Qualitative Disjunctive Logic Programs

Rachel Ben-Eliyahu-Zohary
Azrieli College of Engineering
Jerusalem, Israel
What is QDLP?

Example:

Bird(x) $\leftarrow$ Penguin(x) \hspace{1cm} 1.0
Mammal(x) $\leftarrow$ Cat(x) \hspace{1cm} 1.0
$\neg$ Fly(x) $\leftarrow$ Penguin(x), not Fly(x) \hspace{1cm} 0.99
Fly(x) $\leftarrow$ Bird(x), not $\neg$ Fly(x) \hspace{1cm} 0.9
Cat(Pit) | Penguin(Pit) $\leftarrow$ \hspace{1cm} 0.2 | 0.5

Numbers in RED – degree of belief which we call Dominance
What is QDLP? (cont.)

Bird(x) ⇐ Penguin(x) 1.0
Mammal(x) ⇐ Cat(x) 1.0
¬Fly(x) ⇐ Penguin(x), not Fly(x) 0.99
Fly(x) ⇐ Bird(x), not ¬Fly(x) 0.9
Cat(Pit) | Penguin(Pit) ⇐ 0.2 | 0.5

Models:
1. Penguin(Pit) (0.5), Bird(Pit) (0.5), Fly(Pit) (0.45)
2. Penguin(Pit) (0.5), Bird(Pit) (0.5), ¬Fly(Pit) (0.495)
3. Cat(Pit) (0.2), Mammal(Pit) (0.2)
What is QLP? (cont.)

Models:
1. Penguin(Pit) (0.5), Bird(Pit) (0.5), Fly(Pit) (0.45)
2. Penguin(Pit) (0.5), Bird(Pit) (0.5), ¬Fly(Pit) (0.495)
3. Cat(Pit) (0.2), Mammal(Pit) (0.2)

Numbers is RED – degree of belief which we call dominance
What is QDLP?

• QDLP is a disjunctive logic program where a positive number not greater than 1 called dominance is associated with each literal in the head of each rule in the program.

• The intuition is that literals with higher dominance are more plausible or more reliable.

• Literals in the answer sets of QDLPs are also annotated with weights, with the intuition that a literal with a higher weight has stronger evidence.
What’s new?!

None of the (many!) qualitative logic programs suggested in the past have **ALL** of the characteristics listed below:

- The programs are a generalization of extended disjunctive logic programs under answer set semantics
- Each literal in head of rule in the program has a degree of belief
- Each literal in an answer set has a degree of belief
- The answer sets of the programs can be computed very efficiently using the *state-of-the-art* systems for computing answer sets of extended logic programs
Motivation 1 – Ontology Matching

1. Vehicles
   - Cars
     - Diesel
     - Electronic
   - Trucks
     - Pickup
   - Airplanes

2. Land Vehicles
   - Car
     - Hybrid
     - Diesel
   - Train
Ontology Matching

**Ontology 1:**

- Concept(V), Concept(C_s), Concept(T), Concept(A), Concept(D), Concept(E), Concept(P).
- IsA(C,V), IsA(T,V), IsA(A,V), IsA(D,C), IsA(E,C), IsA(P,T).

**Ontology 2:**

- Concept(LV), Concept(C), Concept(T), Concept(H).
- IsA(C,LV), IsA(T,LV), IsA(H,C), IsA(D,C).

**Child to Father matching default:**

\[ \text{Match}(x,y) \iff \text{Match}(x',y') \land \text{IsA}(x',x) \land \text{IsA}(y',y) \land \neg \text{Match}(x,y). \]
Motivation 1 – Ontology Matching

1. Vehicles
   - Cars
     - Diesel
     - Electronic
   - Trucks
     - Pickup
   - Airplanes

2. Land Vehicles
   - Car
     - Hybrid
     - Diesel
   - Train
More Motivation - Ontology Matching

• **Child to Father** matching default:
  \[
  \text{Match}(x, y) \leftarrow \text{Match}(x', y') , \text{IsA}(x', x) , \text{IsA}(y', y) ,\]
  \[
  (0.8)
  \not \text{Match}(x, y)
  \]

