Structure Learning with Distributed Parameter Learning for Probabilistic Ontologies

Giuseppe Cota[1] Riccardo Zese[1] Elena Bellodi[1]
Fabrizio Riguzzi[2] Evelina Lamma[1]

Dipartimento di Ingegneria – University of Ferrara

Dipartimento di Matematica e Informatica – University of Ferrara
[giuseppe.cota,riccardo.zese,elena.bellodi,fabrizio.riguzzi,
evelina.lamma]@unife.it

September 23, 2015

Outline

Probabilistic Description Logics: Introduction

- Basic Concepts:
  - **Ontology:** a formal and explicit description of a domain of interest
  - **Description Logics (DLs):** family of logical formalism (decidable) fragments of first order logic
  - **Knowledge Base:** set of axioms expressed by a DL

The semantics: DISPONTE

- In real world domains the information is often **uncertain**
- The **Distribution Semantics** underlies many probabilistic logic programming languages, such as PRISM, ICL, LPADs, ProbLog
- **DISPONTE applies the Distribution Semantics of probabilistic logic programming to description logics**
- DISPONTE semantics for probabilistic $\mathcal{SROIQ}(\mathbf{D})$.
  - Probabilistic axioms: $p :: E$
    e.g., $p :: C \sqsubseteq D$ represents the fact that we believe in the truth of $C \sqsubseteq D$ with probability p.

Example

$$0.4 :: \mathit{fluffy} : \mathit{Cat} \quad (1)$$
$$0.3 :: \mathit{tom} : \mathit{Cat} \quad (2)$$
$$0.6 :: \mathit{Cat} \sqsubseteq \mathit{Pet} \quad (3)$$

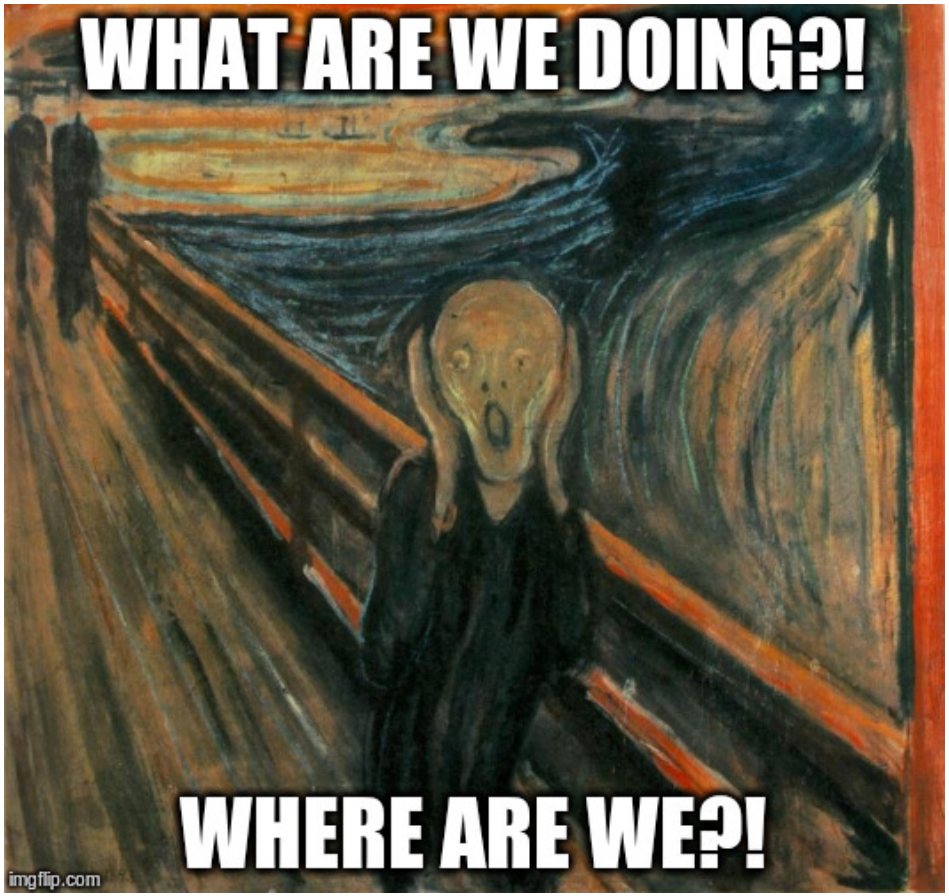
$$\exists \mathit{hasAnimal}.\mathit{Pet} \sqsubseteq \mathit{NatureLover} \quad (4)$$
$$(\mathit{kevin}, \mathit{fluffy}) : \mathit{hasAnimal} \quad (5)$$
$$(\mathit{kevin}, \mathit{tom}) : \mathit{hasAnimal} \quad (6)$$

