5th International Symposium on Imprecise Probability: Theories and Applications, Prague, Czech Republic, 2007

Inference in Credal Networks Through Integer Programming

Cassio Polpo de Campos
Escola de Artes, Ciências e Humanidades
Universidade de São Paulo, Brazil
cassiopc@usp.br

Fabio Gagliardi Cozman
Escola Politécnica
Universidade de São Paulo, Brazil
fgcozman@usp.br

Abstract

A credal network associates a directed acyclic graph with a collection of sets of probability measures; it offers a compact representation for sets of multivariate distributions. In this paper we present a new algorithm for inference in credal networks based on an integer programming reformulation. We are concerned with computation of lower/upper probabilities for a variable in a given credal network. Experiments reported in this paper indicate that this new algorithm has better performance than existing ones for some important classes of networks.

Keywords. Credal networks, Integer programming.

1 Introduction

This paper presents novel techniques for marginal inference in *credal networks*. The goal is to provide an algorithm that can handle graphical models for precise and imprecise probabilistic assessments based on integer programming.

Credal networks represent a set of joint probability measures through a directed acyclic graph and a collection of local sets of probability measures [3, 5, 13]. The structure of the graph indicates relations of independence between variables; the "size" of the sets of probabilities encodes the imprecision in the probability values. Section 2 reviews basic properties of credal networks and Section 3 addresses the inference problem we are interested in. Basically, a belief updating inference in the context of credal networks is a computation of upper/lower probability for some conjunction of events, given observations. Most existing algorithms for belief updating cannot handle large networks; when they can, they suffer from numerical instability [9].

This paper aims at enlarging the class of networks that can be successfully processed exactly. We focus on developing a reformulation that is particularly efficient for polytree-shaped networks. Section 4 reviews a multilinear reformulation and presents a new inference algorithm based on bilinear and integer programming. Section 5 shows through experiments that the algorithm can process large polytree networks, surpassing existing algorithms [8, 9]. Section 6 concludes the paper.

2 Credal sets and credal networks

A few preliminary definitions are important. A convex set of probability distributions is called a *credal set* [18]. A credal set for X is denoted by $K(X)$; we assume that every random variable is categorical and that every credal set has a finite number of vertices. A conditional credal set is a set of conditional distributions, obtained by applying Bayes rule to each distribution in a credal set of joint distributions. The theory of sets of probability distributions adopted in this paper can be placed in the framework of coherent behavior by selecting axioms advocated by several authors, for instance by Walley [23]. We emphasize that our setting is restricted to categorical variables, thus we can brush away subtle but crucial differences between proposed frameworks concerning issues of conglomerability and countable additivity.

The sets $K(X|Y)$ are *separately specified* when there is no constraint on the conditional set $K(X|Y = y_1)$ that is based on the properties of $K(X|Y = y_2)$, for any $y_2 \neq y_1$ — that is, the conditional sets bear no relationship to each other. In this paper we assume that local credal sets are always separately specified; justifications for this separability assumption can be found in [7]. Given a number of marginal and conditional credal sets, an *extension* of these sets is a joint credal set with the given marginal and conditional credal sets. In this paper we are exclusively concerned with the largest possible extension for any collection of marginal and conditional credal sets.

Given a credal set $K(X)$ and an event A, the *upper* and *lower* probability of A are respectively $\overline{P}(A) = \max_{p(X) \in K(X)} P(A)$ and $\underline{P}(A) = \min_{p(X) \in K(X)} P(A)$.

A *credal network* $N = (G, \mathbb{X}, \mathbb{K})$ is composed by a directed acyclic graph $G = (V, E)$ where each node of V is associated with a random variable $X_i \in \mathbb{X}$ and with a collection of conditional credal sets $K(X_i|\text{pa}(X_i)) \in \mathbb{K}$, where $\text{pa}(X_i)$ denotes the parents of the node associated to X_i in the graph. In the remainder of this paper, we refer to X_i and its associated node interchangeably. Note that we have a conditional credal set related to X_i for each instantiation of $\text{pa}(X_i)$. A root node is associated with a single marginal credal set. We take that in a credal network every random variable is independent of its nondescendants nonparents given its parents; this is the *Markov condition* on the network. In this paper we adopt the concept of *strong independence*[1]: two random variables X and Y are strongly independent when every extreme point of $K(X, Y)$ satisfies standard stochastic independence of X and Y (that is, $p(X|Y) = p(X)$ and $p(Y|X) = p(Y)$) [5]. Strong independence is the most commonly adopted concept of independence for credal sets, probably due to its obvious connection with standard stochastic independence. There are concepts of independence that are less precise in the sense that they admit distributions that do not factorize; an example is epistemic independence [11, 23].

Given a credal network, an *extension* of the network is any joint credal set that satisfies all constraints encoded in the network. The *strong extension* of a credal network is the largest joint credal set such that every variable is strongly independent of its nondescendants nonparents given its parents. The strong extension of a credal network is the joint credal set that contains every possible combination of vertices for all credal sets in the network [6]; that is, each vertex of a strong extension factorizes as follows:

$$p(X_1, \ldots, X_n) = \prod_i p(X_i|\text{pa}(X_i)) \,. \tag{1}$$

3 Inference with strong extensions

A *marginal inference* in a credal network is the computation of lower/upper probabilities in an extension of the network. If X_q is a *query* variable and $\mathbf{X}_E$ represents a set of *observed* variables, then an inference is the computation of tight bounds for $p(X_q|\mathbf{X}_E)$ for one or more values of X_q. For inferences in strong extensions, it is known that the distributions that minimize/maximize $p(X_q|\mathbf{X}_E)$ belong to the set of vertices of the extension [13].