• **Father to Child** matching default
  \[
  \text{Match}(x', y') \mid \text{Match}(x', z') \leftarrow \text{Match}(x, y) , \text{IsA}(x', x) , \text{IsA}(y', y) \text{IsA}(z', y) , \not \text{Match}(x', y'),
  \not \text{Match}(x', z')
  \]
  \[
  (0.3) \mid (0.3)
  \]
Computing QAS-using existing methods

• First, we will compute all answer sets of the program while ignoring the numbers (using dlv etc)

• Then, for each answer set, apply the **Proof-Rank Algorithm** to compute the weight of each literal in the answer set

• **Proof-Rank Algorithm is Polynomial!!**
Proof wrt an Answer Set [BeDe94]

• Lemma: Let $P$ be a head cycle free (HCF) extended disjunctive logic program and let $S$ be an answer set of $P$. Then a literal $L$ is in $S$ iff $L$ has a proof with respect to $P$ and $S$.

• A proof of a literal is a sequence of rules that can be used to derive the literal from the program.

• The proof is wrt an answer set $S$ and a program $P$. 
Proof [BeDe94]

• A literal $L$ has a proof w.r.t. a set of literals $S$ and a logic program $P$ if and only if there is a sequence of rules $\delta_1 \ldots \delta_n$ from $P$ such that:
  
• for all $1 \leq i \leq n$ there is one and only one literal in the head of $\delta_i$ that belongs to $S$.  

Rachel Ben-Eliyahu-Zohary, Azrieli College of Engineering, Jerusalem
Proof (cont.)

• L is the head of $\delta_n$.
• for every $\delta_i$ $1 \leq i \leq n$, the body of $\delta_i$ is satisfied by S
• $\delta_1$ has no literals that appear positive in its body, and for each $1 < i \leq n$, each literal that appears positive in the body of $\delta_i$ is in the head of some $\delta_j$ for some $1 \leq j < i$. 
Proof (cont.)

Note that given a proof $\delta_1\cdots\delta_n$ of some literal $L$ w.r.t. a set of literals $S$ and a logic program $P$, for every $1 \leq i \leq n$, $\delta_1\cdots\delta_i$ is also a proof of some literal $L'$ w.r.t. $S$ and $P$. 
Lemma:
Let $P$ be an HCF EDLP. Then a set of literals $S$ is an answer set of $P$ if and only if each literal in $S$ has a proof with respect to $P$ and $S$
Dominance of a Literal in $S$

The dominance of the proof $b_1 \ldots b_n$ is $b(b_1) \ast \ldots \ast b(b_n)$ where $b(b_i)$ is the dominance of the only literal in the head of $b_i$ that belongs to $S$.

The dominance of a literal $L$ is the maximum dominance of any proof of $L$ w.r.t. $S$ and $P$. 
Monotonicity

A QDLP semantics preserves the *monotonicity* property if and only if given a QDLP $P$ and an answer set $S$ for $\hat{P}$, for every proof $\mu$ of any literal $L$ with respect to $P$ and $S$ and for every proof $\mu'$ which is a sub-proof of $\mu$, $\text{dominance}(\mu) \leq \text{dominance}(\mu')$.

A QDLP semantics that preserves the monotonicity property is called *monotonic* QDLP.
Computing QAS – using existing methods

• First, we will compute all answer sets of the program while ignoring the numbers.

