The case of Logic Programs

Fuzzy LPs Basics

- We consider fuzzy LPs, which extends classical LPs, where
  - Truth space is $[0, 1]$
  - Interpretation is a mapping $I : B_{\mathcal{P}} \longrightarrow [0, 1]$
  - Generalized LP rules are of the form

$$R(\mathbf{x}) \leftarrow \exists \mathbf{y}.f(R_1(\mathbf{z}_1), \ldots, R_k(\mathbf{z}_k), p_1(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h))$$

    - Meaning of rules: "take the truth-values of all $R_i(\mathbf{z}_i), p_j(\mathbf{z}'_j)$, combine them using the truth combination function f, and assign the result to $R(\mathbf{x})$"
  - Facts: ground expressions of the form $\langle R(\mathbf{c}), n \rangle$
    - Meaning of facts: "the degree of truth of $R(\mathbf{c})$ is at least n"
  - Fuzzy LP: a set of facts (extensional database) and a set of rules (intentional database). No extensional relation may occur in the head of a rule

- Rules:

$$R(\mathbf{x}) \leftarrow \exists \mathbf{y}.\varphi(\mathbf{x}, \mathbf{y})$$

1. $\mathbf{x}$ are the *distinguished variables*;
2. s is the *score variable*, taking values in $[0, 1]$;
3. $\mathbf{y}$ are existentially quantified variables, called *non-distinguished variables*;
4. $\varphi(\mathbf{x}, \mathbf{y})$ is $f(\mathbf{R}(\mathbf{z}), \mathbf{p}(\mathbf{z}'))$, where $\mathbf{R}$ is a vector of predicates R_i and $\mathbf{p}$ is a vector of fuzzy predicates p_j;
5. $\mathbf{z}, \mathbf{z}'$ are tuples of constants in *KB* or variables in $\mathbf{x}$ or $\mathbf{y}$;
6. p_j is an n_j-ary *fuzzy predicate* assigning to each n_j-ary tuple $\mathbf{c}_j$ the *score* $p_j(\mathbf{c}_j) \in [0, 1]$;
7. f is a monotone *scoring* function $f\colon [0, 1]^{k+h} \longrightarrow [0, 1]$, which combines the scores of the h fuzzy predicates $p_j(\mathbf{c}_j)$ with the k scores $R_i(\mathbf{c}_i)$

Semantics of fuzzy LPs

- $\mathcal{P}^*$ is constructed as follows (as for the classical case):
  1. set $\mathcal{P}^*$ to the set of all ground instantiations of rules in $\mathcal{P}$;
  2. replace a fact $p(\mathbf{c})$ in $\mathcal{P}^*$ with the rule $p(\mathbf{c}) \leftarrow 1$
  3. if atom A is not head of any rule in $\mathcal{P}^*$, then add $A \leftarrow 0$ to $\mathcal{P}^*$;
  4. replace several rules in $\mathcal{P}^*$ having same head

$$\left.\begin{array}{l} A \leftarrow \varphi_1 \\ A \leftarrow \varphi_2 \\ \quad \vdots \\ A \leftarrow \varphi_n \end{array}\right\} \text{ with } A \leftarrow \varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_n \, .$$

- Note: in $\mathcal{P}^*$ each atom $A \in B_{\mathcal{P}}$ is head of exactly one rule
- Herbrand Base of $\mathcal{P}$ is the set $B_{\mathcal{P}}$ of ground atoms
- Interpretation is a function $I : B_{\mathcal{P}} \rightarrow [0, 1]$.
- Model $I \models \mathcal{P}$ iff for all $r \in \mathcal{P}^*$ $I \models r$, where $I \models A \leftarrow \varphi$ iff $I(\varphi) \leq I(A)$
- Note:

$$I(f(R_1(\mathbf{c}_1), \ldots, R_k(\mathbf{c}_k), p_1(\mathbf{c}'_1), \ldots, p_h(\mathbf{c}'_h))) = f(I(R_1(\mathbf{c}_1)), \ldots, I(R_k(\mathbf{c}_k)), p_1(\mathbf{c}'_1), \ldots, p_h(\mathbf{c}'_h)))$$

Fuzzy LP Query Answering

- Least model $M_{\mathcal{P}}$ of $\mathcal{P}$ exists and is least fixed-point of

$$T_{\mathcal{P}}(I)(A) = I(\varphi), \text{ for all } A \leftarrow \varphi \in \mathcal{P}^*$$

