INTERVAL-VALUED PROBABILISTIC LOGIC FOR LOGIC PROGRAMS

Phan Dinh Dieu and Phan Hong Giang

Institute of Information Technology,

1. Introduction

This paper presents an approximate method for the probabilistic entailment problem in knowledge bases where a portion of knowledge is given by a sentence in propositional logic accompanied with an interval representing its truth probability.

Let $\mathcal{B}$ be a knowledge base consisting of sentences $S_1, S_2, \ldots, S_L$ given together with their inteval-valued truth probabilities $I_1, I_2, \ldots, I_L$: $\mathcal{B} = \{< S_i,\ I_i > | i = 1, \ldots, L\}$. Let S be any given sentence. By the semantics introduced by Nilsson in [11], the problem of entailling interval I for S from $\mathcal{B}$ is reduced to solving two linear programming problems which are usually of size $\sim 2L \times 2^L$ (about $2L$ linear constraints and 2^L variables). The method presented in this paper proposes a new way to solve the problem. It reduces the entailment problem to the one of finding the set of *"prime implicants"* of S expressed through variables $S_1, S_2, \ldots, S_L$. It allows us to avoid linear programming problems of very large sizes and therefore makes the probabilistic entailment problem manageable. However, instead of exact solution, it gives only approximate one.

This approximate method is shown to be especially efficient for probabilistic logic programs when logical skeletons of knowledge bases form usual logic programs. In this case the set of prime implicants can be found by *SLD*- resolution applied to a deterministic extension of logic programs.

The proposed method is presented in section 2 for general case and in section 3 for the case of probabilistic logic programs. The last section containts discussion and some concluding remarks.

2. Entailment problem and approximate solution

2.1. Entailment problem in probabilistic logic

Firstly, we recall some basic notions of the theory of interval-valued probabilistic logic developped on the basis of the semantics given by N. J. Nilsson in [11]. A presentation in details of this theory can be found in [12,14].

A knowledge base $\mathcal{B}$ is assumed to be given in the form:

$$\mathcal{B} = \{< S_i, I_i > | i = 1, \ldots, L\}$$

where S_i is a sentence in propositional logic, and $I_i = [\alpha_i, \beta_i] \subseteq [0,1]$ is the interval-value for its truth probability. Let S be a target sentence. Its truth probability must be also an interval denoted by $I(S) = [\alpha(S), \beta(S)]$. The entailment problem is the problem of calculate the lower and upper bounds of $I(S)$.

We put $\Gamma = \{S_1, S_2, \ldots, S_L\}$ and $\Sigma = \{S_1, S_2, \ldots, S_L, S\}$. Γ is called *logical skeleton* of $\mathcal{B}$.

Let $U = \{u_1, \ldots, u_k\}$ where $u_j = (u_{1j}, \ldots, u_{Lj})^T$ for $j = 1, \ldots, k$ and $V = \{v_1, \ldots, v_h\}$ where $v_m = (v_{1m}, \ldots, v_{Lm}, v_{sm})^T$ for $m = 1, \ldots, h$ be the sets of consistent vectors of boolean values of sentences in Γ and Σ respectively.

Each vector of V characterizes a Σ-*class* of possible worlds. Given a probability distribution $q = (q_1, \ldots, q_h)$ over the set of Σ-classes, the truth probability $\pi(S_i)$ of the sentence S_i is defined to be the sum of probabilities of classes of worlds in which S_i is *true*, i.e., $\pi(S_i) = v_{i1}.q_1 + \ldots + v_{ih}.q_h$. From this semantics it follows that lower bound $\alpha(S_i)$ and upper bound $\beta(S_i)$ of interval $I(S_i)$ are the solutions of the following linear programming problems:

$$\alpha(S) = \min (v_{s1}.q_1 + \ldots + v_{sh}.q_h) \tag{1}$$

$$\beta(S) = \max (v_{s1}.q_1 + \ldots + v_{sh}.q_h) \tag{2}$$

subject to constraints:

$$\begin{cases} v_{i1}q_1 + \ldots + v_{ih}q_h \in I_i & \text{for } i = 1, \ldots, L \\ \sum_{m=1}^{h} q_m = 1 & \\ q_m \geq 0 & \text{for } m = 1, \ldots, h \end{cases} \tag{3}$$

We usually denote $[\alpha(S), \beta(S)]$ by $F(S, \mathcal{B})$ and write $\mathcal{B} \vdash < S, F(S, \mathcal{B}) >$.