• Then, for each answer set, apply the **Proof-Rank Algorithm** to compute the weight of each literal in the answer set

• **Proof-Rank Algorithm** is Polynomial!!
Qualitative Logic Program Example

• Consider the following Qualitative Logic Program (QLP) $\Pi$:

1. $\neg D \leftarrow not \ C \ (0.4)$
2. $C \leftarrow \neg D \ (0.9)$
3. $C \leftarrow \ (0.8)$
4. $A \mid E \leftarrow not \ B \ (0.5 \mid 0.9)$
5. $\neg D \leftarrow A, \ C \ (0.8)$
6. $\neg D \leftarrow C \ (1.0)$
7. $\neg D \mid C \leftarrow E \ (0.6 \mid 0.5)$

• 2 answer sets: $\{A,C, \neg D\}$, $\{C, \neg D,E\}$
Proof-Rank Algorithm

• Suppose we are given the program \( P \):

1. \( \neg D \leftarrow \text{not } C \) (0.4)
2. \( C \leftarrow \neg D \) (0.9)
3. \( C \leftarrow \) (0.8)
4. \( A \mid E \leftarrow \text{not } B \) (0.5 | 0.9)
5. \( \neg D \leftarrow A, C \) (0.8)
6. \( \neg D \leftarrow C \) (1.0)
7. \( \neg D \mid C \leftarrow E \) (0.6 | 0.5)

And \( S= \{A, C, \neg D\} \).

Red(\( P, S \)) is:

2. \( C \leftarrow \neg D \) (0.9)
3. \( C \leftarrow \) (0.8)
4. \( A \mid E \leftarrow \) (0.5 | 0.9)
5. \( \neg D \leftarrow A, C \) (0.8)
6. \( \neg D \leftarrow C \) (1.0)
7. \( \neg D \mid C \leftarrow E \) (0.6 | 0.5)
Next Step

Leave only rules with only literals from S in the body and exactly one literal in the head that belongs to S

So the reduct

2. \( C \leftarrow \neg D \) (0.9)
3. \( C \leftarrow (0.8) \)
4. \( A \mid E \leftarrow \) (0.5 | 0.9)
5. \( \neg D \leftarrow A, C \) (0.8) \( S = \{A,C, \neg D\} \).
6. \( \neg D \leftarrow C \) (1.0)
7. \( \neg D \mid C \leftarrow E \) (0.6 | 0.5)

Becomes

2. \( C \leftarrow \neg D \) (0.9)
3. \( C \leftarrow (0.8) \)
4. \( A \leftarrow (0.5) \)
5. \( \neg D \leftarrow A, C \) (0.8)
6. \( \neg D \leftarrow C \) (1.0)
Data Structures of Proof-Rank

• For each rule $r$ in $\Pi$:
  – $\text{body}[r]$ – the number of literals in the body of $r$ for which the best proof was not found yet

• For each literal $L$ in the answer set:
  – $\text{dom}[L]$ – the dominance of $L$ computed so far
  – $\text{proof}[L]$ – the best proof for $L$ found so far
  – $\text{final}[L]$ – a boolean indicating whether the best proof for $L$ was found

• A heap where all the literals $L$ with $\text{dom}>0$ are kept according to their dominance
Proof-Rank Main Steps

1. Initialize all data structures
2. Sort the literals in the heap
3. While the heap is not empty:
   i. \( L = \text{top}[\text{heap}] \)
   ii. \( \text{final}[L] = \text{true} \)
   iii. For each rule \( r \) such that \( L \) is in the body of \( r \) and the dominance of the literal in the head of \( r \) is not final, decrease body[\( r \)] by 1 and if body[\( r \)] = 0 update the literal in the head of \( r \)
Back to the Example...

**Reduct**

2. $C \leftarrow \neg D$ (0.9)  
3. $C \leftarrow$ (0.8)  
4. $A \leftarrow$ (0.5)  
5. $\neg D \leftarrow A, C$ (0.8)  
6. $\neg D \leftarrow C$ (1.0)

$\text{body}[2] = 1$  
$\text{body}[3] = 0$  
$\text{body}[4] = 0$  
$\text{body}[5] = 2$  
$\text{body}[6] = 1$

$S = \{A, C, \neg D\}$.

**Heap**

**Literals**

<table>
<thead>
<tr>
<th>Literal</th>
<th>dom</th>
<th>proof</th>
<th>final</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>0.5</td>
<td>${4}$</td>
<td>false</td>
</tr>
<tr>
<td>$C$</td>
<td>0.8</td>
<td>${3}$</td>
<td>false</td>
</tr>
<tr>
<td>$\neg D$</td>
<td></td>
<td></td>
<td>false</td>
</tr>
</tbody>
</table>
Back to the Example...