[1]We note that other concepts of independence are found in the literature [4, 12].

An inference can be produced by combinatorial optimization, as we must find a vertex for each local credal set $K(X_i|\text{pa}(X_i))$ so that Expression (1) leads to a maximum/minimum of $p(X_q|\mathbf{X}_E)$. In general, inference offers tremendous computational challenges — consider the following example, taken from Rocha et al. [8]. Take a network with three nodes, $X \rightarrow Y \leftarrow Z$, where X, Y and Z have four categories each, and where all credal sets have four vertices each. There are 4^{18} different joint distributions factorizing as Expression (1), where local distributions are vertices of local credal sets. Rocha et al. [8] discuss branch-and-bound procedures that can handle situations such as this, but that still have difficulties in large networks. The only known polynomial algorithm for strong extensions is the 2U algorithm, which only processes polytrees with *binary* variables [13]. Other exact inference algorithms based on enumeration examine all potential vertices of the strong extension to produce the required lower/upper values [2, 3, 5, 7]; these algorithms face serious difficulties in large networks.

A different way to look at the computation of inferences is to recognize that a lower/upper value for $p(X_q|\mathbf{X}_E)$ is obtained by minimization/maximization of a fraction containing polynomials in probability values. This is in fact the strategy discussed in Section 4; our results suggest that this is the most profitable strategy to take for exact inference with strong extensions.

4 Inference as a multilinear programming problem

A marginal inference for a strong extension can be formulated as a multilinear programming problem. The goal is to minimize/maximize the expression

$$\sum_{X_i \setminus X_q} \prod_i p(X_i|\text{pa}(X_i)) \tag{2}$$

subject to constraints on the local probabilities $p(X_i|\text{pa}(X_i))$. For a query with evidence, we may use the constraint $p(X_q|\mathbf{X}_E) = \frac{p(X_q, \mathbf{X}_E)}{p(\mathbf{X}_E)}$, that can be turned into a multilinear constraint.[2] In this problem we must deal with a large number of terms in the multilinear objective function (the number of terms is exponential on the size of the network), as shown in Example 1.

Example 1 *Take the network presented in Figure 1. Suppose that random variables are binary and we want*

[2]We assume that the probability of evidence is strictly greater than zero, leaving for future work the important case where lower probabilities equal to zero may happen.

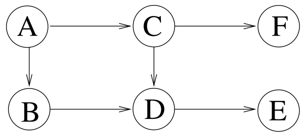

Figure 1: Simple multi-connected network.

to evaluate the maximum possible value for the probability of $(E = e) \wedge (F = f)$*; this is obtained by solving:*

$$\max \sum_{A,B,C,D} p(f|C) \cdot p(e|D) \cdot p(D|B,C) \cdot \cdot p(B|A) \cdot p(C|A) \cdot p(A)\,, \tag{3}$$

subject to linear constraints (from local credal sets). We have a multilinear objective function with 16 nonlinear terms of degree six. The probability functions p are seen as optimization variables in the multilinear program.

As presented by Campos and Cozman [9], we can run a symbolic variable elimination algorithm to obtain a simpler objective function.

Example 2 *Take the network and specifications of Example 1. Instead of optimizing Expression (3), the symbolic variable elimination procedure transforms it into a problem with simpler multilinear functions of degree at most three, by grouping terms and introducing new optimization variables. The result is a multilinear program with 22 nonlinear terms:*

$$\max \sum_{D} p(e|D)\, p(D,f) \quad \text{subject to}$$

$$p(B,C) = \sum_{A} p(B|A)\, p(C|A)\, p(A)\,, \textit{ for all } B,C$$

$$p(C,D) = \sum_{B} p(D|B,C)\, p(B,C)\,, \textit{ for all } C,D$$

$$p(D,f) = \sum_{C} p(f|C)\, p(C,D)\,, \textit{ for all } D$$

plus the linear constraints.

Note that the reformulated problem presented in Example 2 contains terms of smaller degree than the original problem (Example 1). This is important in multilinear programming, as it is difficult to handle problems with high degree. Besides that, the transformation usually leads a smaller number of nonlinear terms than in the direct version given by Expression (1), although that was not the case in Example 2.

4.1 A bilinear transformation

The new multilinear terms obtained with the symbolic variable elimination procedure just described have smaller degree than the original ones, but the maximum degree is at least as large as the tree-width of network's moral graph (even if we know an optimal elimination ordering for the variables) [9].

We present in this section a new transformation procedure, which naturally produces multilinear programming problems with maximum degree of two (that is, bilinear problems) regardless of network's topology, and where the following property holds: each bilinear term has at least one variable that is defined (even though by a credal set) in the input (that is, each bilinear term has at most one auxiliary variable). This property will be essential for obtaining an integer program in Section 4.2, and it is specially efficient for polytree networks, which are defined by graph without any cycles (directed or not).

Prior to the algorithm itself, we must present some useful definitions.

Definition 3 *An ordering for the network variables is said a* precedence ordering *if, for each variable in the ordering, all its ancestors in the network's graph appear before it in the ordering.*

Definition 4 (Robertson and Seymour [20, 21]). *Given a graph* $G = (V, E)$*, a sequence* $V_1, \ldots, V_r$ *of subsets of* V *is a* path-decomposition *of* G *if the following conditions are satisfied:*

- $\bigcup_i V_i = V$.
- *For every edge* $e \in E$*, some* V_i *contains both endpoints of* e.
- *For* $1 \leq i \leq j \leq k \leq r$, $V_i \cap V_k \subseteq V_j$.