- M can be computed as the limit of

$$\begin{array}{lcl} \mathbf{I}_0 & = & \mathbf{0} \\ \mathbf{I}_{i+1} & = & T_{\mathcal{P}}(\mathbf{I}_i)\ . \end{array}$$

- Entailment: for a ground expression $\langle q(\mathbf{c}), s\rangle$, $s \in [0, 1]$

$$\mathcal{P} \models \langle q(\mathbf{c}), s\rangle \text{ iff least model of } \mathcal{P} \text{ satisfies } I(q(\mathbf{c})) \geq s$$

- We say that s is *tight* iff $s = \sup\{s' \mid \mathcal{P} \models \langle q(\mathbf{c}), s'\rangle\}$
- If $\mathcal{P} \models \langle q(\mathbf{c}), s\rangle$ and s is tight then $\langle \mathbf{c}, s\rangle$ is called an *answer* to q
- The answer set of q w.r.t. $\mathcal{P}$ is defined as

$$ans(\mathcal{P}, q) = \{\langle \mathbf{c}, s\rangle \mid \mathcal{P} \models \langle q(\mathbf{c}), s\rangle,\ s \text{ is tight}\}$$

Top-k Retrieval: Given a fuzzy LP $\mathcal{P}$, and a query q, retrieve k answers $\langle \mathbf{c}, s\rangle$ with maximal scores and rank them in decreasing order relative to the score s, denoted

$$ans_k(\mathcal{P}, q) = \text{Top}_k\ ans(\mathcal{P}, q)\ .$$

- Fuzzy LPs may be tricky:

$$\begin{array}{lcl} \langle A, 0 \rangle & & \\ A & \leftarrow & (A+1)/2 \end{array}$$

- In the minimal model the truth of A is 1 (requires ω $T_{\mathcal{P}}$ iterations)!
- There are several ways to avoid this pathological behavior:
  - We may consider $L = \{0, \frac{1}{n}, \frac{2}{n} \ldots, \frac{n-1}{n}, 1\}$, n natural number, e.g. $n = 100$
  - In $A \leftarrow f(B_1, \ldots, B_n)$, f is bounded, i.e. $f(x_1, \ldots, x_n) \leq x_i$

Example: Soft shopping agent

- I may represent my preferences in Logic Programming with the rules

$$
\begin{aligned}
Pref_1(x,p) &\leftarrow HasPrice(x,p) \wedge LS(10000,14000)(p)\\
Pref_2(x) &\leftarrow HasKM(x,k) \wedge LS(13000,17000)(k)\\
Buy(x,p) &\leftarrow 0.7 \cdot Pref_1(x,p) + 0.3 \cdot Pref_2(x)
\end{aligned}
$$

ID	MODEL	PRICE	KM
455	MAZDA 3	12500	10000
34	ALFA 156	12000	15000
1812	FORD FOCUS	11000	16000
⋮	⋮	⋮	⋮

- Problem: All tuples of the database have a score:
  - We cannot compute the score of all tuples, then rank them. Brute force approach not feasible.
- Top-k problem: Determine efficiently just the top-k ranked tuples, without evaluating the score of all tuples. E.g. top-3 tuples

ID	PRICE	SCORE
1812	11000	0.6
455	12500	0.56
34	12000	0.50

General top-down query procedure for Many-valued LPs

- Idea: use theory of fixed-point computation of equational systems over truth space (complete lattice or complete partial order)
- Assign a variable x_i to an atom $A_i \in B_{\mathcal{P}}$
- Map a rule $A \leftarrow f(A_1, \ldots, A_n) \in \mathcal{P}^*$ into the equation $x_A = f(x_{A_1}, \ldots, x_{A_n})$
- A LP $\mathcal{P}$ is thus mapped into the equational system

$$\begin{cases} x_1 & = & f_1(x_{1_1}, \ldots, x_{1_{a_1}}) \\ & \vdots & \\ x_n & = & f_n(x_{n_1}, \ldots, x_{n_{an}}) \end{cases}$$

- f_i is monotone and, thus, the system has least fixed-point, which is the limit of

$$\begin{array}{lcl} \mathbf{y}_0 & = & \mathbf{0} \\ \mathbf{y}_{i+1} & = & \mathbf{f}(\mathbf{y}_i)\ . \end{array}$$

  where $\mathbf{f} = \langle f_1, \ldots, f_n \rangle$ and $\mathbf{f}(\mathbf{x}) = \langle f_1(x_1), \ldots, f_n(x_n) \rangle$
- The least-fixed point is the least model of $\mathcal{P}$
- Consequence: If top-down procedure exists for equational systems then it works for fuzzy LPs too!