Reduct

2. \( C \leftarrow \neg D \) (0.9) \( \text{body}[2] = 1 \)
3. \( C \leftarrow \) (0.8) \( \text{body}[3] = 0 \)
4. \( A \leftarrow \) (0.5) \( \text{body}[4] = 0 \)
5. \( \neg D \leftarrow A, C \) (0.8) \( \text{body}[5] = 1 \)
6. \( \neg D \leftarrow C \) (1.0) \( \text{body}[6] = 0 \)

Heap

top \rightarrow \text{C} \rightarrow \text{A}

Literals

<table>
<thead>
<tr>
<th>Literal</th>
<th>dom</th>
<th>proof</th>
<th>final</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>{4}</td>
<td>false</td>
</tr>
<tr>
<td>C</td>
<td>0.8</td>
<td>{3}</td>
<td>true</td>
</tr>
<tr>
<td>(\neg D)</td>
<td>0.8</td>
<td></td>
<td>false</td>
</tr>
</tbody>
</table>
Back to the Example...

Heap

Reduct

4. A ← (0.5)
5. ¬D ← A, C (0.8)
6. ¬D ← C (1.0)

body[4] = 0
body[5] = 2
body[6] = 0

Literals

<table>
<thead>
<tr>
<th>Literal</th>
<th>dom</th>
<th>proof</th>
<th>final</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>{4}</td>
<td>false</td>
</tr>
<tr>
<td>C</td>
<td>0.8</td>
<td>{3}</td>
<td>true</td>
</tr>
<tr>
<td>¬D</td>
<td>0.8</td>
<td></td>
<td>false</td>
</tr>
</tbody>
</table>
Back to the Example...

**Heap**

- top
- D
- A

**Reduct**

4. A ← (0.5)  
5. ¬D ← A, C (0.8)  
6. ¬D ← C (1.0)  

<table>
<thead>
<tr>
<th>Literal</th>
<th>dom</th>
<th>proof</th>
<th>final</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.5</td>
<td>{4}</td>
<td>false</td>
</tr>
<tr>
<td>C</td>
<td>0.8</td>
<td>{3}</td>
<td>true</td>
</tr>
<tr>
<td>¬D</td>
<td>0.8</td>
<td>{6}</td>
<td>true</td>
</tr>
</tbody>
</table>
Back to the Example...

Reduct
4. $A \leftarrow (0.5)$
$\text{body}[4] = 0$

Heap

Literal | dom | proof | final
--- | --- | --- | ---
$A$ | 0.5 | \{4\} | true
$C$ | 0.8 | \{3\} | true
$\neg D$ | 0.8 | \{6\} | true
Proof-Rank Algorithm – Result

• So for the program $P$:

1. $\neg D \leftarrow \text{not } C \ (0.4)$
2. $C \leftarrow \neg D \ (0.9)$
3. $C \leftarrow \ (0.8)$
4. $A \mid E \leftarrow \text{not } B \ (0.5 \mid 0.9)$
5. $\neg D \leftarrow A \ , \ C \ (0.8)$
6. $\neg D \leftarrow C \ (1.0)$
7. $\neg D \mid C \leftarrow E \ (0.6 \mid 0.5)$

One quantified answer set is:
$S = \{A \ (0.5) \ , \ C \ (0.8), \ \neg D \ (0.8)\}$
The LPmatch System
Two Ontologies to Match

1. vehicles
   - Cars
     - Diesel
     - Electronic
   - Trucks
     - Pickup
   - Airplanes

2. Land Vehicles
   - Car
     - Hybrid
     - Diesel
   - Train
LPmatch System Description

Step 1: Generating the QLP program

Given two ontologies to match, create 3 files:

File 1: contains all the ontology's entities and relations between them.

File 2: contains the matching facts provided by the first line matcher.

