Logical Credal Networks
Supplementary Material

Radu Marinescu
IBM Research
radu.marinescu@ie.ibm.com

Haifeng Qian
AWS AI Labs
qianhf@amazon.com

Alexander Gray
IBM Research
alexander.gray@ibm.com

Debarun Bhattacharjya
IBM Research
debarunb@us.ibm.com

Francisco Barahona
IBM Research
barahon@us.ibm.com

Tian Gao
IBM Research
tgao@us.ibm.com

Ryan Riegel
IBM Research
ryan.riegel@ibm.com

Pravinda Sahu
IBM Consulting
pravisah@in.ibm.com

A Bayesian and Credal Networks

A *Bayesian network* (BN) [13] is defined by a tuple $\langle \mathbf{X}, \mathbf{D}, \mathbf{P}, G \rangle$, where $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a set of variables over multi-valued domains $\mathbf{D} = \{D_1, \ldots, D_n\}$, G is a directed acyclic graph (DAG) over $\mathbf{X}$ as nodes, and $\mathbf{P} = \{P_i\}$ where $P_i = P(X_i|pa(X_i))$ are *conditional probability tables* (CPTs) associated with each variable X_i and $pa(X_i)$ are the parents of X_i in G. A belief network represents a joint probability distribution over $\mathbf{X}$, $P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|pa(X_i))$. An evidence set e is an instantiated set of variables.

Given evidence e, computing the probability of evidence is given by:

$$P(e) = \sum_{\mathbf{X}} \prod_{i=1}^{n} P(X_i|pa(X_i)) \tag{A.1}$$

Similarly, the posterior marginal probability of a variable X_j given evidence e is given by:

$$P(X_j|e) = \frac{1}{P(e)} \cdot \sum_{\mathbf{X} \setminus X_j} \prod_{i=1}^{n} P(X_i|pa(X_i)) \tag{A.2}$$

A *credal set* is a set of probability distributions for a variable X and is typically denoted by $\mathbb{K}(X)$ [9]. Credal sets generalize probability intervals and belief functions, thus offering a general framework for representation and reasoning with imprecise probabilities.

We consider closed convex sets of probability distributions. Given a credal set $\mathbb{K}(X)$ and a function $f(X)$, the lower and upper expectations of $f(X)$ denoted by $\underline{E}[f(X)]$ and $\overline{E}[f(X)]$ are defined respectively as:

$$\underline{E}[f(X)] = \min_{P(X) \in \mathbb{K}(X)} E_P[f(X)] \tag{A.3}$$

$$\overline{E}[f(X)] = \max_{P(X) \in \mathbb{K}(X)} E_P[f(X)] \tag{A.4}$$

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

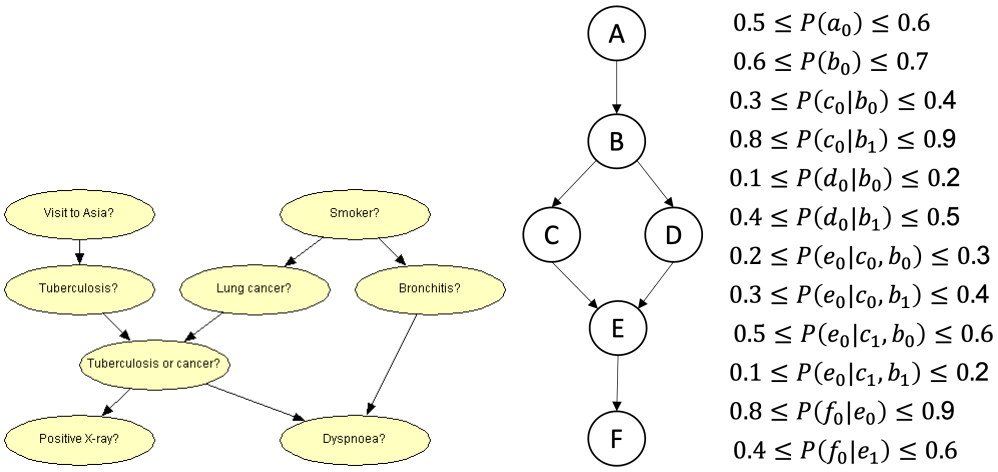

Figure A.1: A Bayesian network (left) and a credal network (right) with binary variables.

where $E_P[f(X)]$ indicates standard expectation. Similarly, the lower probability and upper probability of an event A are defined respectively as:

$$\underline{P}(A) = \min_{P(X) \in \mathbb{K}(X)} P(A)] \tag{A.5}$$

$$\overline{P}(A) = \max_{P(X) \in \mathbb{K}(X)} P(A)] \tag{A.6}$$

A set of probability distributions, its convex hull and its vertices produce the same lower and upper expectations as well as the same lower and upper probabilities.