Procedure *Solve*$(\mathcal{S}, Q)$
Input: monotonic system $\mathcal{S} = \langle \mathcal{L}, V, \mathbf{f} \rangle$, where $Q \subseteq V$ is the set of query variables;
Output: A set $B \subseteq V$, with $Q \subseteq B$ such that the mapping v equals lfp(f) on B.

1. $A := Q$, $dg := Q$, $in := \emptyset$, **for all** $x \in V$ **do** $v(x) = 0$, $exp(x) = 0$
2. **while** $A \neq \emptyset$ **do**
3. **select** $x_i \in A$, $A := A \setminus \{x_i\}$, $dg := dg \cup s(x_i)$
4. $r := f_i(v(x_{i_1}), ..., v(x_{i_{a_i}}))$
5. **if** $r \succ v(x_i)$ **then** $v(x_i) := r$, $A := A \cup (p(x_i) \cap dg)$ **fi**
6. **if not** $exp(x_i)$ **then** $exp(x_i) = 1$, $A := A \cup (s(x_i) \setminus in)$, $in := in \cup s(x_i)$ **fi**

od

For $q(\mathbf{x}) \leftarrow \phi \in \mathcal{P}$, with $\mathrm{s}(q)$ we denote the set of *sons* of q w.r.t. r, i.e. the set of intentional predicate symbols occurring in ϕ. With $\mathrm{p}(q)$ we denote the set of *parents* of q, i.e. the set $\mathrm{p}(q) = \{p_i : q \in \mathrm{s}(p_i, r)\}$ (the set of predicate symbols directly depending on q).

Top-k retrieval in LPs

- If the database contains a huge amount of facts, a brute force approach fails:
  - one cannot anymore compute the score of all tuples, rank all of them and only then return the top-k
- Better solutions exists for restricted fuzzy LP languages: Datalog + restriction on the score combination functions appearing in the body

Basic Idea

- We do not compute all answers, but determine answers incrementally
- At each step i, from the tuples seen so far in the database, we compute a threshold δ
- The threshold δ has the property that any successively retrieved answer will have a score $s \leq \delta$
- Therefore, we can stop as soon as we have gathered k answers above δ, because any successively computed answer will have a score below δ

Procedure *TopAnswers*($\mathcal{K}$, Q, k)
Input: KB $\mathcal{K}$, intensional query relation symbol Q, $k \geq 1$;
Output: Mapping `rankedList` such that `rankedList`(Q) contains top-k answers of Q
Init: $\delta = 1$, **for all** rules $r : P(\mathbf{x}) \leftarrow \phi$ in P **do**
if P intensional **then** `rankedList`(P) $= \emptyset$;
if P extensional **then** `rankedList`(P) $= T_P$ **endfor**

1. **loop**
2. `Active` $:= \{Q\}$, `dg` $:= \{Q\}$, `in` $:= \emptyset$,
for all rules $r : P(\mathbf{x}) \leftarrow \phi$ **do** `exp`(P, r) $=$ `false`;
3. **while** (`Active` $\neq \emptyset$) **do**
4. **select** $P \in$ `A` where $r : P(\mathbf{x}) \leftarrow \phi$, `Active` $:=$ `Active` $\setminus \{P\}$, `dg` $:=$ `dg` $\cup$ `s`(P, r);
5. $\langle \mathbf{t}, s \rangle :=$ *getNextTuple*(P, r)
6. **if** $\langle \mathbf{t}, s \rangle \neq$ `NULL` **then** insert $\langle \mathbf{t}, s \rangle$ into `rankedList`(P),
`Active` $:=$ `Active` $\cup$ (`p`(P) $\cap$ `dg`);
7. **if not** `exp`(P, r) **then** `exp`(P, r) $=$ `true`,
`Active` $:=$ `Active` $\cup$ (`s`(P, r) $\setminus$ `in`), `in` $:=$ `in` $\cup$ `s`(p, r);
endwhile
8. Update threshold δ;
9. **until** (`rankedList`(Q) does contain k top-ranked tuples with score above δ)
or (`rL`$' =$ `rankedList`);
10. **return** top-k ranked tuples in `rankedList`(Q);