File 3: contains rules for the logic program. In the 1st stage: rules that create only a few number of models (less than 10).
Step 1: Generating the QLP program

**File 1**: contains all the ontology's entities and relations between them

- `class_gen(cs_department)`.  
- `class_gen(faculty)`.  
- `class_gen(exelent_students)`.  
- `...`  
- `class_numsubclasses_gen(faculty,4)`.
- `...`
Step 1: Generating the QLP program

File 1: contains all the ontology's entities and relations between them

- class_subclass_gen(faculty, cs_department).
- ...
- property_gen(hasStudents).
- property_gen(hasMembers).
- ...
- property_subproperty_gen(hasMembers, hasStudents).
- ...

Rachel Ben-Eliyahu-Zohary, Azrieli College of Engineering, Jerusalem
Step 1: Generating the QLP program

File 1: contains all the ontology's entities and relations between them

obp_domain_gen(hasStudents,cs_department).
opb_range_gen(hasStudents,exelent_students).
...
restriction_gen(faculty,hasMembers,card).
Step 1: Generating the QLP program

File 1: contains all the ontology's entities and relations between them

We do the same for specific ontology, just write "spec" instead of "gen"
Step 1: Generating the QLP program

File 2: contains the matching facts provided by the first line matcher

initial matches are based on the linguistic and statistical analysis of the ontology's entities

For example,

• match(classX, classY).
• p_match(propX, propY).
• ...

Rachel Ben-Eliyahu-Zohary, Azrieli College of Engineering, Jerusalem
Step 1: Generating the QLP program

File 2: contains the matching facts provided by the first line matcher

In the case that first line matcher of Falcon returns no matches, we try to find initial matches using structural similarities between ontologies.
Step 1: Generating the QLP program
Example of rules used at the 1\textsuperscript{st} stage

match(X1,X2):-
match(Y1,Y2),class_subclass_spec(X1,Y1),
class_subclass_gen(X2,Y2),not \neg\,\text{match}(X1,X2).

Classes $X_1$ and $X_2$ are matched if their subclasses $Y_1$ and $Y_2$ are matched
Dominance $= 0.8$
Step 1: Generating the QLP program

Example of rules used at the 1\textsuperscript{st} stage

\[ \neg \text{match}(X_1, X_2) : - \]
\[ \text{match}(X_1, Y_2), \text{class\_spec}(X_1), \text{class\_gen}(X_2), \]
\[ \text{class\_gen}(Y_2), X_2 \neq Y_2. \]

\[ \neg \text{match}(X_1, X_2) : - \]
\[ \text{match}(Y_1, X_2), \text{class\_spec}(X_1), \text{class\_spec}(Y_1), \]
\[ \text{class\_gen}(X_2), X_1 \neq Y_1. \]

Each class is matched only once
Step 1: Generating the QLP program

Example of rules used at the 1\textsuperscript{st} stage

\begin{verbatim}
\texttt{p\_match(X,Y):-
match(X1,Y1), match(X2,Y2),
obp\_domain\_spec(X,X1), obp\_domain\_gen(Y,Y1),
obp\_range\_spec(X,X2), obp\_range\_gen(Y,Y2),
not \neg p\_match(X,Y).}
\end{verbatim}

Object properties X and Y are matched if their domains X1, Y1 and their ranges X2, Y2 are matched

Dominance = 0.9
LPmatch System Description

Step 2: Run dlv

- If several models are generated, use intersection
- Run dlv several times, each time with new found matches as facts, and using all possible rules.
- In most cases we get a huge amount of models. We use only first 20-100 models for the further process
Rules added at Step 2: example

match(X1,X2) :-match(Y1,Y2),
class_subclass_spec(Y1,X1),
class_subclass_gen(Y2,X2),
    not \neg match(X1,X2).

Classes X1 and X2 are matched if their superclasses Y1 and Y2 are matched

Dominance = 0.3
Rules added at Step 2: example

match(X,Y):-
p_match(X1,Y1), obp_range_spec(X1,X), obp_range_gen(Y1,Y), not ¬match(X,Y).