Conditioning is equated to element-wise application of Bayes rule in a credal set; the posterior credal set is the union of all posterior probability distributions. A conditional credal set $\mathbb{K}(X|Y = y)$ contains conditional probability distributions $P(X|Y = y)$ for random variables X and Y.

Credal sets are typically specified by listing constraints on probability values. If a credal set is closed and convex, then it can be described by listing its extreme points (and taking the convex hull). For example, given a binary variable X, a closed convex credal set $\mathbb{K}(X)$ is fully captured by a single closed interval $[\underline{P}(X = 1), \overline{P}(X = 1)]$.

A *credal network* (CN) [4] is defined by a pair $(G, \mathbb{K})$, where G is a DAG over discrete variables $\mathbf{X}$ and $\mathbb{K}$ is a set of conditional credal sets $\mathbb{K}(X_i|pa(X_i))$ each one associated with a variable $X_i \in \mathbf{X}$ and its parents $pa(X_i)$ in G. We consider *separately specified* credal networks where each variable X_i and each configuration π_{ik} of its parents $pa(X_i)$ in G is associated with a closed convex conditional credal set $\mathbb{K}(X_i|pa(X_i) = \pi_{ik})$ which is specified separately from all others, namely there are no constraints among distributions in these sets.

Example 1. *Figure A.1 (left) shows a Bayesian network with 8 binary variables while Figure A.1 (right) shows a simple credal network with 6 binary variables $\{A, B, C, D, E, F\}$. In this case, each of the conditional credal sets associated with the variables is given by a closed interval. Therefore, we have that $P(a_0) \in [0.5, 0.6]$ and $P(a_1) \in [0.4, 0.5]$, respectively, where a_0 and a_1 are the values of variable A in the credal network.*

B Exact Inference in LCNs

Consider the following LCN:

$$0.6 \leq P(a \wedge b) \leq 1 \tag{B.1}$$

$$0 \leq P(a \mid c) \leq 0.2 \tag{B.2}$$

$$0 \leq P(a \mid \neg c) \leq 0.8 \tag{B.3}$$

$$0 \leq P(b \mid d) \leq 0.7 \tag{B.4}$$

$$0 \leq P(b \mid \neg d) \leq 0.3 \tag{B.5}$$

where $\tau = True$ for the sentences. The implied constraints by the Markov condition are: c and d are independent; a is conditionally independent of d given b and c; b is conditionally independent of c given a and d.

Define sixteen variables $p_{0,0,0,0}, p_{0,0,0,1}, \cdots, p_{1,1,1,1}$ to represent the probabilities of the sixteen interpretations, where the four bits in subscript represent the truth values of a, b, c and d respectively. For example, $p_{0,0,0,0}$ is the probability that a, b, c and d are all false.

Consider a query on the marginal probability of $P(c)$. We formulate two optimization problems:

$$p_{i,j,k,l} \geq 0 \quad , \quad \forall i,j,k,l \in \{0,1\} \tag{B.6}$$

$$\sum_{i,j,k,l \in \{0,1\}} p_{i,j,k,l} = 1 \tag{B.7}$$

$$\sum_{k,l \in \{0,1\}} p_{1,1,k,l} \geq 0.6 \tag{B.8}$$

$$\sum_{j,l \in \{0,1\}} p_{1,j,1,l} \leq 0.2 \cdot \sum_{i,j,l \in \{0,1\}} p_{i,j,1,l} \tag{B.9}$$

$$\sum_{j,l \in \{0,1\}} p_{1,j,0,l} \leq 0.8 \cdot \sum_{i,j,l \in \{0,1\}} p_{i,j,0,l} \tag{B.10}$$

$$\sum_{i,k \in \{0,1\}} p_{i,1,k,1} \leq 0.7 \cdot \sum_{i,j,k \in \{0,1\}} p_{i,j,k,1} \tag{B.11}$$

$$\sum_{i,k \in \{0,1\}} p_{i,1,k,0} \leq 0.3 \cdot \sum_{i,j,k \in \{0,1\}} p_{i,j,k,0} \tag{B.12}$$

$$\sum_{i,j,l \in \{0,1\}} p_{i,j,1,l} \cdot \sum_{i,j,k \in \{0,1\}} p_{i,j,k,1} = \sum_{i,j \in \{0,1\}} p_{i,j,1,1} \tag{B.13}$$

$$\sum_{l \in \{0,1\}} p_{1,0,0,l} \cdot \sum_{i \in \{0,1\}} p_{i,0,0,1} = p_{1,0,0,1} \cdot \sum_{i,l \in \{0,1\}} p_{i,0,0,l} \tag{B.14}$$