Definition 5 (Robertson and Seymour [20, 21]). *The* path-width *of* G*, denoted by* $pw(G)$*, is the minimum value* $h \geq 0$ *such that* G *has a path-decomposition* $V_1, \ldots, V_r$ *with* $|V_i| \leq h + 1$ *for* $i = 1, \ldots, r$.

Definition 6 *The path-width of a credal network* $N = (G, \mathbb{X}, \mathbb{K})$*, or just* $pw(N)$*, is the path-width of its graph* G.

The idea of `Bilinear-Transformation` algorithm is to process the network variables top-down, using a *precedence ordering*. At each step we construct a constraint that defines the relationship between the query and the current variable being processed. A variable may be processed only if all its ancestors have already been processed. The active nodes at each step form a path-decomposition of the network's graph. Note that we cannot use other decompositions such as joint trees, because we would get multilinear terms with more than one auxiliary variable, that is, the result would not be a bilinear programming problem with that described property. We proceed with the idea of the transformation using an example.

Example 7 *Suppose we want to query the probability of e, f in the network presented in Figure 1. The first step of the* `Bilinear-Transformation` *algorithm is to choose a precedence ordering for the network variables (when there is evidence, all the process must be repeated for the queries and for the observed variables). We will use the ordering A, C, B, D, E, F. The first variable to be processed is A (it is the only variable without parents). We have the queries e, f and will write their joint probability using $p(A)$ (which is defined in the network specification) and inserting A in the conditional part. So we create the constraint*

$$p(e,f) = \sum_{A\in\{a,\overline{a}\}} p(A)\cdot p(e,f|A).$$

Functions $p(e,f|A)$ are auxiliary (they do not appear in the network), and we must create constraints to define them (for all possible instantiations of A). The current variable to be processed is C. Thus, for all $A \in \{a,\overline{a}\}$:

$$p(e,f|A) = \sum_{C\in\{c,\overline{c}\}} p(C|A)\cdot p(e,f|A,C).$$

At this stage, our queries are conditioned on A and C. Following the idea, we process B, obtaining

$$p(e,f|A,C) = \sum_{B\in\{b,\overline{b}\}} p(B|A)\cdot p(e,f|B,C),$$

which must be written for all $A \in \{a,\overline{a}\}, C \in \{c,\overline{c}\}$. Note that at this point A disappeared from the conditioning side, because B and C together separate the query variables from A. Now the current variable to be treated is D, and our queries are conditioned on B, C, that is, we must define how to evaluate $p(e,f|B,C)$. We have, for all $B \in \{b,\overline{b}\}, C \in \{c,\overline{c}\}$, that

$$p(e,f|B,C) = \sum_{D\in\{d,\overline{d}\}} p(D|B,C)\cdot p(e,f|C,D).$$

At this moment, e, f are conditioned on C, D (again, B is not present anymore as C, D separate the queries from B). Now we will process E, but because the only two remaining variables are E and F and they are not parent of each other, their order in fact does not matter. Thus,

$$p(e,f|C,D) = p(e|D)\cdot p(f|C),$$

for all $C \in \{c,\overline{c}\}, D \in \{d,\overline{d}\}$. Note that, as $p(f|C)$ is specified in the network, we can stop. We have completed the procedure as both $p(e|D)$ and $p(f|C)$ appear in the network. Note that, if we had chosen another ordering such as A, B, C, D, E, F or A, B, C, F, D, E, more constraints might be needed.

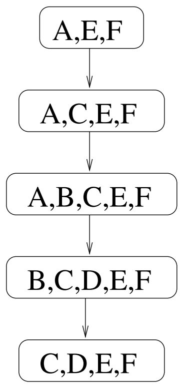

Figure 2: Path-decomposition of Example 7.

Figure 2 shows the path-decomposition induced by the ordering of Example 7. Note that there is an one-to-one relation between decomposition components and constraints in the example. The elements in a given component appear together in some constraint of the reformulation, either in the conditioned (including queries) or in the conditioning sides.

The algorithm is presented using pseudo-code in Figure 3. Functions g appearing in line 24 of the algorithm are just conditioned probability functions. We use the letter g instead of p because of another transformation presented in next section, where functions g have special meaning. If we just want a bilinear transformation, g should be simply replaced by p, even though we note that names of optimization variables are not an issue (they just need to be coherent among each other). Regarding the complexity of the algorithm, we define the path-width of a precedence ordering as the width of the path-decomposition induced by that ordering, and present the following theorem.

Theorem 8 *Let $N = (G, \mathbb{X}, \mathbb{K})$ be a credal network where the maximum number of categories of a random variable is $O(|V|)$. Suppose o' is a precedence ordering for the variables in $\mathbb{X}$. Then* `Bilinear-Transformation` *runs in time $O(|V|^{pw(o')+k})$, for k constant.*

Proof: All non-loop lines of the algorithm can clearly be executed in polynomial time in the number of nodes, that is, $O(|V|^K)$, for K constant.