Classes X and Y are matched if X is range of object property X1, Y is range of object property Y1 and object properties X1 and Y1 are matched

Dominance = 0.2
Evaluation Results
Screenshot of the LPmatch Tool
Evaluation Methodology

• A reference alignment (a set of correspondences between ontology elements) is generated manually.

• The quality measures are defined by comparing the matches proposed by a matcher to be evaluated to the reference alignment.

• Three measures are considered:
  – Precision is the fraction of real matches among proposed matches.
  – Recall is the fraction of proposed real matches among all real matches.
  – F-measure is a weighted harmonic mean of precision and recall.

• The evaluation was performed using the OAEI 2009 benchmark test library.
Comparison to other algorithms
(OAEI 2009 benchmark test library)

• Entire benchmark test library (tests 101 – 304):

• LPmatch ranked #5 (if would have participated in the campaign)
Comparison to other algorithms (OAEI 2009 benchmark test library)

- Systematic tests with unscrambled labels (232 – 247):

![Bar chart showing comparison of F-measure among different algorithms, with LPmatch ranked #1.](chart.png)
Comparison to other algorithms
(OAEI 2009 benchmark test library)

• Tests with scrambled labels, no comments or properties, and expanded hierarchy (261-2 – 261-8):

LPmatch ranked #1

Rachel Ben-Eliyahu-Zohary, Azrieli College of Engineering, Jerusalem
Comparison to Falcon

- As our “First Line Matcher” (FLM) we used the linguistic matcher of Falcon – a known and published OM algorithm.
- To validate the significance of LPmatch, we show a comparison to the results of Falcon (which did not participate in the OAEI 2009 campaign) on the entire benchmark (tests 101 – 304):

<table>
<thead>
<tr>
<th></th>
<th>Falcon</th>
<th>Basic LPmatch</th>
<th>LPmatch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>0.83</td>
<td>0.83</td>
<td>0.86</td>
</tr>
<tr>
<td>Recall</td>
<td>0.69</td>
<td>0.73</td>
<td>0.75</td>
</tr>
<tr>
<td>F-measure</td>
<td>0.73</td>
<td>0.77</td>
<td>0.78</td>
</tr>
</tbody>
</table>

- Basic LPmatch – a version of the algorithm that uses Falcon's linguistic matcher with no additions as its FLM.
- LPmatch – adds a simple FLM that is based on "structure" whenever Falcon's linguistic matcher fails to find even a single match.
QLP vs. Intersection

- We compare the performance of different versions of LPmatch based on the chosen maximal number of models.
- The versions vary in the strategy of computing the resulting model – using QLPs vs. intersecting all the models.

**Precision**

QLPs result in improved recall alongside impaired precision.

**Recall**

Semi-automatic systems value recall over precision.

- Number of models only slightly affects the QLP strategy.
Runtime Performance

• computing the models (done by DLV) is the most time-consuming operation in LPmatch's computation process
  – The resulting runtime of the two strategies is practically the same

• Computing only the first 20 models seems quite enough
Significance of Rules

- We incrementally add the rules
  - We do this for test 202 from the benchmark library of the OAEI-2009 campaign
  - It is a relatively hard test with no names or comments

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
<th>Added rules</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>First Line Matcher</td>
<td>-</td>
<td>0.12</td>
</tr>
<tr>
<td>1</td>
<td>Basic class rules</td>
<td>1,2,5,6,7</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>Object property rules</td>
<td>3,4</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>Subproperty rules</td>
<td>8,9</td>
<td>0.38</td>
</tr>
<tr>
<td>2</td>
<td>Superclass rule</td>
<td>10</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>Advanced property rules</td>
<td>11,12,13,14,15</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>Restriction rules</td>
<td>16,17</td>
<td>0.68</td>
</tr>
</tbody>
</table>

- When solving such a hard problem, each group of added rules may improve the matching
Conclusions

• Qualitative Logic Programs are of Practical Importance

• The version presented here is simple and easy to compute

• We need good methods for incremental model computation
Thank You!