$$\sum_{l \in \{0,1\}} p_{1,0,1,l} \cdot \sum_{i \in \{0,1\}} p_{i,0,1,1} = p_{1,0,1,1} \cdot \sum_{i,l \in \{0,1\}} p_{i,0,1,l} \tag{B.15}$$

$$\sum_{l \in \{0,1\}} p_{1,1,0,l} \cdot \sum_{i \in \{0,1\}} p_{i,1,0,1} = p_{1,1,0,1} \cdot \sum_{i,l \in \{0,1\}} p_{i,1,0,l} \tag{B.16}$$

$$\sum_{l \in \{0,1\}} p_{1,1,1,l} \cdot \sum_{i \in \{0,1\}} p_{i,1,1,1} = p_{1,1,1,1} \cdot \sum_{i,l \in \{0,1\}} p_{i,1,1,l} \tag{B.17}$$

$$\sum_{k \in \{0,1\}} p_{0,1,k,0} \cdot \sum_{j \in \{0,1\}} p_{0,j,1,0} = p_{0,1,1,0} \cdot \sum_{j,k \in \{0,1\}} p_{0,j,k,0} \tag{B.18}$$

$$\sum_{k \in \{0,1\}} p_{0,1,k,1} \cdot \sum_{j \in \{0,1\}} p_{0,j,1,1} = p_{0,1,1,1} \cdot \sum_{j,k \in \{0,1\}} p_{0,j,k,1} \tag{B.19}$$

$$\sum_{k \in \{0,1\}} p_{1,1,k,0} \cdot \sum_{j \in \{0,1\}} p_{1,j,1,0} = p_{1,1,1,0} \cdot \sum_{j,k \in \{0,1\}} p_{1,j,k,0} \tag{B.20}$$

$$\sum_{k \in \{0,1\}} p_{1,1,k,1} \cdot \sum_{j \in \{0,1\}} p_{1,j,1,1} = p_{1,1,1,1} \cdot \sum_{j,k \in \{0,1\}} p_{1,j,k,1} \tag{B.21}$$

$$\text{maximize/minimize} \sum_{i,j,l \in \{0,1\}} p_{i,j,1,l} \tag{B.22}$$

Constraints (B.6)(B.7) ensure that the sixteen variables are a valid probability distribution. (B.8–B.12) are explicit and linear constraints from the LCN sentences (B.1–B.5). (B.13–B.21) are implicit and quadratic constraints from the Markov condition, and they have been reduced using techniques from [1]. By maximizing and minimizing the objective function (B.22), we obtain the upper and lower bounds for $P(c)$, which are 0.33 and 0 respectively. For another query on the posterior probability of

$P(a \mid b)$, we replace (B.22) with the following objective:

$$\frac{\sum_{k,l\in\{0,1\}} p_{1,1,k,l}}{\sum_{i,k,l\in\{0,1\}} p_{i,1,k,l}} \tag{B.23}$$

and the resulting interval is $[0.85, 1]$. In some scenarios we may be interested in the model with the maximum entropy [3] and therefore minimize the following objective instead.

$$\sum_{i,j,k,l\in\{0,1\}} p_{i,j,k,l} \cdot \log p_{i,j,k,l} \tag{B.24}$$

Complexity Given an LCN $\mathcal{L}$ with n atomic formulas, the corresponding quadratic program $\mathcal{P}$ has $N = 2^n$ variables corresponding to all of $\mathcal{L}$'s interpretations. Solving $\mathcal{P}$ is NP-hard in general [12], and therefore the complexity of exact inference in LCNs can be bounded by $O(exp(N))$, where N is the number of variables.

C Details of the Mastermind Experiments

C.1 Puzzle Generation

Algorithm C.1 specifies how the puzzles are generated. The reason that we run Knuth's algorithm three times is to obtain a longer board and thereby reduce the number of MAP ties that have the same posterior probabilities. With three, most of the puzzles have a single MAP code to be used as the ground truth. In the case that there are multiple MAP codes, we consider an inference algorithm correct if it guesses any one of them.

Algorithm C.1 Generate a Mastermind puzzle.

1: Sample a hidden code from uniform distribution.
2: Sample each $P(l_i)$ uniformly from [0.3,0.7].
3: Run Knuth's algorithm 3 times until success or contradiction. Each l_i is sampled according to $P(l_i)$ from step 2. If l_i = True, uniformly sample a feedback from possible lies.
4: If the hidden code is guessed in any of the 3 runs, exit.
5: Save the board as a puzzle.
6: Compute the exact MAP as the ground truth.

Figure C.1 illustrates one puzzle generated by Algorithm C.1. Each row has two parts: the first is a guess by Knuth's algorithm, which is composed of 4 colored pegs out of six possible colors, the second part is the feedback which may or may not be a lie.

