Inference in Credal Networks with Branch-and-Bound Algorithms *

JOSÉ CARLOS FERREIRA DA ROCHA
Escola Politécnica, Universidade de São Paulo and Universidade Estadual de Ponta Grossa, Brazil

FABIO GAGLIARDI COZMAN
Escola Politécnica, Universidade de São Paulo, Brazil

Abstract

A credal network associates sets of probability distributions with directed acyclic graphs. Under strong independence assumptions, inference with credal networks is equivalent to a signomial program under linear constraints, a problem that is NP-hard even for categorical variables and polytree models. We describe an approach for inference with polytrees that is based on branch-and-bound optimization/search algorithms. We use bounds generated by Tessem's A/R algorithm, and consider various branch-and-bound schemes.

Keywords

Credal networks, strong independence, probability intervals, inference, branch-and-bound algorithms.

1 Introduction

A credal network provides a representation for imprecise probabilistic knowledge through direct acyclic graphs (DAGs) [1]. In this formalism, each node in a DAG represents a random variable, and each variable is associated with convex sets of probability distributions. The structure of the graph indicates relations of probabilistic independence between variables. In this paper we interpret independence relations as statements of strong independence [1, 2].

A credal network can be viewed as a Bayesian network [3] with relaxed numerical statements. Credal networks can be used to study the robustness of Bayesian networks [4], or to represent vague or incomplete probability statements.

*The first author is supported in part by CAPES. The work has received substantial support from HP Labs through "Convênio Redes Bayesianas para Aprendizado".

An *inference* with a credal network is the computation of upper and lower probability values for each category of a *query* variable. This computation is NP-hard even for polytrees [5], and it can be viewed as a signomial program under linear constraints [6]. Exact and approximate inference algorithms have been proposed in the literature, but no algorithm can handle large credal networks exactly.

In this article we propose new algorithms for inferences in polytrees. The idea is to use branch-and-bound search/optimization techniques to produce inferences. We explore Tessem's A/R algorithm [7] as a bound generation mechanism. We show how this approach can generate exact and approximate inferences, illustrating the main ideas with of examples.

The organization of the text is as follows. Sections 2 and 3 present a summary of credal networks and branch-and-bound techniques. Section 4 describes how exact and approximate inference can be performed with branch-and-bound techniques and the A/R algorithm. Section 5 shows how exact and approximate techniques can be combined through decomposition of networks. Section 6 discusses the proposed algorithms and results.

2 Credal sets, credal networks and inference

A convex set of probability distributions is called a credal set [8].[1] Denote the probability density of a categorical random variable X by $p(X)$. A credal set for X is denoted by $K(X)$; we assume that every credal set has a finite number of vertices. We can represent such a set just enumerating its vertices. A conditional credal set is a set of conditional distributions. We obtain a conditional credal set applying Bayes rule to each distribution in a joint credal set.

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. A collection of marginal and conditional credal sets can have more than one extension. In this paper we are always interested in computing the largest possible extension for a given collection of marginal and conditional credal sets.

Credal networks associate credal sets with a direct acyclic graph. In analogy to Bayesian networks, in a credal network every node of a directed acyclic graph is associated with a variable,[2] and every variable is associated with a collection of *local* credal sets $K(X|\mathrm{pa}(X))$, where $\mathrm{pa}(X)$ denotes the parents of variable X in the graph. That is, a node stores the credal sets

$$\{K(X|\mathrm{pa}(X) = \pi_1), \ldots, K(X|\mathrm{pa}(X) = \pi_m)\},$$

where $\{\pi_1, \ldots, \pi_m\}$ are the instances of $\mathrm{pa}(X)$. A root node has only one credal set associated with it.

[1] We deal only with convex sets.

[2] To simplify the text, we represent a node and its variable with the same symbol.

The sets $K(X|\mathrm{pa}(X))$ are called *separately specified* when there is no relationship between them for different values of $\mathrm{pa}(X)$. In this paper we assume that local credal sets are always separately specified.