The loop of line 4 is executed twice if we have evidence, and only once if we do not have evidence. Line 12 loop is executed $|U|$ times, that is, $O(|V|)$. The loop of line 26 is executed $O(c)$ times, where c is the

```
BILINEAR-TRANSFORMATION($N, \mathbb{Q}, \mathbb{E}$)
        $N = (G, \mathbb{X}, \mathbb{K})$ is the network, $G = (V, E)$ its graph, $\mathbb{X}$ its variables and $\mathbb{K}$ its local credal sets.
        $\mathbb{Q}$ is an instantiation for a set of queried variables.
        $\mathbb{E}$ is an instantiation for a set of observed variables.
        The result of this procedure is a bilinear programming problem
        (bilinear objective function of line 1 and a set of bilinear constraints from lines 2 and 24).
 1   ▷ The program will maximize or minimize $t$, which will be the objective function.
 2   Insert the following constraint into the bilinear program:
            $p(\mathbb{Q}, \mathbb{E}) = t \cdot p(\mathbb{E})$
 3   ▷ Now we create constraints to evaluate $p(\mathbb{Q}, \mathbb{E})$ and $p(\mathbb{E})$.
 4   for $W = \mathbb{Q} \cup \mathbb{E}$ and $W = \mathbb{E}$
 5       do
 6           ▷ $U$ is the set of all relevant variables.
 7           $U \leftarrow \{X \in V \setminus W$ such that $X$ is an ancestor of some $w \in W\}$.
 8           ▷ There are many ways to choose a precedence ordering. Do it polynomially.
 9           Rename the variables of $U$ as $X_1, X_2, \ldots, X_{|U|}$ according to a precedence ordering.
10           ▷ Initially, functions are not conditioned. $L$ is a list of sets of conditioning variables.
11           Let $L$ be an empty queue of sets. Insert $\emptyset$ in the end of $L$.
12           for $i \leftarrow 1$ to $|U|$
13               do
14                   ▷ $L'$ will have the conditioning sets to be considered on the next loop step.
15                   Let $L'$ be an empty queue of sets.
16                   ▷ Conditioning sets from the previous step are processed.
17                   while $L$ is not empty
18                       do
19                           $S \leftarrow$ first element of $L$ (remove $S$ from $L$).
20                           ▷ $S'$ are separated variables (with respect to the query variables) when
                               inserting $X_i$ in the conditioning part. The separation is based on
                               the graph structure (known as d-separation [19]).
21                           $S' \leftarrow \{s \in S$ such that $\{X_i\} \cup S \setminus \{s\}$ separates $W$ from $s\}$.
22                           ▷ The variables in $S'$ are no more relevant.
23                           $R \leftarrow S \setminus S'$.
24                           Depending on $W$, insert the following constraint into the bilinear program:
                                   if $W = \mathbb{Q} \cup \mathbb{E}$ then $p(W|S) = \sum_{x_{ij}} p(x_{ij}|\mathrm{pa}(X_i)) \cdot p(W|R, x_{ij})$.
                                   if $W = \mathbb{E}$ then $g(W|S) = \sum_{x_{ij}} p(x_{ij}|\mathrm{pa}(X_i)) \cdot g(W|R, x_{ij})$.
                                   ▷ $x_{ij}$ is a category of $X_i$.
                                   ▷ $\mathrm{pa}(X_i)$ is an instantiation complying with $W$, $R$ and $S$.
25                           ▷ Now we insert into $L'$ the conditioning sets for the next step.
26                           for each $x_{ij}$ of $X_i$
27                               do
28                                   $R' \leftarrow R \cup \{x_{ij}\}$.
29                                   Insert $R'$ in the end of $L'$.
30                           ▷ End of for
31                   ▷ End of while
32                   $L \leftarrow L'$
33           ▷ End of for
34   ▷ End of for
```

Figure 3: Reformulation algorithm for inferences in credal networks.

maximum number of categories of a random variable.

Line 17 loop executes $|L|$ times. Let u be the maximum size of a set stored in L. Then the number of elements in L at each round is $O(c^u)$, because at most c^u unequal instantiations for u variables are possible. Note that the sizes of sets stored in L are exactly the sizes of components in the path-decomposition induced by o'. So the bottleneck is the size of the elements in L, which is equal (in worst case) to $pw(o')$. Thus the complexity of `Bilinear-Transformation` is $(k = K + 2)$:

$$2 \cdot O(|V|) \cdot O(|V|^K) \cdot O(c) \cdot O(c^{pw(o')}) = O(|V|^{pw(o')+k}).$$

Note that, with recent results of Feige et al. [14], it is possible to approximate the optimum path-width of the network by $\log |V| \sqrt{\log pw(N)}$. So, the algorithm runs in time $O(|V|^{\log c \cdot pw(N) \cdot \sqrt{\log pw(N)} + k})$. □

Corollary 9 *When restricted to polytrees, the algorithm* `Bilinear-Transformation` *runs in polynomial time in the size of input.*

Proof: In a polytree, each variable separates its parents from its descendants. The path-width is bounded by d, the maximum degree of network's graph, and it is easy to find an ordering with such width (a greedy algorithm will succeed). So the complexity is $O(|V|^d \cdot |V|^k)$, k constant. As the input size needed to specify the local credal sets of the network is already exponential on d, the corollary follows. □

4.2 An integer programming version

We show in this section how to obtain an integer program from that bilinear program generated by `Bilinear-Transformation`. We must note some useful properties:

1. A multiplication of a rational optimization variable $x \in [0, 1]$ by a boolean variable $b \in \{0, 1\}$ can be encoded by linear constraints: replace the nonlinear term $x \cdot b$ by a new variable y_{xb} and insert the constraints:
$$\begin{aligned} 0 \leq \quad & y_{xb} \quad \leq b \\ x - 1 + b \leq \quad & y_{xb} \quad \leq x \end{aligned}$$