In addition to puzzles, we also need to generate knowledge as in (25-28) and alike. As shown, the formulas are AND/OR of two consecutive l_i's, and they alternate between AND and OR. Note that we could replace them with arbitrary formulas. For each formula, we first compute the exact point probability p based on the $P(l_i)$ values from step 2 of Algorithm C.1. Then we compute the widest probability interval $[x, y]$ for this formula if $P(l_i)$ could take any value in [0.3,0.7]: for AND, $x = 0.09$ and $y = 0.49$; for OR, $x = 0.51$ and $y = 0.91$. Then we sample a number uniformly from $[x, p]$ as the lower bound for the formula, and sample a number uniformly from $[p, y]$ as the upper bound. This ensures that the knowledge in sentences like (25-28) are correct.

C.2 Details on Puzzle Solving

Figure C.2 depicts the structure of the Bayesian network associated with our Mastermind puzzles with uncertainty. Specifically, variables $\{h_1, h_2, h_3, h_4\}$ correspond to the hidden code and they all have a domain of size 6 corresponding to the available colors. Variables $\{e_1, e_2, \ldots, e_k\}$ correspond to the feedback received from the code-maker and they all have a domain of 14 values corresponding to all possible combinations of black and white feedback pegs. Variables $\{l_1, l_2, \ldots, l_k\}$ are Boolean variables which indicate whether the code-maker lied or not when he provided the feedback in each of the rounds. k is 11 for the example of Figure C.1 and it varies for different puzzles.

The parameters of the network are determined as follows. $P(l_i)$ is sampled uniformly from $[0.3, 0.7]$ for each puzzle. $P(h_i)$ is a uniform distribution such that $P(h_i = c_i) = 1/6$ where $c_i \in \{1, 2, \ldots, 6\}$

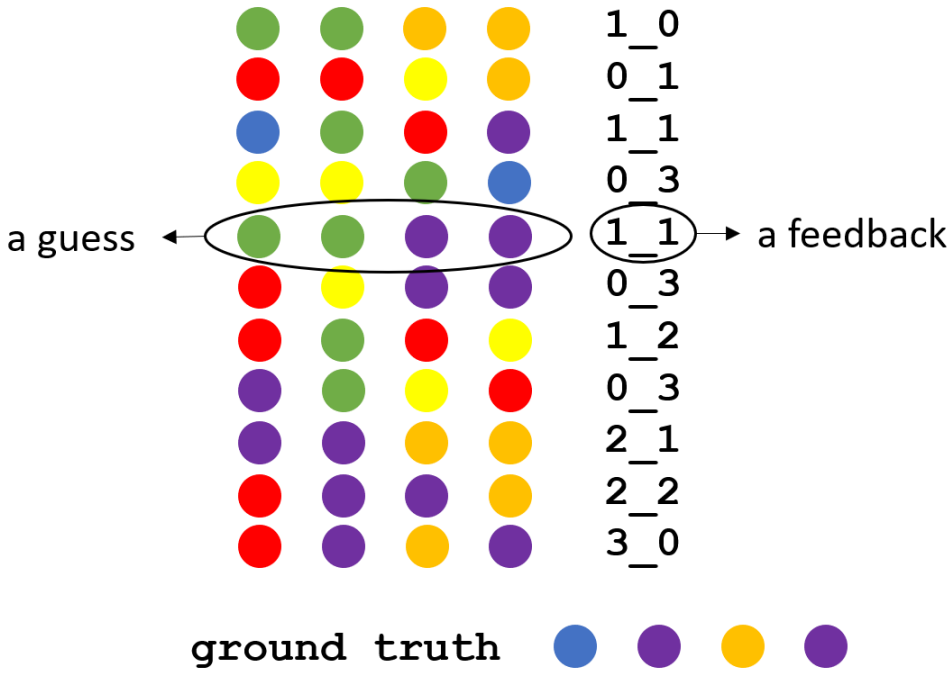

Figure C.1: An example Mastermind puzzle.