The basic assumption in a credal network is that every variable is independent of its nondescendants nonparents given its parents. Obviously the import of such a condition depends on which concept of independence for credal sets is adopted [1, 2, 9]. In this paper we adopt the concept of *strong independence*: two variables X and Y are strongly independent when every extreme point of $K(X,Y)$ satisfies stochastic independence of X and Y (that is, each vertex $p(X,Y) \in K(X,Y)$ satisfies $p(X|Y) = p(X)$ and $p(Y|X) = p(Y)$ for all possible conditioning values) [10].

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, such that the vertices are combined as follows [1]:

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

Figure 1 shows the structure of a credal network that is latter used in examples.

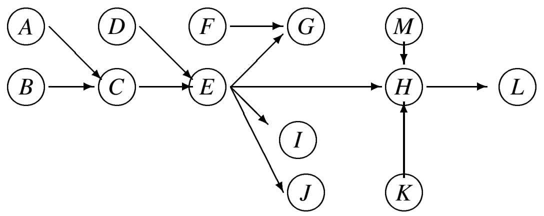

Figure 1: A polytree credal network.

An *inference* in a credal network is the computation of tight bounds for probability values in an extension of the network. These bounds are called *upper* and *lower* probabilities. 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.

Algorithms for exact inference in strong extensions can be found in [1, 5, 11, 12]. The only known polynomial algorithm for strong extensions is the 2U algorithm, which processes polytrees with binary variables [13]. In general, exact inference in credal networks is a NP-hard problem (even for polytrees), so approximate algorithms are a natural solution. We distinguish *outer* approximations from *inner* ones; the former are produced when the correct interval between lower and upper probabilities is enclosed in the approximate interval; the latter approximations are produced when the correct interval encloses the approximate interval. Outer approximations can be found in [7, 14, 15], and inner approximate algorithms can be found in [4, 16, 17]. Generally speaking, inner algorithms are obtained by local optimization methods.

In this paper we are interested in inferences with strong extensions. The difficulty faced by inference algorithms is the potentially enormous number of vertices that a strong extension can have — even a relatively small network can dwarf the best exact algorithms. Consider the following example, taken from [5]:

Example 1 Consider a network with four variables X, Y, Z and W; W is the sole child of X, Y and Z, and there are no other arrows in the network. Suppose that all variables have three values and that every local credal set has only three vertices. The vertices of the strong extension $K(X,Y,Z,W)$ factorize as $p(W,X,Y,Z) = p(W|X,Y,Z)\,p(X)\,p(Y)\,p(Z)$. Now, W is associated with 27 credal sets; therefore there are 3^{27} ways to combine the vertices of these credal sets. These 3^{27} vertices must be combined with every combination of vertices of $K(X)$, $K(Y)$ and $K(Z)$. So, the potential number of vertices in $K(X,Y,Z,W)$ is 3^{30}.

We note that inference in credal networks is an optimization problem. Consider the computation of an upper probability:

- The goal is to find a distribution $p(X_i|\mathrm{pa}(X_i))$ in $K(X_i|\mathrm{pa}(X_i))$, for each variable X_i, so as to maximize the probability value $p(X_q|\mathbf{X}_E)$.
- The objective function $p(X_q|\mathbf{X}_E)$ is a fraction of multilinear expressions:

$$p(X_q|\mathbf{X}_E) = \frac{\sum_{X_1,\dots,X_n\setminus\{X_q,\mathbf{X}_E\}} \prod_i p(X_i|\mathrm{pa}(X_i))}{\sum_{X_1,\dots,X_n\setminus\mathbf{X}_E} \prod_i p(X_i|\mathrm{pa}(X_i))}.$$

- The maximization is subject to linear constraints, given our assumption of credal sets with finitely many vertices.