Example. For a knowledge base given as follows:

$\mathcal{B} = \{S_1 : A \vee \neg B$ [.9 1.]
$S_2 : B \vee \neg D$ [.8 .9]
$S_3 : C \vee \neg A$ [.6 .8]
$S_4 : D$ [.8 1.]
$S_5 : C$ [.2 .4] $\}$

$S: A$

Its logical skeleton and basic matrices will be:

$\Gamma = \{S_1 : A \vee \neg B$
$S_2 : B \vee \neg D$
$S_3 : C \vee \neg A$
$S_4 : D$
$S_5 : C \ \}$

$$U = \begin{array}{c} \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 & u_9 & u_{10} & u_{11} & u_{12} \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

and

$$V = \begin{array}{c} \\ S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \\ S \end{array} \begin{array}{c} \begin{array}{cccccccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & v_{13} & v_{14} \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

2.2. Reduction of linear programming problems

Let $u = (u_1, \ldots, u_k)$ be a k-dimentional boolean vector and $\sigma \in \{0,1\}$. We denote $u\sigma = (u_1, \ldots, u_k, \sigma)$ and call it an $(k+1)$-dimentional extension of u.

We note that each Γ-consistent vector $u \in U$ has at least one extension $u\sigma \in V$ and conversely each Σ-consistent vector $v \in V$ is an extension of some vector in U. We can partition the set U into the following subsets:

$$U = T(S) \cup F(S) \cup N(S) \quad \text{where}$$

$$T(S) = \{u_j \in U | u_j 1 \in V \text{ and } u_j 0 \notin V\}$$
$$F(S) = \{u_j \in U | u_j 0 \in V \text{ and } u_j 1 \notin V\}$$
$$N(S) = \{u_j \in U | u_j 1 \in V \text{ and } u_j 0 \in V\} = (U \setminus T(S)) \setminus F(S) .$$

Example (continued).

$T(A) = \{u_1 = (1,1,1,1,1)^T,\ u_3 = (1,1,0,1,0)^T,\ u_4 = (1,1,0,0,0)^T,\ u_6 = (1,0,0,1,0)^T\}$
$F(A) = \{u_7 = (0,1,1,1,1)^T,\ u_8 = (0,1,1,0,1)^T,\ u_9 = (0,1,1,1,0)^T,\ u_{10} = (0,1,1,0,0)^T,$
$u_{11} = (1,0,1,1,0)^T,\ u_{12} = (1,1,1,0,0)^T\}$
$N(A) = \{u_2 = (1,1,1,0,1)^T,\ u_5 = (1,0,1,1,1)^T\}$

Now let consider two following linear programming problems:

$$\alpha(S) = min \sum_{u_j \in T(S)} p_j \tag{4}$$

$$\beta(S) = 1 - min \sum_{u_j \in F(S)} p_j \tag{5}$$

subject to constraints:

$$\begin{cases} u_{i1}.p_1 + \ldots + u_{ik}.p_k \in I_i & \text{for } i = 1,\ldots,L \\ \sum_{j=1}^{k} p_j = 1 & \\ p_j \geq 0 & \text{for } j = 1,\ldots,k \end{cases} \tag{6}$$

The major difference between two pairs of problems (1),(2),(3) and (4),(5),(6) lays in the fact that in the former, minimization (maximization) is done for the sum of all variables with corresponding values in the last row is 1, but in the later it is done for the sum of variables which belong to group $T(S)$. The relationship between them is made clear in the following theorem.