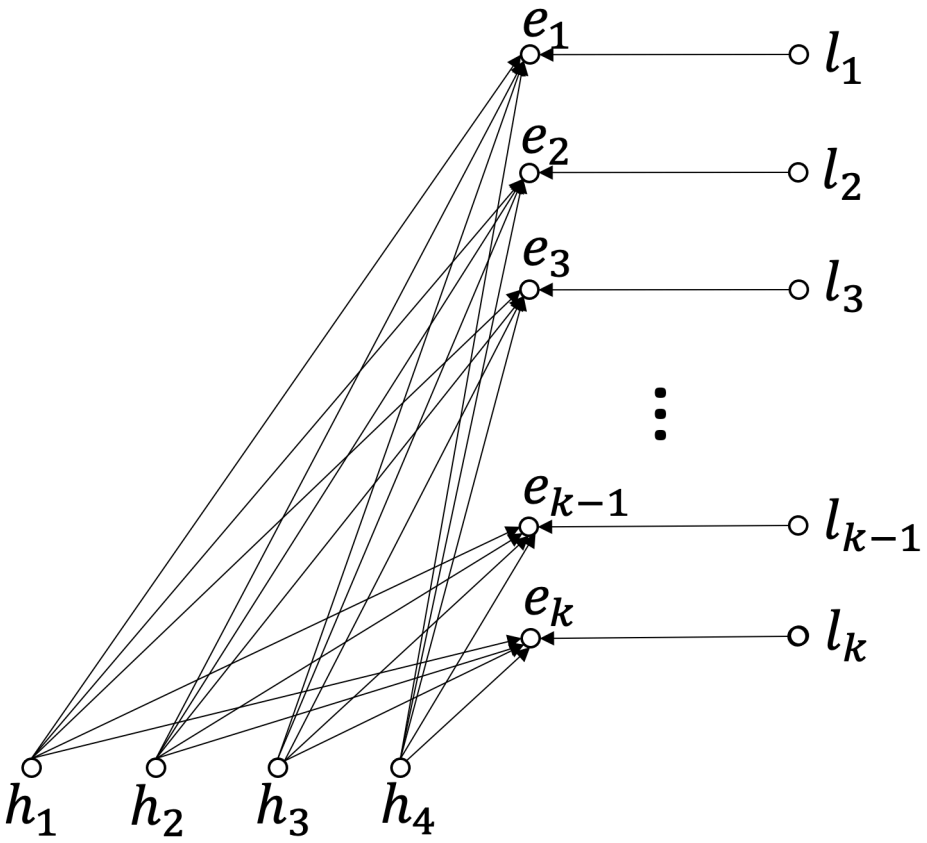

Figure C.2: The structure of the Bayesian network of the Mastermind puzzle.

is one of the 6 colors used. For $P(e_j|h_1, h_2, h_3, h_4, l_j)$ we distinguish 2 cases, depending on the value of l_j.

- If $l_j = false$ (i.e., code-maker doesn't lie) then we have that $P(e_j = f|h_1 = c_1, h_2 = c_2, h_3 = c_3, h_4 = c_4, l_j = false) = 1$ if the feedback value f is the correct feedback for the j^{th} guess on the board and with code (c_1, c_2, c_3, c_4) as the hidden code; $P(e_j = f|h_1 = c_1, h_2 = c_2, h_3 = c_3, h_4 = c_4, l_j = false) = 0$ otherwise. For example, in Figure C.1, the highlighted fifth guess is (green,green,purple,purple), and therefore for the ground truth as (c_1, c_2, c_3, c_4) we would have $P(e_5 =$ "1_1"$|h_1 =$ blue, $h_2 =$ purple, $h_3 =$ brown, $h_4 =$ purple, $l_j = false) = 1$ and it is zero for any feedback other than "1_1".
- If $l_j = true$ (i.e., code-maker lies) then we have that $P(e_j = f|h_1 = c_1, h_2 = c_2, h_3 = c_3, h_4 = c_4, l_j = true) = 1/L$ if the feedback value f is a false yet possible feedback for the j^{th} guess on the board and with code (c_1, c_2, c_3, c_4) as the hidden code, and L is the number of possible lies; $P(e_j = f|h_1 = c_1, h_2 = c_2, h_3 = c_3, h_4 = c_4, l_j = true) = 0$ otherwise. For example, let us consider again the highlighted fifth guess in Figure C.1 and the ground truth as (c_1, c_2, c_3, c_4). Out of all 14 feedbacks, "1_1" is the correct feedback, "1_3" is an impossible feedback for this guess of (green,green,purple,purple). Therefore we have $L = 14 - 2 = 12$ and $P(e_5 = f|h_1 =$ blue, $h_2 =$ purple, $h_3 =$ brown, $h_4 =$ purple, $l_j = true) = 1/12$ for any f other than "1_1" and "1_3".

The inference task of is to compute the hidden code:

$$\text{argmax}_{c_1,c_2,c_3,c_4} P\left(h_1 = c_1, \cdots, h_4 = c_4 \mid e_1, \cdots, e_k\right) \tag{C.1}$$

However, the inference algorithms do not have a complete specification of the Bayesian network of Figure C.2: rather than having the point values of $P(l_i)$, they have (25-28) instead. We also note that we solve (D.1) in a brute-force manner, namely we explicitly enumerate the hidden code configurations, and for each configuration we solve a marginal inference problem $P(q|e_1, ..., e_k)$ where $q \triangleq q_1 \wedge q_2 \wedge q_3 \wedge q_4$ such that q_j is the color assignment to the j^{th} code variable $h_j = c_j$, and $e_1, ..., e_k$ represents the feedback.