2. We can represent each local credal set as a combination of its vertices. Suppose X is a network variable with parents $Y_1, \ldots, Y_r$, and that vertices $\alpha_1, \ldots, \alpha_s$ define the credal set for $p(X|y_1, \ldots, y_r)$ (the dimension of each α_i equals the number of categories of X). For each instantiation of $X, Y_1, \ldots, Y_r$ we have
$$p(x|y_1, \ldots, y_r) = \sum_{i=1}^{s} \alpha_i(x) \cdot b^{(i)}_{y_1,\ldots,y_r}, \qquad (4)$$
where $\alpha_i(x)$ are known values (specified in the network) and $b^{(i)}_{y_1,\ldots,y_r}$ are boolean variables such that
$$\sum_{i=1}^{s} b^{(i)}_{y_1,\ldots,y_r} = 1,$$
that is, only one of these $b^{(i)}_{y_1,\ldots,y_r}$ variables is one, thus selecting a vertex.

3. Each nonlinear term appearing in the constraints created by `Bilinear-Transformation` is a multiplication of a rational variable and a variable appearing in the network specification (defined by the local credal sets).

These observations lead us to the following procedure to replace each product $r \cdot p(x|y_1, \ldots, y_r)$ of each constraint created by `Bilinear-Transformation`:

$$\begin{aligned} r \cdot p(x|y_1, \ldots, y_r) \quad &:= \quad \sum_{i=1}^{s} \alpha_i(x) \cdot y^{(i)}_{rb_{y_1,\ldots,y_r}}, \\ y^{(i)}_{rb_{y_1,\ldots,y_r}} \quad &\geq \quad 0, \\ y^{(i)}_{rb_{y_1,\ldots,y_r}} \quad &\leq \quad b^{(i)}_{y_1,\ldots,y_r}, \\ y^{(i)}_{rb_{y_1,\ldots,y_r}} \quad &\geq \quad r - 1 + b^{(i)}_{y_1,\ldots,y_r}, \\ y^{(i)}_{rb_{y_1,\ldots,y_r}} \quad &\leq \quad r, \\ \sum_{i=1}^{s} b^{(i)}_{y_1,\ldots,y_r} \quad &= \quad 1, \end{aligned}$$

where $A := B$ means to replace A by B. Although we need to work with all vertices of credal sets and it may be hard to enumerate all of them, many important models can easily be translated into lists of vertices. For example, capacities of infinite order (also known as belief functions) can be expressed by mass assignments that are attached to sets of categories; vertices are simply obtained by combining the ways in which mass assignments are to be distributed [22, 23].

There is still a problem to address to get an integer version. The constraint inserted during line 2 of algorithm `Bilinear-Transformation` is nonlinear and the variables involved in its product are not in the network specification and thus cannot be directly replaced by some linear constraints and integer variables. We solve this problem by calling the loop of line 4 twice: in the first time, we evaluate $p(\mathbb{Q}, \mathbb{E})$ (using functions named p); in the second time, we evaluate $g(\mathbb{E}) = t \cdot p(\mathbb{E})$, that is, functions g do not mean probability functions but t times probability functions.

For example, the constraint in line 2 becomes simply $p(\mathbb{Q}|\mathbb{E}) = g(\mathbb{E})$. Each constraint inserted in line 24 of the algorithm

$$g(W|S) = \sum_{x_{ij}} p(x_{ij}|\mathrm{pa}(X_i)) \cdot g(W|R, x_{ij})$$

in fact means

$$t \cdot p(W|S) = \sum_{x_{ij}} p(x_{ij}|\mathrm{pa}(X_i)) \cdot t \cdot p(W|R, x_{ij}),$$

with the t variable hidden inside the g functions, whose are seen as optimization variables. So, on the last step of the inner loop, the constraint $g(W|S) = t \cdot p(W|S)$ must be included to pull t out of g, transforming it into p again. Because the $p(W|S)$ of the last step is certainly specified in the credal network input, this product can now be linearized using those ideas described on items from 1 to 3.

Example 10 *Suppose we want to evaluate $p(a|d)$ in the network of Figure 1. First, we symbolically evaluate $p(a, d)$ using p functions:*

$$\begin{aligned}
p(a,d) &= p(a) \cdot p(d|a) \\
p(d|a) &= p(b|a) \cdot p(d|a,b) + p(\overline{b}|a) \cdot p(d|a,\overline{b}) \\
p(d|a,b) &= p(c|a) \cdot p(d|b,c) + p(\overline{c}|a) \cdot p(d|b,\overline{c}) \\
p(d|a,\overline{b}) &= p(c|a) \cdot p(d|\overline{b},c) + p(\overline{c}|a) \cdot p(d|\overline{b},\overline{c})
\end{aligned}$$