Theorem 1. *Problems* (1)–(3) ((2)–(3)) *and* (4)–(6) ((5)–(6)) *have the same solution.*

Proof. Let α_1 and α_2 be the values of $\alpha(S)$ given by solving problems (1) – (3) and (4) – (6) respectively. We shall prove $\alpha_1 = \alpha_2$.

Suppose that α_1 is reached at vector $(q_1,\ldots,q_h)$. We define a vector $p = (p_1,\ldots,p_k)$ as follows:

$$p_j = \begin{cases} q_m, & \text{if } v_m = u_j\sigma \text{ for some } \sigma \in \{0,1\} \text{ and } u_j \in T(S) \cup F(S) \\ q_m + q_n & \text{if } u_j \in N(S) \text{ and } v_m = u_j 1,\ v_n = u_j 0 \end{cases}$$

It is easy to see that vector p satisfies conditions (6), and moreover,

$$\alpha_1 = \sum_{m=1}^{h} v_{sm} q_m \geq \sum_{u_j \in T(S)} p_j,$$

Therefore $\alpha_1 \geq \alpha_2$.

Conversely, suppose that value α_2 is reached at vector $(p_1,\ldots,p_k)$. Now we define a vector $q = (q_1,\ldots,q_m)$ as follows:

$$q_m = \begin{cases} p_j & \text{if } v_m = u_j\sigma \text{ for some } \sigma \in \{0,1\} \text{ and } u_j \in T(S) \cup F(S) \\ 0 & \text{if } v_m = u_j 1, \text{ and } u_j \in N(S) \\ p_j & \text{if } v_m = u_j 0 \text{ and } u_j \in N(S) \end{cases}$$

It is obvious that vector q satisfies conditions (3), and moreover,

$$\alpha_2 = \sum_{u_j \in T(S)} p_j \geq \sum_{m=1}^{h} v_{sm} q_m,$$

Therefore $\alpha_2 \geq \alpha_1$. Thus $\alpha_1 = \alpha_2$ is proved. Similarly, we can also prove that the values of $\beta(S)$ given by solving problems (2) – (3) and (5) – (6) are the same.

Example (continued).

Solving problems (1) – (3), (2) – (3), (4) – (6) and (5) - (6) for $S = A$ by the simplex method we have the same pair of answers $\alpha(A) = 0.5, \beta(A) = 0.8$.

Corollary 2. *Let $\mathcal{B}$ be a knowledge base, and S and S' be two sentences. If $T(S) = T(S')$ then $\alpha(S) = \alpha(S')$; and if $F(S) = F(S')$ then $\beta(S) = \beta(S')$.*

Remark. Theorem 1 renders a slight reduction of the cost of finding interval $F(S, \mathcal{B})$ by linear programming method. Indeed, the size of problem (4) – (6) is smaller than that of problem (1) – (3) since $k \leq h$; moreover the formulation of constraints (6) only requires the fixed set U of Γ-consistent vectors of $\mathcal{B}$ which does not depends on the target sentence (in the case of (3), for each new target we need a new basic matrix).

2.3. An approximate solution

Let consider the set $W = \{0, 1, *\}^L$, where $*$ stands for *"undefined"* . Elements of W are called L-dimentional extended boolean vectors. Let $u = (u_1, \ldots, u_L)$ be an extended vector. Denote $\Gamma^u = S_1^{u_1} \wedge \ldots \wedge S_L^{u_L}$, where $S^1 = S$, $S^0 = \neg S$ (we write also $\bar{S}$), and $S^* = true$. For example if $u = (1, *, 0, 1, *)$ then $\Gamma^u = S_1 \bar{S}_3 S_4$.

Let us define a partially ordering relation $\preceq$ on $\{0, 1, *\}$, specified by axioms $\forall x \in \{0, 1, *\}$ $x \preceq x$, $0 \preceq *$ and $1 \preceq *$. This relation can be naturally generalized on the elements of W. For $u, v \in W$, we write $u \preceq v$ iff $u_j \preceq v_j$ for $j = 1, \ldots, L$.