This maximization problem belongs to the field of *signomial* programming [6], as observed independently by [4, 12, 18]. Signomial programs are generally solved dividing the feasible set ("branching" on various subsets) and obtaining outer approximations ("bounding" the objective function in each subset) [6, 19]. That is, signomial programming is solved by branch-and-bound procedures. The great advantage of signomial programming over more general optimization problems is that it is possible to obtain bounds for signomial programs using geometric programming — a well establish field that can be tackled efficiently through convex programming [20]. However, direct application of geometric programming bounds to strong extensions seems to face difficulties. First, the inference problem is an "implicit" signomial programming, as the objective function is encoded in the graph through Expression (1); each combination of variables in the credal network would be a maximizer in the geometric program. Second, and perhaps more importantly, the "degree of difficulty" of a geometric program depends on the number of polynomial terms in the program — note that Expression (1) summarizes a large number of terms.

In this paper we adopt the basic idea of branching and bounding to compute lower and upper probabilities, but instead of relying on properties of geometric programming, we use bounds that have been specifically developed for strong extensions.

3 Branch-and-bound search and optimization

Branch-and-bound techniques appear in artificial intelligence, optimization and constraint satisfaction [21, 22]. The basic purpose of a branch-and-bound algorithm is to optimize a function. For example, take a problem P stated as:

$$\begin{array}{lll} (P) & \max f(w) & \\ & \text{s.t.} & g(w) \leq 0, w \in \mathbf{W}, \end{array}$$

where $\mathbf{W} \subseteq \Re^n$, f is a real valued function, and the image of g is contained in $\Re^m$. A branch-and-bound technique is suitable for P whenever it is possible to divide P in sub-instances that are easier to solve or approximate than P itself, and such that the solution for P is present in one of these sub-instances [23, 24]. Additionally, a branch-and-bound technique requires a bound r (overestimation) for the solution of P. This upper bound is usually obtained from a relaxed version of P, indicated by R. Obviously, R must be easier and faster to solve than P, and must give a good approximation for P. The relaxed bound for P is denoted by $r(P)$.

In our implementation we use the following version of branch-and-bound [25]; several variants exist for it [23].

Algorithm 1 - Depth-first branch-and-bound

- *Input:* a problem P.
- *Output:* the value of $\max f(w)$, denoted by $\bar{p}$.

1. Initialize $\hat{p}$ with a small value (necessarily smaller than $\bar{p}$).
2. If $\mathbf{W}$ contains a single value w then: update $\hat{p}$ with $f(w)$ when $f(w) > \hat{p}$;
3. else:
   (a) using decomposition, obtain a list L of sub-instances of P; each sub-instance is denoted by P_h and has feasible region $\mathbf{W}_h$.

(b) for each P_h do

i. if $\mathbf{W}_h$ is feasible in the original problem and $r(P_h) > \hat{p}$, call recursively depth-first branch-and-bound over P_h.

4. Take the last $\hat{p}$ as $\bar{p}$.

This algorithm can be viewed as a search in a tree where the root node contains P and descendant nodes contain sub-instances of P. The leaf nodes contain problems that can be exactly solved. When a leaf node l is reached, the value for f at l is computed; if this value is the largest one up to that moment, it is retained. Non-leaf nodes are processed by relaxing the original problem and producing bounds. Every non-leaf node is expanded by decomposition into sub-instances, as long as its bound is larger than the current best value.

4 Branch-and-bound inference in strong extensions

This section contains the central ideas in this paper. We use a branch-and-bound procedure where

- branching occurs at every vertex of credal sets, and
- bounding is achieved by Tessem's A/R algorithm [7].

Given a query variable X and a credal network $\mathcal{N}$, a single run of the branch-and-bound procedure computes the lower or upper probability for a single state of X, denoted by x.

The first step is to discard variables that are not used to compute the inference; this can be done using d-separation [26]. The resulting network is denoted by $\mathcal{N}_0$.