Note that we have both p functions defined in the network and auxiliary p functions. Now we evaluate $g(d)$, using g functions that hide t until the last step:

$$\begin{aligned}
g(d) &= p(a) \cdot g(d|a) + p(\overline{a}) \cdot g(d|\overline{a}) \\
g(d|a) &= p(b|a) \cdot g(d|a,b) + p(\overline{b}|a) \cdot g(d|a,\overline{b}) \\
g(d|\overline{a}) &= p(b|\overline{a}) \cdot g(d|\overline{a},b) + p(\overline{b}|\overline{a}) \cdot g(d|\overline{a},\overline{b}) \\
g(d|a,b) &= p(c|a) \cdot g(d|b,c) + p(\overline{c}|a) \cdot g(d|b,\overline{c}) \\
g(d|a,\overline{b}) &= p(c|a) \cdot g(d|\overline{b},c) + p(\overline{c}|a) \cdot g(d|\overline{b},\overline{c}) \\
g(d|\overline{a},b) &= p(c|\overline{a}) \cdot g(d|b,c) + p(\overline{c}|\overline{a}) \cdot g(d|b,\overline{c}) \\
g(d|\overline{a},\overline{b}) &= p(c|\overline{a}) \cdot g(d|\overline{b},c) + p(\overline{c}|\overline{a}) \cdot g(d|\overline{b},\overline{c}) \\
g(d|b,c) &= t \cdot p(d|b,c) \\
g(d|b,\overline{c}) &= t \cdot p(d|b,\overline{c}) \\
g(d|\overline{b},c) &= t \cdot p(d|\overline{b},c) \\
g(d|\overline{b},\overline{c}) &= t \cdot p(d|\overline{b},\overline{c})
\end{aligned}$$

To force t as the variable to maximize/minimize, we impose that $p(a, d) = g(d)$ (remember that $g(d)$ means $t \cdot p(d)$). Now take the last constraint ($g(d|\overline{b},\overline{c}) = t \cdot p(d|\overline{b},\overline{c})$) to illustrate the linearization of a product (the same idea must be applied to all products in all constraints). Suppose that $p(d|\overline{b},\overline{c}) \in [l, u]$, with l and u known. The constraint becomes

$$g(d|\overline{b},\overline{c}) = l \cdot y^{(1)}_{tp(d|\overline{b},\overline{c})} + u \cdot y^{(2)}_{tp(d|\overline{b},\overline{c})}$$

and we include

$$\begin{aligned}
y^{(1)}_{tp(d|\overline{b},\overline{c})} &\geq 0, \\
y^{(1)}_{tp(d|\overline{b},\overline{c})} &\leq b^{(1)}_{d|\overline{b},\overline{c}}, \\
y^{(1)}_{tp(d|\overline{b},\overline{c})} &\geq t - 1 + b^{(1)}_{d|\overline{b},\overline{c}}, \\
y^{(1)}_{tp(d|\overline{b},\overline{c})} &\leq t, \\
y^{(2)}_{tp(d|\overline{b},\overline{c})} &\geq 0, \\
y^{(2)}_{tp(d|\overline{b},\overline{c})} &\leq b^{(2)}_{d|\overline{b},\overline{c}}, \\
y^{(2)}_{tp(d|\overline{b},\overline{c})} &\geq t - 1 + b^{(2)}_{d|\overline{b},\overline{c}}, \\
y^{(2)}_{tp(d|\overline{b},\overline{c})} &\leq t, \\
b^{(1)}_{d|\overline{b},\overline{c}} + b^{(2)}_{d|\overline{b},\overline{c}} &= 1,
\end{aligned}$$

where the new created boolean variables $b^{(i)}_{d|\overline{b},\overline{c}}$ indicate which vertex to use: l or u. The variable $p(d|\overline{b},\overline{c})$ has disappeared (its possible values l and u still remain), and t and new variables $b^{(i)}_{d|\overline{b},\overline{c}}$ appear linearly in the constraints.

Because only one vertex of each local credal set will be chosen, we can go further in the reformulation, obtaining a smaller number of boolean optimization variables. According to the linearization just described, we represent each local credal set as a combination of its vertices and create one boolean optimization variable for each vertex of each local credal set. Instead of this transformation, we can use another idea, interpreting the boolean optimization variables as the binary representation of a vertex index. Suppose $\alpha_0, \ldots, \alpha_{s-1}$ are the vertices and that s is a power of two (we do not loose generality because, if s is not a power of two, we can always repeat several times one of the already existent vertices to reach the next power of two; these additional vertices do not change the result as they are equal to some old vertex). Now let $1 \leq j \leq \log_2 s$ be an integer indexing a bit of the number i, and for each instantiation of variable X with parents $Y_1, \ldots, Y_r$ we define

$$p(x|y_1, \ldots, y_r) = \sum_{i=0}^{s-1} \alpha_i(x) \times \prod_{\text{bit } j \text{ of } i} b^j_{y_1,\ldots,y_r} \times \prod_{\text{not bit } j \text{ of } i} (1 - b^j_{y_1,\ldots,y_r}), \quad (5)$$

where (not) bit j of i means that the jth bit of i is (not) one. That is, instead of a boolean variable that indicates (with a zero or one) if a given vertex should be used (and only one of them actually should), we multiply a collection of boolean variables according to the binary representation of i (the vertex index). This product guarantees that the result is one if and only if all b variables of its binary representation are set to one.