Let $u \in W$. We say that a boolean vector $v \in \{0, 1\}^L$ is an *instance* of u, if v is obtained from u by replacing (independently) each occurence of $*$ by 0 or 1. Denote by $E'(u)$ the set of all instances of u and then put $E(u) = E'(u) \cap U$, (Remember U is the set Γ-consistent vectors, so $E(u)$ is the set of all Γ-consistent instances of u).

Let M be a subset of W. We denote

$$E(M) = \bigcup_{u \in M} E(u)$$

and notice that

$$E(M) = T\left(\bigvee_{u \in M} \Gamma^u\right) \tag{7}$$

A vector $u \in W$ is called a Γ-*supporting vector* for S if $E(u) \subseteq T(S)$. Further, if there does not exist any other Γ- supporting vector u' such that $u \preceq u'$ then we call u a *maximal* Γ-*supporting vector* for S.

It is evident that if $u \in T(S)$ then $\Gamma^u \models S$, and if $u \in F(S)$ then $\Gamma^u \models \neg S$. If $u \in W$ is Γ-supporting vector for S, then $\Gamma^u \not\models S$; in this case we say that Γ^u is an *implicant* of S. If u is maximal Γ-supporting vector for S then Γ^u is called *prime implicant* of S (expressed through variables $S_1, \ldots, S_L$).

A set $M \subseteq W$ is called a *complete set* of Γ-supporting vectors for S if $E(M) = T(S)$. From (7), it follows that if M is complete set of Γ-supporting vectors for S then

$$\alpha(S) = \alpha\left(\bigvee_{u \in M} \Gamma^u\right) \tag{8}$$

We say that M is a *maximal complete set* if it is complete and each element of M is a maximal Γ-supporting vector for S. Note that the set of all maximal Γ-supporting vectors is maximal complete.

We say that two vectors $u, v \in W$ are *incompatible* iff $E(u) \cap E(v) = \emptyset$. In other words, u and v are incompatible iff there is an index $1 \leq i \leq L$ such that either $u_i = 0$ and $v_i = 1$ or $u_i = 1$ and $v_i = 0$.

Now let M be the set of all maximal Γ-supporting vectors for S. A subset of M is called *reduced* iff it contains pairwise incompatible vectors. Let $\{M_1, M_2, \ldots, M_r\}$ be the set of reduced subsets of M. From (8) we have

$$\alpha(S) = \alpha\left(\bigvee_{1 \leq i \leq r} \left(\bigvee_{u \in M_i} \Gamma^u\right)\right) \tag{9}$$

Recall that in the theory of probabilistic logic we have:

(a) $\alpha(S_1 \vee S_2) \geq max(\alpha(S_1), \alpha(S_2))$,

(b) if $S_1 \wedge S_2$ is *false* then $\alpha(S_1 \vee S_2) = \alpha(S_1) + \alpha(S_2)$,

(c) $\alpha(S_1 \wedge S_2) \geq max(0,\ \alpha(S_1) + \alpha(S_2) - 1)$.

From these facts and (9) we have:

$$\alpha(S) \geq \max_{1 \leq i \leq r} \left(\sum_{u \in M_i} \alpha(\Gamma^u)\right), \tag{10}$$

where

$$\alpha(\Gamma^u) \geq max\,(0,\ \alpha(S_1^{u_1}) + \cdots + \alpha(S_L^{u_L}) - L + 1) \tag{11}$$

The expression in the right side of (10) with $\alpha(\Gamma^u)$ evaluated by the right side of (11) can be taken as an approximation for $\alpha(S)$.

The upper bound of the interval for S can be calculated as the lower bound for $\neg S$, since $\beta(S) = 1 - \alpha(\neg S)$.

In summary, the approximate algorithm for calculation of interval $[\alpha(S), \beta(S)]$ consists of four steps:

1. Find the set M of all maximal Γ-supporting vectors for S;
2. Find all maximal reduced subsets of M, assume they are $M_1, \ldots M_r$;
3. Calculate the approximate value for $\alpha(S)$ by expressions given in the right sides of (10) and (11);
4. Similarly calculate the approximate value for $\alpha(\neg S)$, and take $\beta(S) = 1 - \alpha(\neg S)$.

Example (continued).