4.1 Branching

The root node in the branch-and-bound search tree is $\mathcal{N}_0$. The root node $\mathcal{N}_0$ is then divided into several simpler credal networks $\{\mathcal{N}_{01}, \ldots, \mathcal{N}_{0q}\}$. Each one of these networks is obtained as follows. We select one credal set in $\mathcal{N}_0$, and produce as many networks as there are vertices in this credal set — each network is associated with a single vertex of the selected credal set. This decomposition procedure is then applied recursively, following the branch-and-bound algorithm. At each step, a credal set is "expanded". Using this decomposition strategy, a leaf node contains a Bayesian network, obtained by a particular selection of vertices in all credal sets in the credal network. When a leaf node is reached, a variable elimination algorithm is used to perform inference in the Bayesian network defined by the leaf [27]. Such an algorithm produces a probability value $p(x|\mathbf{X}_E)$; if $p(x|\mathbf{X}_E)$ is greater than the current maximum probability, the latter value is updated.

We always select the non-expanded credal set nearest to the queried variable, but we always keep the query variable to be processed at last (a similar criterion is used in [28] to deal with partial evaluation of belief nets). We have tried several criteria for the selection of the credal sets that are expanded, and we found that the procedure just described is quite appropriate.

4.2 Bounding

For non-leaf nodes in the search tree, we run the A/R algorithm as a relaxation of exact inference [7], because this algorithm produces outer bounds rather quickly. The A/R algorithm focuses on polytrees, even though it can be modified to handle more general networks [14].

The A/R algorithm assumes that every credal set is approximated by a collection of probability intervals. So we must convert the credal network to an interval-based Bayesian network (conditional probability tables contain intervals). Obviously the replacement of credal sets by probability intervals introduces potential inaccuracies into the process.

The A/R algorithm mimics the dynamics of Pearl's belief propagation algorithm [3]. The functions λ, π and the messages used in BP are still defined with identical purposes, but they are now interval-valued functions. The idea is to manipulate these intervals using interval arithmetic and two additional techniques called by Tessem *annihilation* and *reinforcement*.

We can understand the basic ideas in the A/R algorithm by looking at the computation of the interval-valued message $\pi(X)$ — this message is computed at a node X with parents $Y_0,\ldots,Y_k$. Consider then the computation of $\pi_*(x_j)$, the lower bound of $\pi(x_j)$ for a particular value x_j:

1. Construct a interval-valued function $\beta(Y_0,\ldots,Y_k)$ by interval-multiplication of the messages $\pi_X(Y_i)$ received by X (these messages are also interval-valued).

2. Construct a distribution $p(Y_0,\ldots,Y_k)$ that is consistent with the intervals in $\beta(Y_0,\ldots,Y_k)$, such that $p(Y_0,\ldots,Y_k)$ minimizes the sum

$$\sum_{Y_0,\ldots,Y_k} \underline{p}(x_j|Y_0,\ldots,Y_k)\, p(Y_0,\ldots,Y_k)\,,$$

   where $\underline{p}(x_j|Y_0,\ldots,Y_k)$ is the lower value for $p(x_j|Y_0,\ldots,Y_k)$; the minimum of the sum is $\pi_*(x_j)$.

These operations are efficient because it is not hard to find $p(Y_0,\ldots,Y_k)$ in step 2: sort $\underline{p}(x_j|Y_0,\ldots,Y_k)$ in increasing order, and distribute probability mass (consistently with $\beta(Y_0,\ldots,Y_k)$) from the smallest to the largest value of $\underline{p}(x_j|Y_0,\ldots,Y_k)$. The same operations can be adapted to compute the upper bound $\overline{\pi}^*(x_j)$.