Example 11 *Let X be a random variable with three categories (x_0, x_1, x_2) and one parent, named Z. Let Z have two categories $(z, \overline{z})$. Suppose the credal set for $p(X|z)$ has four vertices $(\alpha_0, \ldots, \alpha_3)$ with three dimensions each. Then we define the boolean optimization variables $b_z^{(1)}, b_z^{(2)}$ and the constraints:*

$$\begin{aligned}
p(x_0|z) &= \alpha_0(x_0)\cdot(1-b_z^{(1)})\cdot(1-b_z^{(2)}) + \\
&\quad \alpha_1(x_0)\cdot b_z^{(1)}\cdot(1-b_z^{(2)}) + \\
&\quad \alpha_2(x_0)\cdot(1-b_z^{(1)})\cdot b_z^{(2)} + \\
&\quad \alpha_3(x_0)\cdot b_z^{(1)}\cdot b_z^{(2)} \\
p(x_1|z) &= \alpha_0(x_1)\cdot(1-b_z^{(1)})\cdot(1-b_z^{(2)}) + \\
&\quad \alpha_1(x_1)\cdot b_z^{(1)}\cdot(1-b_z^{(2)}) + \\
&\quad \alpha_2(x_1)\cdot(1-b_z^{(1)})\cdot b_z^{(2)} + \\
&\quad \alpha_3(x_1)\cdot b_z^{(1)}\cdot b_z^{(2)} \\
p(x_2|z) &= \alpha_0(x_2)\cdot(1-b_z^{(1)})\cdot(1-b_z^{(2)}) + \\
&\quad \alpha_1(x_2)\cdot b_z^{(1)}\cdot(1-b_z^{(2)}) + \\
&\quad \alpha_2(x_2)\cdot(1-b_z^{(1)})\cdot b_z^{(2)} + \\
&\quad \alpha_3(x_2)\cdot b_z^{(1)}\cdot b_z^{(2)}
\end{aligned}$$

After some simple algebraic manipulation of Equations (5), we still have to deal with products of boolean variables. The procedure is straightforward: If $b_1, b_2, \ldots, b_r$ are boolean variables, then the product $\prod_i b_i$ can be replaced by the continuous variable y, with additional constraints:

$$\begin{aligned}
0 \leq \quad & y \leq 1 \\
& y \leq b_i, \text{ for all } i \\
\sum_i b_i - r + 1 \leq \quad & y
\end{aligned} \tag{6}$$

The number of boolean optimization variables in the integer programming version is $O(\log_2 \prod_{X\in V} c_X)$, where c_X is the number of categories of the random variable associated to node X. Thus, the reformulation to an integer programming problem is performed by running the `Bilinear-Transformation` algorithm together with the linearization step. The linearization inserts a logarithmic number of new constraints for each constraint generated by `Bilinear-Transformation` (when using the ideas of Expressions (5) and (6)). The number of new boolean optimization variables is small and does not increase the overall complexity of the reformulation. For polytrees, we still have a polynomial time procedure.

5 Computational results

To illustrate the behavior of our methods, we present two sets of experiments. First we deal with test sets containing multi-connected networks (randomly generated using the BNGenerator software [17] or using the topology of the Alarm network [1]). Latter we treat randomly generated polytrees. In each network we perform a belief updating inference with a predefined variable (we have chosen the most challenging variables).

Table 1 shows results of `Bilinear-Transformation` followed by the linearization step for four different network. Rows present type of the network, total number of nodes, number of nodes involved in the inference, number of vertices in the credal sets, resulting continuous optimization variables, resulting boolean variables and resulting optimization constraints. All the tests were done by transforming inferences in multi-connected credal networks into integer programming problems. The chosen inferences represent the most challenging inference for each network. We processed networks with different variables (binary and ternary), and different sizes of credal sets per node of the network. Note that the size of resulting problems (specially the number of boolean variables to optimize) is large. Existing exact optimization solvers usually can not handle such large number of boolean variables, but approximation ideas are still possible. As we can see, the number of integer variables is too high for processing such networks.

Restricting our attention to polytrees, Table 2 presents twenty polytree-shaped credal networks. They have ternary variables and at most three vertices by locally and separately specified credal set. Rows present name of network, total number of nodes, number of nodes involved in the inference, generated continuous optimization variables, generated boolean variables, generated constraints, time for solving the integer programming problem and number of branch-and-bound nodes evaluated by the solver.

Analyzing Table 1 (multi-connected networks) and Table 2 (polytree networks), we see that the algorithm could generate much smaller problems in the latter case. That happens because of the relationship between tree-width and path-width of a polytrees: they are almost the same.

Because the integer programming reformulation is usually less dependant on some convergence criteria and numerical problems than nonlinear programming techniques (such as multilinear programming [8, 9]), the integer programming reformulation achieves good performance together with reliable results. All tests

Network topology	Nodes	Active Nodes	Vertices by credal set	Continuous variables	Boolean variables	Constraints
Dense binary	10	10	2	1523	120	1523
Dense ternary	10	10	3	3954	202	3954
Alarm network	37	24	2	34537	161	34351
Alarm network	37	24	4	51293	322	65075

Table 1: Size of integer programming problems generated from some multi-connected credal networks.

were performed on a Intel Xeon 2.8Ghz (4MB of L2 cache) with 4GB of RAM memory. The integer programming problems were solved using the AMPL modeling language [15, 16] and the CPLEX solver.

6 Conclusion

We have discussed in this paper a new idea for inferences in credal networks. The main contribution is the use of bilinear and integer programming techniques. Although many authors have suggested and worked with multilinear programming as a possible approach to inference, as far as we know no investigation or implementation of bilinear and/or integer programming reformulations have been conducted.

Results produced in our experiments seem promising and surpass existing exact algorithms for inference in polytree-shaped credal networks with respect to the number of network variables that could be dealt [5, 9]. Although there is a polynomial time algorithm for inferences in binary polytrees [13], the problem is `NP-Complete` in general polytrees [10]. So our reformulation is a new idea to address this hard problem, with good empirical performance. Furthermore, integer programming produces outer bounds based on linear programming relaxations, which can be used for approximate procedures. Known approximate techniques, such as cutting planes, can be applied.