The maximal Γ-supporting vectors for S are $(1, 1, *. 1, *)$ and $(*, *, 0; *, *)$. They are compatible, therefore

$$\alpha(S) \geq max\left(\alpha(S_1 S_3 S_4),\ \alpha(\bar{S}_3)\right) = max(0\ 5,\ 0.2) = 0.5$$

The maximal Γ-supporting vectors for $\neg S$ are $(0, *, *, *, *)$ and $(*, *, 1, *, 0)$. They are compatible, therefore

$$\alpha(\neg S) \geq max\left(\alpha(\bar{S}_1),\ \alpha(S_3 \bar{S}_5)\right) = max(0,\ 0.2) = 0.2$$

It means $\beta(S) = 1 - 0.2 = 0.8$

We note that the interval [0.5, 0.8] is exacly what obtained by solving linear programming problems.

3. Probabilistic logic programs

As mentioned above, the proposed appoximation for the entailment problem in probabilistic logic requires the set of all maximal Γ-supporting vectors for target sentence S. There is a straightforward way to compute it from the basic matrix for Γ. But this way becomes impractical once Γ has dozens of sentences, because the formulation of the basic matrix alone, in general case, has the exponential complexity. Naturally, some question rise: is there any way to compute the set of maximal Γ-supporting vectors without having the basic matrix for Γ? At least, for what restricted class of knowledge bases the question can be answered positively?

In this section, we consider knowledge base $\mathcal{B}$, logical skeleton Γ of which is a set of disjunctions[1] of literals. By interpreting expression $< A(x), [\alpha(x), \beta(x)] >$ as a set $\{< A(c), [\alpha(c), \beta(c)] > | c$ *is constant symbol* $\}$, the database logic programs and knowledge bases of a large number of expert systems using first order language

[1] A knowledge portion $< S, [\alpha(S), \beta(S)] >$, where S is a conjunction of literals $S = B_1 \wedge \ldots \wedge B_m$ could be madfit in this framework by using its negative form $< \neg S, [1 - \beta(S), 1 - \alpha(S)] >$, where $\neg S = \neg B_1 \vee \ldots \vee \neg B_m$.

having no function symbol and a finite number of constant symbols could be brought back to propositional language. Therefore, they fall into the considered here class.

The procedure of computing the set of maximal Γ–supporting vectors for an atom A comprises two stages. Firstly, Γ is unfolded into Γ_e - a set of definite program clauses. Then, on the obtained Γ_e, a modified version of SLD–resolution is applied. We will prove that the proposed algorithm is correct. We use the terms *definite program clause, definite program, definite goal, resolvent, SLD-derivation, SLD-refutation, and SLD-tree* as defined in [10].

3.1. Unfolding the declared program

For each declared clause S_i of Γ - $S_i : A_1 \vee A_2 \vee \ldots \vee A_k$ we create $2k$ new clauses:

$A_l \leftarrow A_1^-, \ldots, A_{l-1}^-, A_{l+1}^-, \ldots, A_k^-, S_i$ for $l = 1, \ldots, k$

$A_l^- \leftarrow S_i^-$ for $l = 1, \ldots, k$,

where S_i are special symbols, which are reserved for naming the clauses. A_l^- and S_i^- are newly introduced symbols. The set Γ_e of newly created clauses is called *unfolded program* from Γ.

Example (continued). Unfolded program Γ_e will be as follows:

$\Gamma_e = \{$ $A \leftarrow S_1, B$ (1)
$B^- \leftarrow S_1, A^-$ (2)
$B \leftarrow S_1^-$ (3)
$A^- \leftarrow S_1^-$ (4)
$B \leftarrow S_2, D$ (5)
$D^- \leftarrow S_2, B^-$ (6)
$D \leftarrow S_2^-$ (7)
$B^- \leftarrow S_2^-$ (8)
$C \leftarrow S_3, A$ (9)
$A^- \leftarrow S_3, C^-$ (10)
$A \leftarrow S_3^-$ (11)
$C^- \leftarrow S_3^-$ (12)
$D \leftarrow S_4$ (13)
$D^- \leftarrow S_4^-$ (14)
$C \leftarrow S_5$ (15)
$C^- \leftarrow S_5^-\}$. (16)

A clause of k literals in Γ will correspond to $2k$ clauses in Γ_e. For a declared program of L clauses, and l literals each, the unfolded program will consist of $2L * l$ clauses. In practice, the numbers of literals in a clause of knowledge bases are often limited to small enough constant. If the number of literals in declared clauses is bounded then the size of unfolded programs is linear proportional to one of declared programs.