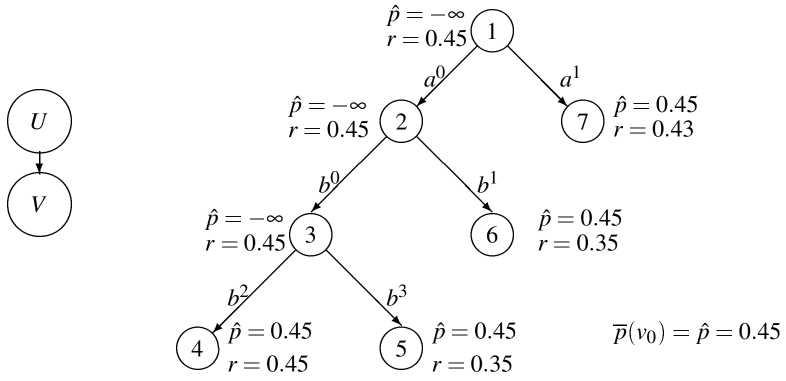

Figure 2: An example of branch-and-bound based inference. Left: a simple credal network, where $K(U)$ is the convex hull of $\{a^0, a^1\}$, with $a^0 = (0.5, 0.5)$ and $a^1 = (0.3, 0.7)$; $K(V|u_0)$ is the convex hull of $\{b^0, b^1\}$, with $b^0 = (0.5, 0.5)$ and $b^1 = (0.3; 0.7)$; $K(V|u_1)$ is the convex hull of $\{b^2, b^3\}$, with $b^2 = (0.4, 0.6)$ and $b^3 = (0.2, 0.8)$. Right: Search tree for computation of $\overline{p}(v_0)$.

The A/R algorithm prescribes similar operations for computation of $\lambda_X(Y_i)$ and $\pi_{Z_i}(X)$ (where Z_i is a child of X). The function $\lambda(X)$ is obtained by direct interval multiplication. Finally, the algorithm uses annihilation or reinforcement operations to "normalize" the functions $\lambda_X(Y_i)$, $\pi_{Z_i}(X)$, and the product $\pi(X)\lambda(X)$ — "normalization" means simply computing bounds that take into account the fact that probability distributions add up to one.

In our branch-and-bound procedure, the deeper a node is in the search tree, the more point probabilities are manipulated by the A/R algorithm.

Example 2 Figure 2 shows a very simple network and the the basic steps of our branch-and-bound algorithm when computing $\overline{p}(v_0)$. Nodes in the search tree represent credal networks; the numbering inside nodes indicates the order in which nodes are visited. The value r is obtained by the A/R algorithm. Close to each arc in the search tree we indicate which vertex (and for which credal set) was expanded.

4.3 Experiments

We have implemented the branch-and-bound scheme in a Java program, using Pentium IV machines to run tests. We ran experiments with networks containing

Table 1: Cost for exact inference for E in the network of Figure 1.

# states per variable	# vertices per credal sets	Potential size of the strong extension	Visited nodes (mean)	Samples (networks)
03	02	2^{21}	5499	35
03	03	3^{21}	284912	10
04	02	2^{35}	559255	10

variables with three and four states. Each configuration was tested against several randomly generated credal nets [29]. Experiments discussed in this section have no evidence ($\mathbf{X}_E = \emptyset$); this restriction simplifies the presentation with no loss in generality.

We have observed that the size of the search tree explored by branch-and-bound is usually a small fraction of the potential vertices of the strong extension. As an example, consider the network in Figure 1. Table 1 shows relevant results for query variable E, indicating the number of states for variables, the number of vertices for each credal set in the network, and the potential number of vertices of the strong extension. The table indicates how many networks of each type were tested, and the mean number of visited nodes during branch-and-bound. Note the enormous difference between the potential number of vertices and the number of effectively expanded nodes.

As another instructive example, we applied the branch-and-bound scheme to the network described in Example 1. We tested thirty randomly generated credal networks with the same structure and different credal sets; in each one of them we computed the lower and upper probabilities for w_0. These sample networks had ternary variables and three distributions in each credal set. The branch-and-bound search was always able to quickly compute the exact inference, on average exploring 243 nodes per inference.

