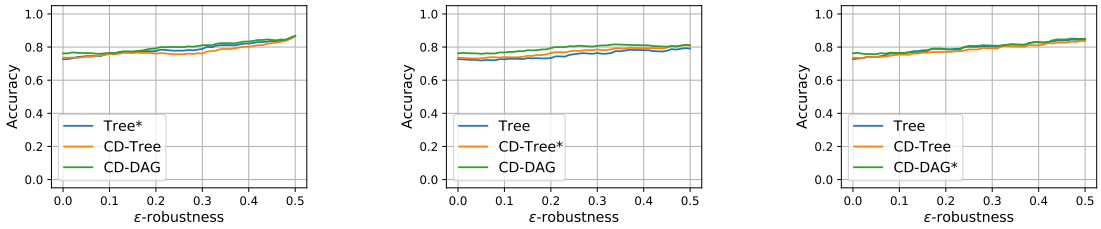
Appendix A

Experiments

In this appendix chapter unused results of the experiments of Chapter 5 are placed. Both for the structure comparison experiment (Section A.1) and the soft evidence experiment (Section A.2).

A.1 Comparison results

In this section, the sample set comparison for the Diabetes dataset is shown in Figure A.1. Furthermore, the structure comparison and the sample set comparison for the Texture dataset are shown in Figures A.2 and A.3 respectively.



(a) Tree set (b) CD-Tree set (c) CD-DAG set

Figure A.1: Diabetes dataset accuracies compared using the same filtered sample sets

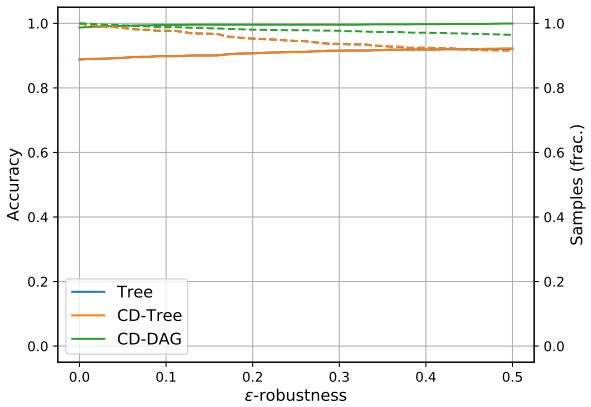
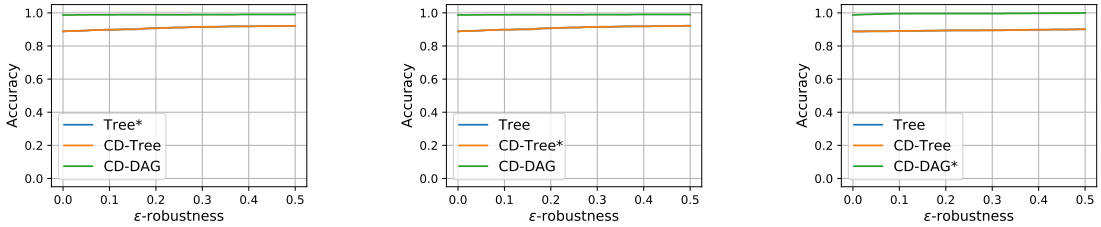


Figure A.2: Structure comparison on Texture dataset

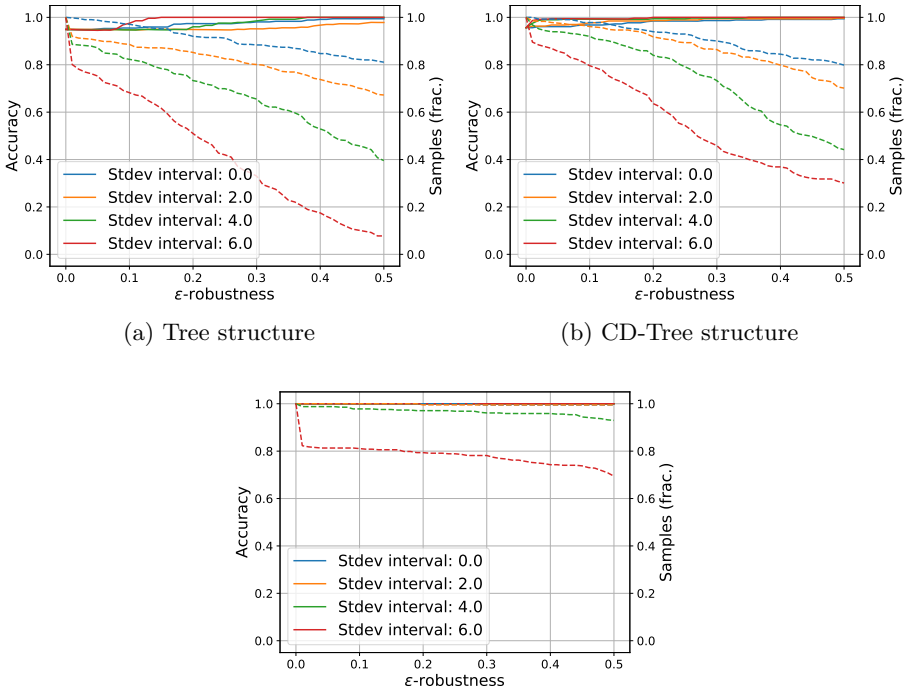


(a) Tree set (b) CD-Tree set (c) CD-DAG set

Figure A.3: Texture dataset accuracies compared using the same filtered sample sets

A.2 Soft evidence results

The results of the soft evidence experiment for the Authent dataset are shown in Figure A.4.



(a) Tree structure (b) CD-Tree structure

(c) CD-DAG

Figure A.4: Soft evidence results on the Authent dataset for the Tree, CD-Tree and CD-DAG structures