3.2. Algorithm for computing a set of Γ-supporting vectors

Formally, Γ_e is a definite logic program, we can apply *SLD*-resolution to prove a clause. We will modify it as follows to find the set of Γ-supporting vectors.

+ Each resolvent will be checked against contradiction, i.e. the simultaneous presence of a complementary pair of either normal literals A and A^-, or special literals S and S^-. If found, stop.
+ Skip subgoal which is a special literal S_i or S_i^-.

With such modifications, the final resolvent for a finite[2] derivation will fall into one of three categories:

+ Success: when all remained subgoals are special literals.
+ Failure of type 1: when the selected sub-goal can not match with any head of program clause of Γ_e.
+ Failure of type 2: when the contradiction found in the resolvent.

We call the modified *SLD*-resolution *PSLD-resolution.*

Let target sentence S be the single atom A. After applying *PSLD*-resolution for $\Gamma_e \cup \{\leftarrow A\}$, we construct for each success resolvent R a *success vector* $v = (v_1, \ldots, v_l)^T$ where:

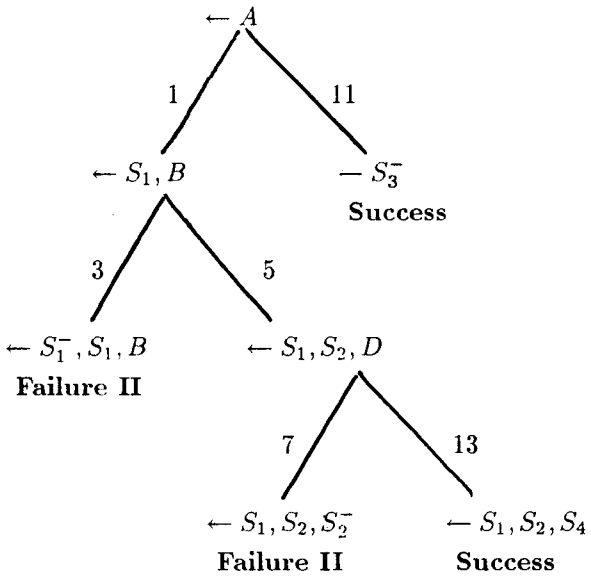
$$v_i = \begin{cases} 1, & \text{if } S_i \in R \\ 0, & \text{if } S_i^- \in R \\ *, & \text{otherwise.} \end{cases}$$

Let V be the set of success vectors.

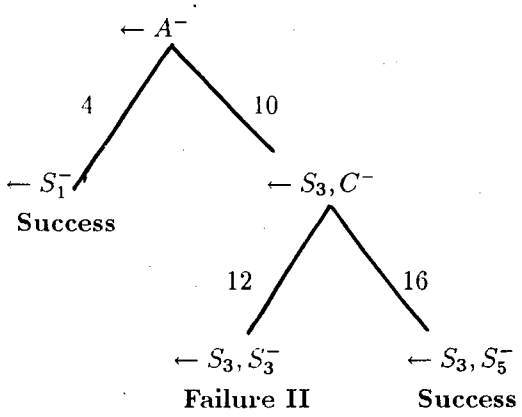
Example (continued). *Solution*

PSLD-tree for $\Gamma_e \cup \{\leftarrow A\}$ has 2 success vectors: $V = \{(1,1,*,1,*), (*,*,0,*,*)\}$. Note that $E(V) = T(A)$, and moreover the elements of V are maximal Γ-supporting for A

[2] For the considered class of logic programs, there are some techniques that allow avoiding infinite derivations. See [8], [15] for example.