Consider another example. We took the polytree structure of the well-known Bayesian network called "Car Starts"[3] and set all of its variables as ternary. We assumed that in practice it would be unusual to have credal sets associated to all variables in a credal network — some distributions could be obtained with greater precision, and in any case the specification of dozens of credal sets is not an easy matter. We therefore introduced credal sets in all root nodes and in the node called $BatteryState$, using ε-contaminated models with $\varepsilon = 0.2$ [30]. The resulting strong extension has 3^{18} potential vertices (about 387 million potential vertices). We ran branch-and-bound inference for all states of the variable $Starts$ and obtained exact values after evaluating 1,139,717 nodes (less than 0.3% of

[3]Microsof Research: http://www.research.microsoft.com/research/dtg/bnformat/autoxml.html.

Table 2: The probability error in underestimated approximate reasoning for E in the network of Figure 1.

# states per variable	# vertices per credal set	Fixed number of visited nodes	Mean relative error
03	03	150000	0.0013
03	03	30000	0.0067
04	02	200000	0.0061
04	02	50000	0.0097

the number of potential vertices were explored). An interesting test was made with the branching strategy. We ran the same inference using a "reverse ordering" for branching; that is, we first expanded the credal sets that were farthest away from the query node. Using this strategy, the branch-and-bound algorithm found the exact values after expanding 4,546,943 nodes. This simple test reinforces the intuition that the most relevant probability values in an inference are the values that are "close" to the query variable.

It is also possible to look at the branch-and-bound scheme not only as an exact algorithm, but also as an algorithm that can be stopped at any time to generate approximate results. We tested this idea by running the branch-and-bound algorithm with a fixed number of nodes. Table 2 shows the mean relative error in inferences (each row is the mean of ten random networks). The relative error is computed using the approximate and the exact values for $\overline{P}(E = e_0)$.

5 Inference with network fragments

If the credal network $\mathcal{N}$ is large, it may not be possible to run the branch-and-bound algorithm to optimality. In this section we propose strategies to handle such problems. The basic idea is to divide the credal network in parts and to run branch-and-bound in these sub-networks, in some suitable order. We illustrate this idea through an example.

Consider the network in Figure 1, with ternary variables and two vertices in each credal set. Suppose that we want to compute exact lower and upper probabilities for variable G and that our space and time constraints allow us to perform an exact inference just for E, but not for G. We then run branch-and-bound and obtain lower and upper probabilities for E. In a particular instance of the network shown in Figure 1, we obtained $p(e_0) \in [0.199; 0.587]$, $p(e_1) \in [0.084; 0.375]$ and $p(e_2) \in [0.212; 0.604]$. We can easily generate the largest credal set that is consistent with these intervals. We obtain $K(E)$ defined by the vertices

$$\{(0.413; 0.375; 0.212), (0.312; 0.084; 0.604), (0.587; 0.084; 0.329),$$

$$(0.199;0.197;0.604),(0.587;0.201;0.212),(0.199;0.375;0.426)\}.$$

Now we can remove E and its antecedents from the network, and replace E by a new node E' that has the marginal credal set of E as its marginal credal set. The transformed network is displayed in Figure 3. We then run exact branch-and-bound based inference for G, obtaining the intervals $p(g_0) \in [0.091;0.447]$, $p(g_1) \in [0.157;0.564]$ and $p(g_2) \in [0.208;0.591]$. Incidentally, we computed the same inferences with an exhautive algorithm in the JavaBayes system[4] and got the same values.

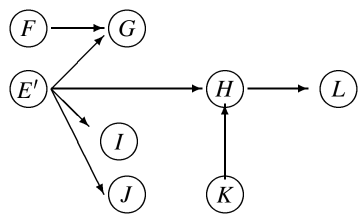

Figure 3: Transformed polytree credal network.

If inferences in the transformed credal network are still unfeasible, we can run an approximate inference algorithm in the transformed credal network. Consider running Tessem's algorithm in the network in Figure 3. We obtain the intervals $p(g_0) \in [0.053,0.502]$, $p(g_1) \in [0.116,0.663]$ and $p(g_2) \in [0.128,0.644]$.