LCN(maxent) is an LCN variant that computes a joint distribution $P(l_1, \ldots, l_k)$ of the l_i variables with the greatest entropy and uses that to determine $P(l_i)$ in each round. Specifically, let $l_1, \ldots, l_k$ denote the lie variables and let $p_{l_1..l_k}$ denote the variables representing the probabilities of all interpretations of $l_1, \ldots, l_k$. Note that we have 2^k interpretations (and as many $p_{l_1..l_k}$ variables). We then solve a QP to determine the max entropy $P(l_1, \ldots, l_k)$, where the objective is to minimize $\sum_{l_1..l_k \in \{0,1\}} p_{l_1..l_k} \log p_{l_1..l_k}$ subject to the constraints (25)-(28) as well as the constraints ensuring that the obtained distribution is a proper probability distribution, i.e., $\sum_{l_1..l_k \in \{0,1\}} p_{l_1..l_k} = 1$ and $p_{l_1..l_k} \geq 0 \; \forall l_1..l_k \in \{0, 1\}$. Since we assume that variables l_i are independent of each other (see Fig. D.2), it follows that $P(l_1, \ldots, l_k) = \prod_{i=1}^{k} P(l_i)$ and therefore, we obtain each of the marginal distributions $P(l_i)$ by marginalizing the joint distribution. Finally, *LCN(maxent)* determines the most likely hidden code by solving Equation (D.1) as a standard Marginal MAP task in the Bayesian network from Figure D.2.

Regarding the MLN formulation, we also note that because each puzzle is sampled from a different underlying Bayesian network, it is impossible to train the MLN weights.

D Details of the Credit Card Fraud Detection Experiments

We note that when we create the train and test sets for the 10 tasks, we sample accounts rather than transactions; in other words, transactions from the same account are either all included in a test set or all excluded from it. Moreover, since the training data contains only single-transaction accounts, the above logic rules and the quantification of their certainty can only come from human experts. The task is therefore how to utilize both knowledge from the training data and knowledge from experts to best detect fraudulent transactions in the test set.

The LCN for credit card fraud detection task is the following. Let X denote the binary Is-Fraud variable; let F_i denote the i^{th} feature variable in the Naive Bayes classifier, and let $f_{i,j}$ denotes the j^{th} possible value for F_i; let $c_{0,i,j}$ denote the probability of $F_i = f_{i,j}$ in legitimate transactions in

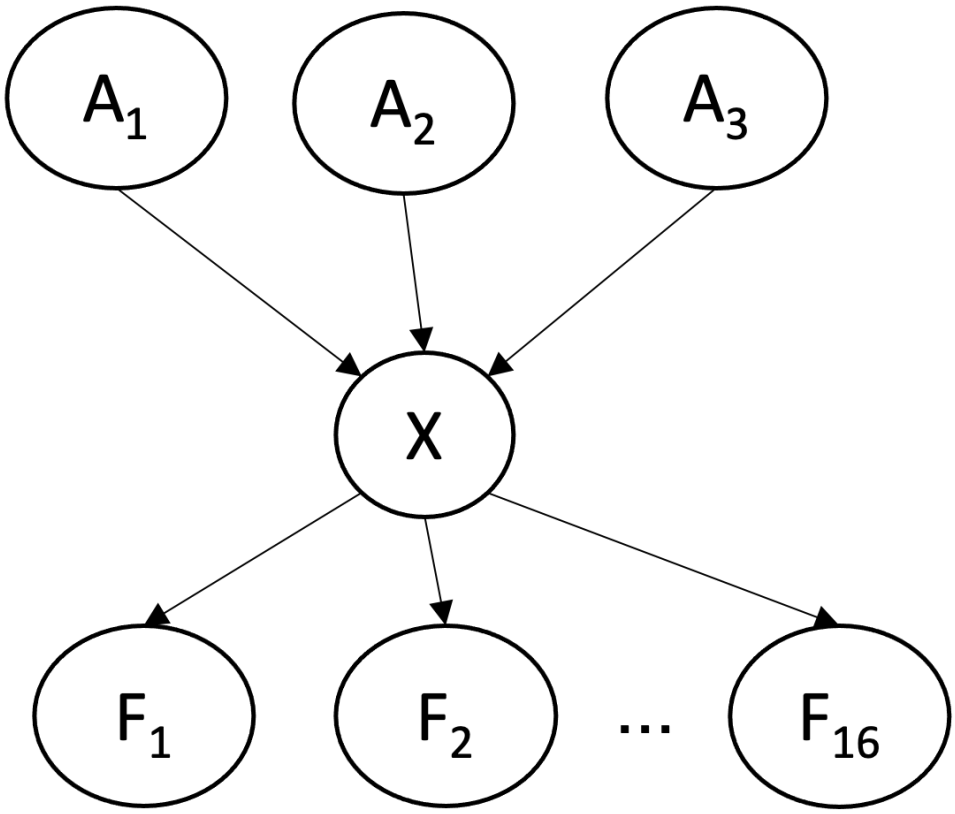

Figure D.1: Bayesian/credal network structure for credit card fraud.