PSLD-tree for $\Gamma_e \cup \{\leftarrow A^-\}$ has 2 success vectors: $V' = \{(0,*,*,*,*),\ (*,*,1,*,0)\}$. Note that $E(V') = F(A)$, and the elements of V' are maximal Γ-supporting for $\neg A$.



Note that by the resolution algorithm, we come to the solutions presented in the last example of section 2. Now, we will validate the proposed approximate method by proving the correctness of the *PSLD*-resolution.

Definition. Let v be a truth vector, a *simplification according to* v of the unfolded program $\Gamma_e(v)$ is the set of clauses which contains S_i provided $v(i) = 1$, or S_i^- provided $v(i) = 0$. Then in the obtained program, all special literals are deleted.

Example (continued). Let $v = (1,1,*,1,*)$

$$\Gamma_e(v) = \{\ A \ - B$$
$$B^- - A^-$$
$$B \ - D$$
$$D^- - B^-$$
$$D \ - \}$$

With identification of A^- and $\neg A$, it is clear that $\Gamma_e(v)$ and Γ^v are logically equivalent.

Theorem 3. Soundness and completeness of the *PSLD*-resolution.

The set V of consistent success vectors yielded by application of PSLD-resolution for $\Gamma_e \cup \{- A\}$ *is complete and contains all prime implicants of A i.e* $E(V) = T(A)$.

Proof. Our proof is based on the soundess and completeness of the resolution principle proved elsewhere (see [10] for example).

Soundness: $E(V) \subseteq T(A)$

Let $v_i \in E(V)$. By definition of $E(V)$, there is a success vector v which has v_i as a consistent ground instance. The success derivation of *PSLD*-resolution for $\Gamma_e \cup \{- A\}$ may be reduced to a normal success *SLD*-derivation for $\Gamma_e(v) \cup \{- A\}$. By the soundness of resolution principle we have $\Gamma_e(v) \models A$. Since v_i is a consistent instance of v we have $\Gamma_e(v_i) \models \Gamma_e(v)$. Therefore $\Gamma_e(v_i) \models A$. It means $v_i \in T(A)$.

Completeness. $T(A) \subseteq E(V)$

Let $t \in T(A)$. By definition $\Gamma^t \models A$, or $\Gamma_e(t) \models A$. By the completeness of *SLD*-resolution, there is a refutation for $\Gamma_e(t) \cup \{- A\}$. The corrsponding derivation in the *PSLD*-tree for $\Gamma_e \cup \{- A\}$ will terminate with a success resolvent which corresponds to a success vector v. t is one of consistent ground instances of v. It means $t \in E(V)$. Moreover if t is a maximal Γ-supporting vector, then by the same argument we can conclude that t is in V. Q.E.D.

4. Conclusion and discussion

The goal of this paper has been to provide an approximate method for solving the entailment problem in the interval-valued probabilistic logic. The problem considered in the framework of Nilsson's semantics amounts to two linear programming problems which are usually of very large size. The key idea of our approximate algorithm is to find for a target sentence S, given a knowledge base $\mathcal{B}$ with logical skeleton Γ, the set M of all Γ-supporting vectors for S and the set $\mathcal{M}$ of all maximal reduced subsets of M. From set $\mathcal{M}$ we can calculate easily a bellow approximate value for the lower bound of the truth probability of S. This appoximate method allows us to avoid linear progamming problems of large sizes. It is shown to be very efficient for probabilistic logic programs, i.e., when logical skeletons of knowledge

bases are usual logic program. In this case the set M of Γ-supporting vectors for S can be found by applying SLD-resolution for a certain extention of Γ. The solution obtained by the proposed method, as has been shown by our experiments, are very closed to - and in many cases, are coincided with the results given by solving corresponding linear programming problems.