In closing, we note that Tessem's algorithm alone in the complete example network produced the intervals $p(g_0) \in [0.040,0.524]$, $p(g_1) \in [0.106,0.698]$ and $p(g_2) \in [0.097,0.667]$.

6 Discussion

Any branch-and-bound algorithm is highly dependent on the quality of the bounds it employs. We have found that Tessem's bounds, while fast to compute and reasonably accurate, are quite wide — usually the search tree is expanded to a large depth before some of its branches are discarded. To give an example, in the computation of inferences for variable E in our samples with ternary variables and three vertices, the branch-and-bound algorithm explored the search tree almost completely down to levels 12 or 13 (the complete search tree has 21 levels).

As an aside, we have also implemented a breadth-first version of branch-and-bound [22], but we have found that the need to store the expanded frontier in such algorithms makes them unfeasible. Breadth-first branch-and-bound will only become a reality if better bounds than Tessem's are found.

[4]Free software, site http://www.cs.cmu.edu/ javabayes.

Generally speaking, we can say that the branch-and-bound algorithm needs to explore a tiny fraction of potential vertices of the strong extension, and is faster than the best existing exact algorithms [5]. For really small credal networks (with a few thousand potential vertices in the strong extension), the overhead of branching and bounding can be significant, and in those cases enumeration algorithms may be faster.

Clearly, the branch-and-bound algorithm with Tessem bounds cannot cope with arbitrarily large problems, and it can face difficulties even in seemingly simple situations. In the network in Figure 1, inferences for variable L could not be found exactly, even after extensive tests.

7 Conclusion

This paper can be best understood as proposing a *family* of solutions for inference in strong extensions, using branch-and-bound algorithms as a unifying idea in such solutions. We have restricted ourselves to polytrees, but branch-and-bound techniques can be used for general inference; we have stressed the use of Tessem bounds, but any bounding scheme can be used.

We believe that our ideas are the first explicit formulation and implementation of inference in credal networks as a search procedure that runs to optimality. Branch-and-bound techniques are rather suitable for this purpose; the experiments show that inference with branch-and-bound and Tessem bounds is a definite improvement over existing algorithms.

We also would like to emphasize the possibility that a network is processed in pieces, using different levels of accuracy in each one of the partial inferences. Such a strategy seems to be appropriate for large networks. Our future research will be focused on developing and implementing general algorithms for decomposing networks and processing fragments with different strategies.

References

[1] Cozman, F.G.: Credal Networks. Artificial Intelligence **120** (2000) 199–233

[2] Couso, I. Moral, S., Walley, P.: Examples of Independence for Imprecise Probabilities. Proc. of the 1st ISIPTA, Ghent Belgium (1999) 121–130

[3] Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Networks of Plausible Inference. Morgan Kaufmann Publishers, San Mateo CA (1988)

[4] Cozman, F.G.: Robustness Analysis of Bayesian Networks with Local Convex Sets of Distributions. Proc. of the 13th Annual Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, San Francisco, CA, (1997) 393-405

[5] Rocha, J.C.F.; Cozman, F.G.: Inference with Separately Specified Sets of Probabilities in Credal Networks. Proc. of the 18th Annual Confence on Uncertainty in Artificial Intelligence, Morgan Kaufmann, San Francisco, CA , (2002) 430-437

[6] Avriel, M.: Advances in Geometric Programming, Plenum Press, New York, 1980.

[7] Tessem, B.: Interval Probability Propagation. Int. Journal of Approximate Reasoning **7**, (1992) 95–120

[8] Levi, I. The Enterprise of Knowledge. MIT Press, Cambridge, Mass, (1980)

[9] Campos, L. M. de, Moral, S.: Independence Concepts for Convex Sets of Probabilities. Proc. of the XI Conference on Artificial Intelligence, Morgan Kaufmann, San Francisco, CA, (1995) 108–115

[10] Walley, P.: Statistical Reasoning with Imprecise Probabilities. Chapman and Hall, London, (1991)

[11] Cano, J.E., Delgado, M., Moral, S.: An Axiomatic Framework for the Propagation of Uncertainty in Directed Acyclic Networks. Int. Journal of Approximate Reasoning **8**, (1993) 253–280.