training data, and let $c_{1,i,j}$ denote that for fraudulent ones; let c_X denote the fraction of fraudulent transactions in training data; let A_1, A_2, A_3 denote the antecedents in the three logic rules.

$$c_X \leq P(X) \leq c_X \tag{D.1}$$
$$c_{0,i,j} \leq P(F_i = f_{i,j} \mid \neg X) \leq c_{0,i,j}\,,\ \forall i,j \tag{D.2}$$
$$c_{1,i,j} \leq P(F_i = f_{i,j} \mid X) \leq c_{1,i,j}\,,\ \forall i,j \tag{D.3}$$
$$0.65 \leq P(A'_1) \leq 0.74 \tag{D.4}$$
$$0.31 \leq P(A'_2) \leq 0.66 \tag{D.5}$$
$$0.44 \leq P(A'_3) \leq 0.72 \tag{D.6}$$
$$1 \leq P(X \mid A_1 \wedge A'_1) \leq 1 \tag{D.7}$$
$$1 \leq P(X \mid A_2 \wedge A'_2) \leq 1 \tag{D.8}$$
$$1 \leq P(X \mid A_3 \wedge A'_3) \leq 1 \tag{D.9}$$

The first three equations (D.1)(D.2)(D.3) are exactly the Naive Bayes model. The latter six equations use three auxiliary variables A'_1, A'_2, A'_3, and the effect is similar to the noisy-OR model in Bayesian_midpoint and Credal. However, unlike credal/Bayesian networks, LCN does not require unique-assessment assumption and therefore (D.1) is still allowed. Table 2 shows that this flexibility of LCN results in substantial performance gains.

We used an *MLN* encoding of the NaiveBayes model extended with the three logic rules as follows: if p_{mid} is the midpoint of one of the three probability intervals then we add $\texttt{antecedent} \wedge \texttt{consequent}$ with weight $\log(p_{\text{mid}})$ and $\texttt{antecedent} \wedge \neg\texttt{consequent}$ with weight $\log(1 - p_{\text{mid}})$, respectively. We note that because the training data contains no account history, it is impossible to train the MLN weights.

Let $t = (F_1 = f_1, \cdots, F_{16} = f_{16})$ be a transaction in the test set, where $F_i = f_i$ means that feature F_i takes value f_i. We then compute $P(X|e)$, and $P(\neg X|e)$, where e is evidence, namely $e = \{F_1 =$

$f_1, \cdots, F_{16} = f_{16}, A_1, A_2, A_3\}$. We label the transaction as fraud if $P(X|e) > P(\neg X|e)$. Note that in this case the maximin and maximax criteria give the same result because the posterior interval for $P(X|e)$ determines the one for $P(\neg X|e)$ (i.e., let $[l, u]$ be the posterior interval for $P(X|e)$, then $[1-u, 1-l]$ is the posterior interval for $P(\neg X|e)$). Once we label all transaction in the test set, we compare with the actual labels and subsequently determine the F1 score for the test set. Recall that the F1 score is:

$$F_1 = \frac{tp}{tp + \frac{1}{2}(fp + fn)} \tag{D.10}$$

where tp stands for true positives, fp stands for false positive, and fn stands for false negatives.

Figure D.1 depicts the structure of Bayesian/credal network we used in this experiment. The prior probabilities and intervals for $P(A_i)$ were set to 0.5 and $[0, 1]$, respectively. Furthermore, $P(F_i|X)$ are derived from equations (D.2) and (D.3), while $P(X|A_1, A_2, A_3)$ is modeled as a standard Noisy-OR conditional probability distribution. For example $P(X|A_1, A_2, A_3) = 1 - (1 - a_1) \cdot (1 - a_2) \cdot (1 - a_3)$ where $P(X|A_1) = a_1$, $P(X|A_2) = a_2$ and $P(X|A_3) = a_3$ are estimated from the training data.

E Additional Related Work

Probabilistic logic combines probabilities with classical logic, thus providing a unified framework for representation and reasoning with complex concepts and relations under uncertainty. Several probabilistic logic formalisms have been proposed over the past decades.