To represent bases of knowledge under uncertainty, a set of propositional sentences weighted with two values in the unit interval was used by many researchers prior to us [1], [5], [7] and [16]. But the syntax may be only thing shared by those approaches. The semantics underlying the weights-numbers differ from one to another. The rule of uncertainty propagation in the support logic programming [1,2,3] is justified by voting model and fuzzy set. Neccesity-valued knowledge base [5] has fuzzy theory semantics. Among probabilistic approaches [13], [16] and [7], the distinguishing feature of our is preserving the uniform (declarative) style of treatment for "rule" and "fact" knowledge of logic programming. Here, we do not have to invent an explicit mechanism of uncertainty propagation. In our method, the classic machinery of resolution is exploited instead.

The accuracy of an approximate method is always a vital question. Assumed Nilsson's semantic, the interval found by linear programming method is the best (tightest), it would be intersting to ask a question: what relation the interval calculated by proposed algorithms forms with the tightest one. At this moment, we are not able to provide a absolute estimation except that the later lays inside the former. But we have an evidence that the accuracy of our method is good comparatively with proposed in literature rules of probability propagation. In [7], Frisch and Haddawy presented a comprehensive set of rules which had inherited many proposed in ealier works. These rules work with conditional probability. It is interesting to note that if we reduce them to unconditionan cases, they could be modeled by our method, i.e, they can be derived as special cases of our algorithms. It means that with restriction to unconditional probabilities our method would provide better approximation.

The serious estimation of this approximate method is a subject of our further work.

References

1. Baldwin J.F., *Support logic programming*, International Journal of Intelligent Systems, 1, 1986, 73-104.
2. Baldwin J.F., *Evidental support logic programming*, Fuzzy sets and systems, 24, 1987, 1-26.
3. Baldwin J.F., *Computational models of uncertainty reasoning in expert systems*, Computer and Math. with Appli., 19, 1990, 105-119.
4. Bundy A., *Incidence calculus: a mechanism for probabilistic reasoning*, Journal of

automated reasoning, 1. 1985, 263-283.
5. Dubois D., Lang J. and Prade H., *Possibilistic logic*, Technical report, IRIT, University Paul Sabatier, Toulouse, France 1991.
6. Dubois D. and Prade H., *A discussion of uncertainty handling in support logic programming, International Journal of Intelligent Systems, 5, 1990, 15-42.*
7. *Frisch A.M. and Haddawy P., Anytime deduction for probabilistic logic, Technical report, University of Wisconsin, Milawkee. 1992.*
8. *van Gelder, Negation as failure using tight derivation for general logic programs, Foundation of deductive databases and logic programming (J. Minker ed.) Morgan Kaufmann, 1988.*
9. *Genesereth M.R. and Nilsson N.J, Logical foundations of artificial intelligence, Morgan Kaufmann, 1987.*
10. *Lloyd J.W, Foundation of logic programming, Second edition, Springer Verlag, 1987.*
11. *Nilsson N.J., Probabilistic logic. Artificial Intelligence, 28, 1986, 71-87.*
12. *Phan D.D., Probabilistic logic for approximate reasoning, Artificial Intelligence and Information control systems of Robot-89, North Holland 1989, 107-112.*
13. *Pearl J., Probabilistic reasoning in intelligent systems. Morgan Kaufmann 1988.*
14. *Phan D.D. On the interval-valued probabilistic logic. Technical Report, ICS, NCSR of Vietnam, 1990.*
15. *Phan H.G., A method for detecting infinitness for Prolog programs, Proceedings of Pacific Rim International Conference on AI, Nagoya, Japan, 1990.*
16. *Raymond Ng. and Subrahmanian V.S., Probabilistic logic programming, Information and Computation, 101, 1992. 150-201*
17. *Quinlan J.R., INFERNO: A cautious approach to uncertain inference, Computer Journal, 26, 1983, 255-269.*

Abstract

This paper presents an approximate method for the probabilistic entailment problem in knowledge bases where a portion of knowledge is given by a sentence in propositional logic accompanied with an interval representing its truth probability. This method reduces the entailment problem to one of finding "prime implicants" of the target sentence expressed through sentences in the given knowledge base. It is shown that in the case of probabilistic logic programs the set of such prime implicants can be found by using the SLD-resolution method for usual definite logic programs.