Other multilinear programming techniques could certainly be investigated in future work, perhaps combining some ideas from multilinear programming with integer programming. As it happens with multilinear programming, integer programming will fail for large networks; in this case approximate inference is the natural solution. The reformulations presented in this paper also contribute in that direction, as approximations for bilinear and integer programming problems are well studied in the literature.

Acknowledgements

This work has been supported by FAPESP grant 2004/09568-0; the second author is partially supported by CNPq grant 3000183/98-4.

References

[1] I. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *European Conf. on Artificial Intelligence in Medicine*, pages 247–256, Berlin, 1989. Springer-Verlag.

[2] A. Cano, J. E. Cano, and S. Moral. Convex sets of probabilities propagation by simulated annealing. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 978–983, Paris, France, 1994.

[3] J. Cano, M. Delgado, and S. Moral. An axiomatic framework for propagating uncertainty in directed acyclic networks. *International Journal of Approximate Reasoning*, 8:253–280, 1993.

[4] I. Couso, S. Moral, and P. Walley. A survey of concepts of independence for imprecise probabilities. *Risk, Decision and Policy*, 5:165–181, 2000.

[5] F. G. Cozman. Credal networks. *Artificial Intelligence*, 120:199–233, 2000.

[6] F. G. Cozman. Separation properties of sets of probabilities. In In *Conference on Uncertainty in Artificial Intelligence*, pages 107–115, San Francisco, 2000. Morgan Kaufmann.

[7] J. C. F. da Rocha and F. G. Cozman. Inference with separately specified sets of probabilities in credal networks. In *Conference on Uncertainty in Artificial Intelligence*, pages 430–437, San Francisco, 2002. Morgan Kaufmann.

[8] J. C. F. da Rocha, F. G. Cozman, and C. P. de Campos. Inference in polytrees with sets of probabilities. In *Conference on Uncertainty in Artificial Intelligence*, pages 217–224, New York, 2003. Morgan Kaufmann.

[9] C. P. de Campos and F. G. Cozman. Inference in credal networks using multilinear programming. In *Second Starting AI Researcher Symposium*, pages 50–61, Valencia, 2004. IOS Press.

Network name	Network nodes	Active nodes	Continuous variables	Boolean variables	Constraints	Solution time (sec)	B&B nodes
poly50v0	50	16	2101	40	2076	6.976	791
poly50v1	50	17	3775	84	3967	7.965	258
poly50v2	50	15	511	50	532	0.187	23
poly50v3	50	15	3244	42	2911	60.224	2743
poly50v4	50	25	1779	33	1996	3.988	977
poly50v5	50	18	1533	38	1582	2.429	935
poly50v6	50	17	1049	41	1169	2.598	1456
poly50v7	50	19	768	20	912	2.332	1307
poly50v8	50	15	2459	53	2569	12.386	708
poly50v9	50	16	775	37	829	0.549	314
poly100v0	100	22	3435	68	3324	48.772	3191
poly100v1	100	28	4101	95	4511	1095.353	89560
poly100v2	100	23	1179	74	1255	628.776	333809
poly100v3	100	21	2540	40	2745	296.844	41915
poly100v4	100	24	1787	63	1837	5.751	3695
poly100v5	100	25	1067	39	1150	8.006	3428
poly100v6	100	25	3340	51	3474	372.833	21352
poly100v7	100	26	672	60	725	0.458	232
poly100v8	100	28	773	47	897	25.146	28860
poly100v9	100	23	10635	57	9853	576.663	2880

Table 2: Tests with random polytree networks.

[10] C. P. de Campos and F. G. Cozman. The inferential complexity of Bayesian and credal networks. In *International Joint Conference on Artificial Intelligence*, pages 1313–1318, 2005.

[11] C. P. de Campos and F. G. Cozman. Computing lower and upper expectations under epistemic independence. *International Journal of Approximate Reasoning*, 44:244–260, 2007.

[12] L. de Campos and S. Moral. Independence concepts for convex sets of probabilities. In *Conference on Uncertainty in Artificial Intelligence*, pages 108–115, San Francisco, 1995. Morgan Kaufmann.

[13] E. Fagiuoli and M. Zaffalon. 2U: An exact interval propagation algorithm for polytrees with binary variables. *Artificial Intelligence*, 106(1):77–107, 1998.

[14] U. Feige, M. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In *ACM Symposium on Theory of computing*, pages 563–572, New York, NY, USA, 2005. ACM Press.

[15] R. Fourer, D. M. Gay, and Brian W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36:519–554, 1990.

[16] R. Fourer, D. M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, Brooks/Cole Publishing Company, 2002.

[17] J. S. Ide and F. G. Cozman. Random generation of Bayesian networks. In *Brazilian Symposium on Artificial Intelligence*, pages 366–375, Recife, 2002. Springer-Verlag.

[18] I. Levi. *The Enterprise of Knowledge*. MIT Press, Cambridge, Massachusetts, 1980.

[19] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.

[20] N. Robertson and P. D. Seymour. Graph minors – II. Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.

[21] N. Robertson and P.D. Seymour. Graph minors – I. Excluding a forest. *J. Combinatorial Theory Series B*, 35:39–61, 1983.

[22] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.

[23] P. Walley. *Statistical Reasoning with Imprecise Probabilities*. Chapman and Hall, London, 1991.