[12] Zaffalon, M.: Inferenze e Decisioni in Condizioni di Incertezza con Modelli Grafici Orientati. Ph.D. Thesis, Università di Milano, Milan, Italy, (1997) (in Italian)

[13] Fagiuoli, E., Zaffalon, M. (1998).: 2U - An Exact Interval Propagation Algorithm for Polytrees with Binary Variables. Artificial Intelligence **106**(1), 77-107

[14] Ha, V.A. *et al*: Geometric Foundations for Interval-Based Probabilities. Annals of Mathematics and Artificial Intelligence, Vol.24, **1-4** (1998) 1–21

[15] Cano, A., Moral, S.: Using Probabilistic Trees to Compute Marginals with Imprecise Probabilities. TR-DECSAI-00-02-14, University of Granada, (2000)

[16] Cano, A., Moral, S.: A Genetic Algorithm to Approximate Convex Sets of Probabilities. 7th Int. Conf. IPMU-96, (Paris, France, July, 1994), (1994) pp 859–864

[17] Cano, A., Moral, S.: Convex Sets of Probabilities Propagation by Simulated Annealing. 5th Int. Conf. IPMU-94, (Paris, France, July, 1994), (1994) pp 4–8

[18] Andersen, K.A., Hooker, J.N.: Bayesian Logic. Decision Support Systems **11**, (1994) 191-210.

[19] Duffin R.J., Peterson, E.L.: Geometric Programming with Signomials. Journal of Optimization Theory and Applications, **11**(1), (1973) 3–35"

[20] Duffin, R.J., Peterson, E.L., Zener, C.: Geometric Programming, Theory and Application. John Willey and Sons, New York, (1967)

[21] Norvig, P., Russell, S.: Artificial Intelligence: a Modern Approach. Prentice Hall, Englewoods, (1995)

[22] Bertsekas, D.P.: Dynamic Programming, Deterministic and Stochastic Models. Prentice Hall, Englewoods, (1987)

[23] Papadimitriou, C., Steiglitz, I.: Combinatorial Optimization, Algorithms and Complexity. Prentice-Hall, Englewood Cliffs, New Jersey, (1982)

[24] Sahinidis, N.V.: BARON, Branch and Reduce Optimization Navigator, User's manual 4.0. University of Illinois at Urbana-Champaign, (2000)

[25] Preiss, B.R.: Data Structures and Algorithms with Object-Oriented Design Patterns in Java. Wiley, New York, (2000)

[26] Cozman, F.G.: Irrelevance and Independence in Quasi-Bayesian Networks. Proc. XIV Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann, San Francisco, CA, (1998) 86–96

[27] Cozman, F.G.: Generalizing Variable Elimination in Bayesian Networks. Workshop on Probabilistic Reasoning in Artificial Intelligence, Editora Tec Art, São Paulo, (2000) 27–32

[28] Draper, D.L., Hanks, S.: Localized Partial Evaluation of Belief Networks. XV Conference on UAI, (1995) 170-177

[29] Ide, J.S., Cozman, F.G.: Random Generation of Bayesian Networks. Proc. of the XVI Braziliam Symposium on Artificial Intelligence, Springer-Verlag, (2002)

[30] Berger, J.O.: Statistical Decision Theory and Bayesian Analysis. Springer, Berlin, (1985)

José Carlos Ferreira da Rocha is a PhD student at the Engineering School (Escola Politécnica), University of São Paulo, and a teaching assistant at UEPG, Deinfo, Ponta Grossa, PR, Brasil, CEP 84030-900 E-mail: jrocha@uepg.br

Fabio Gagliardi Cozman is with the Engineering School (Escola Politécnica), University of São Paulo, São Paulo, SP, Brazil, CEP 05508-900. E-mail: fgcozman@usp.br