Nilsson's probabilistic logic [10, 11] is perhaps the first system that provides a semantical generalization of first-order logic in which the truth values of logical sentences (or formulas) can range between 0 and 1 and are interpreted as the probability of those sentences being true. The formalism allows computing lower and upper bounds on the probability of a query formula.

Bayesian logic [1] combines probabilistic logic and Bayesian networks in order to capture conditional independence relations among propositions. In addition, there is no need to give a complete specification or prior and conditional probabilities as it is required for Bayesian networks. The model allows specifying bounds on conditional probabilities. Nonlinear programming is used to compute a range of probabilities (interval) for a query formula.

Markov Logic Networks (MLN) [15] is a probabilistic logic which applies the ideas of a Markov network to first-order logic, enabling uncertain inference. MLNs generalize first-order logic, in the sense that, in a certain limit, all unsatisfiable statements have a probability of zero, and all tautologies have probability one. Briefly, an MLN is a collection of formulas from first-order logic, to each of which is assigned a real number, the weight. The weights are used to define a joint probability distribution over all possible interpretations of the logic formulas.

Probabilistic Soft Logic (PSL) [7] is a probabilistic logic that combines graphical models (Markov networks) with *soft* or real-valued logic (e.g., Lukasiewicz logic). Inference in PSL is formalized as a convex optimization problem with a hinge-loss objective function.

Probabilistic Logic Programs (PLP) [14] are logic programs in which some of the facts are annotated with probabilities. In general, a logic program is a set of rules where each rule is a universally quantified expression of the form $h : -b_1, b_2, ..., b_n$, where h is an atom and $b_1, ..., b_n$ are literals. ProbLog is a well known implementation of PLP.

Probabilistic Logic Networks (PLN) [8] is a kind of probabilistic logic that associates probabilities to the truth values of the logic formulas. The truth values are represented by intervals and are associated with a *credibility* value which is also a real number between 0 and 1. However, the PLN semantics are different from LCN's.

Stochastic Logic Programs (SLP) [5] is a probabilistic extension of a normal logic program that has been proposed as a flexible way of representing complex probabilistic knowledge; generalising, for example, Hidden Markov Models, Stochastic Context-Free Grammars and Markov networks. Specifically, an SLP is a logic program where some of the predicates have non-negative numbers attached to the clauses which make up their definitions.

Probabilistic Databases (PDB) [2] are relational databases where each tuple of a relation is associated with a real value between 0 and 1 indicating the probability that the tuple belongs to the database. The central problem in PDBs is query evaluation which can be viewed as a counting task on a graphical model that represents the probabilistic database. Temporal Probabilistic Databases (TP) [6] extend PDBs with temporal information. Specifically, each tuple in a relation is associated with a probability value as well as a temporal interval $[a, b]$ where a and b are units of time and $a \leq b$ (or sometimes just timestamps). Query evaluation in TPs handles the temporal intervals using a special temporal algebra and separately from probabilities.

We note that MLN, PSL, PLP, SLP, PDB and TP do not allow specifying probability bounds (i.e., lower and upper bounds). More importantly, our proposed LCN is the only method that can solve effectively the uncertain Mastermind puzzles as well as the credit card fraud detection with uncertain expert knowledge applications.

References

[1] K. Andersen and J. Hooker. Bayesian logic. *Decision Support Systems*, 11(2):191–210, 1994.

[2] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *Proceedings of 13th International Conference on Very Large Data Bases (VLDB)*, pages 71–81, 1987.

[3] Peter Cheeseman. A method of computing generalized bayesian probability values for expert systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 198–202, 1983.

[4] F. Cozman. Credal networks. *Artificial Intelligence*, 120(2):199–233, 2000.

[5] J. Cussens. Stochastic logic programs: Sampling, inference and applications. In *Uncertainty in Artificial Intelligence (UAI)*, pages 115–122, 2000.

[6] Maximilian Dylla, Iris Miliaraki, and Martin Theobald. A temporal-probabilistic database model for information extraction. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 1810–1821, 2013.

[7] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. MIT Press, 2007.

[8] B. Goertzel, M. Ikle, I. Lyon Freire Goertzel, and A. Heljakka. *Probabilistic Logic Networks: A Comprehensive Conceptual, Mathematical and Computational Framework for Uncertain Inference*. Springer, 2008.

[9] I. Levi. *The Enterprise of Knowledge*. MIT Press, 1980.

[10] N. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.

[11] N. Nilsson. Probabilistic logic revisited. *Artificial Intelligence*, 59(1-2):39–42, 1994.

[12] P. Pardalos and S. Vavasis. Quadratic programming with one negative eigenvalue is (strongly) np-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

[13] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[14] L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton. *Probabilistic Inductive Logic Programming - Theory and Applications*. Springer, 2008.

[15] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.