Procedure *getNextTuple*(P, r)
Input: intensional relation symbol P and rule $r : P(\mathbf{x}) \leftarrow \exists \mathbf{y}.f(R_1(\mathbf{z}_1), \ldots, R_n(\mathbf{z}_l)) \in P$;
Output: Next tuple satisfying the body of the r together with the score
Init:
loop
1. Generate next new instance tuple $\langle \mathbf{t}, s \rangle$ of P, using tuples in `rankedList`(R_i) and (RankSQL or Postgres)
2. **if** there is no $\langle \mathbf{t}, s' \rangle \in$ `rankedList`(P, r) with $s \leq s'$ **then** exit loop

until no new valid join tuple can be generated
3. **return** $\langle \mathbf{t}, s \rangle$ if it exists **else return** `NULL`

Example

Logic Program $\mathcal{P}$ is

$$q(x) \leftarrow p(x)$$
$$p(x) \leftarrow \min(r_1(x, y), r_2(y, z))$$

RecordID	r_1			r_2		
1	a	b	1.0	m	h	0.95
2	c	d	0.9	m	j	0.85
3	e	f	0.8	f	k	0.75
4	l	m	0.7	m	n	0.65
5	o	p	0.6	p	q	0.55
⋮	⋮	⋮	⋮	⋮	⋮	⋮

What is

$$Top_1(\mathcal{P}, q) = Top_1\{\langle c, s\rangle \mid \mathcal{P} \models q(c, s)\} \ ?$$

$$q(x) \leftarrow p(x)$$
$$p(x) \leftarrow \min(r_1(x, y), r_2(y, z))$$

	RecordID	r_1			r_2			
	1	*a*	*b*	1.0	*m*	*h*	0.95	
	2	*c*	*d*	0.9	*m*	*j*	0.85	
	3	*e*	*f*	0.8	*f*	*k*	0.75	←
→	4	*l*	*m*	0.7	*m*	*n*	0.65	
	5	*o*	*p*	0.6	*p*	*q*	0.55	
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	

Action: STOP, top-1 tuple score is equal or above threshold $0.75 = \max(\min(1.0, 0.75), \min(0.7, 0.95))$

Queue	δ
–	0.75

Predicate	*Answers*
q	$\langle e, 0.75\rangle \langle l, 0.7\rangle$
p	$\langle e, 0.75\rangle, \langle l, 0.7\rangle$

$$Top_1(\mathcal{P}, q) = \{\langle e, 0.75\rangle\}$$

Note: no further answer will have score above threshold δ

Threshold computation

For an intentional predicate p, head of a rule $r : p(\mathbf{x}) \leftarrow f(p_1, p_2, \ldots, p_n)$.

- consider a threshold variable δ^p
- with $r.\mathbf{t}^{\perp}_{p_i}$ ($r.\mathbf{t}^{\top}_{p_i}$) we denote the last tuple seen (the top ranked one) in $\texttt{rankedList}(p, r)$
- we define

$$\begin{aligned} p_i^{\top} &= \max(\delta^{p_i}, r.\mathbf{t}^{\top}_{p_i}.score) \\ p_i^{\perp} &= \delta^{p_i} \end{aligned}$$

- if p_i is an extensional predicate, we define

$$\begin{aligned} p_i^{\top} &= r.\mathbf{t}^{\top}_{p_i}.score \\ p_i^{\perp} &= r.\mathbf{t}^{\perp}_{p_i}.score \end{aligned}$$

- for rule r we consider the equation $\delta(r)$

$$\delta^p = \max(f(p_1^{\perp}, p_2^{\top}, \ldots, p_n^{\top}), f(p_1^{\top}, p_2^{\perp}, \ldots, p_n^{\top}), \ldots, f(p^{\top}, p^{\top}, \ldots, p_n^{\perp}))$$

- consider the set of equations of all equations involving intentional predicates, i.e.

$$\Delta = \bigcup_{r \in P} \{\delta(r)\} .$$

- for a query $q(\mathbf{x})$, the threshold δ of the *TopAnswers* algorithm is defined as to be

$$\delta = \bar{\delta}^q ,$$

where $\bar{\delta}^q$ is the solution to δ^q in the minimal solution $\bar{\Delta}$ of the set of equations Δ.

- note that $\bar{\delta}^q$, can be computed iteratively as least fixed-point