- $Q = \mathit{kevin} : \mathit{NatureLover}$ has two explanations:

  $\{ (1), (3) \}$
  $\{ (2), (3) \}$

- $P(Q) = 0.4 \times 0.6 \times (1 - 0.3) + 0.3 \times 0.6 = 0.348$
- $P(Q)$ can be computed with the inference system **BUNDLE**

???



What does *structure learning* mean (in this context)?

Reasoning task

In the field of Probabilistic Inductive Logic Programming the reasoning task is composed by three main issues:

1. **Inference:** we want to compute the probability of a query
2. **Parameter Learning:** given a structure (the logical formulas), we want to know the parameters (weights) of the logical formulas
3. **Structure Learning:** we want to learn both the structure and the parameters

Reasoning task

In the field of Probabilistic Inductive Logic Programming the reasoning task is composed by three main issues:

1. **Inference:** we want to compute the probability of a query
2. **Parameter Learning:** given a structure (the logical formulas), we want to know the parameters (weights) of the logical formulas
3. **Structure Learning:** we want to learn both the structure and the parameters

Reasoning task

In the field of Probabilistic Inductive Logic Programming the reasoning task is composed by three main issues:

1. **Inference:** we want to compute the probability of a query
2. **Parameter Learning:** given a structure (the logical formulas), we want to know the parameters (weights) of the logical formulas
3. **Structure Learning:** we want to learn both the structure and the parameters

Reasoning task

In the field of Probabilistic Inductive Logic Programming the reasoning task is composed by three main issues:

1. **Inference:** we want to compute the probability of a query
2. **Parameter Learning:** given a structure (the logical formulas), we want to know the parameters (weights) of the logical formulas
3. **Structure Learning:** we want to learn both the structure and the parameters

Reasoning task

In the field of Probabilistic Inductive Logic Programming the reasoning task is composed by three main issues:

1. **Inference:** we want to compute the probability of a query
2. **Parameter Learning:** given a structure (the logical formulas), we want to know the parameters (weights) of the logical formulas
3. **Structure Learning:** we want to learn both the structure and the parameters • We are here!

Systems and Algorithms

1. Inference
   - **BUNDLE:** a system for reasoning on DISPONTE KBs using Binary Decision Diagrams (BDDs)
2. Parameter Learning
   - **EDGE:** an algorithm that learns the probabilistic parameters of a DISPONTE KB
   - **EDGE^{MR}**: distributed version of EDGE
3. Structure Learning
   - **LEAP:** structure learning algorithm for DISPONTE KBs
   - **LEAP^{MR}**: distributed version of LEAP

Systems and Algorithms

1. Inference
   - **BUNDLE:** a system for reasoning on DISPONTE KBs using Binary Decision Diagrams (BDDs)
2. Parameter Learning
   - **EDGE:** an algorithm that learns the probabilistic parameters of a DISPONTE KB
   - **EDGE^{MR}**: distributed version of EDGE
3. Structure Learning
   - **LEAP:** structure learning algorithm for DISPONTE KBs
   - **LEAP^{MR}**: distributed version of LEAP • We are here!

EDGE: Em over bDds for description loGics paramEter learning

- **Supervised parameter learning algorithm**
- **Input:** a DISPONTE theory, a set of examples that represent queries
  - The queries are assertional axioms divided into:

    positive examples information that we regard as true for which we would like to get high probability;

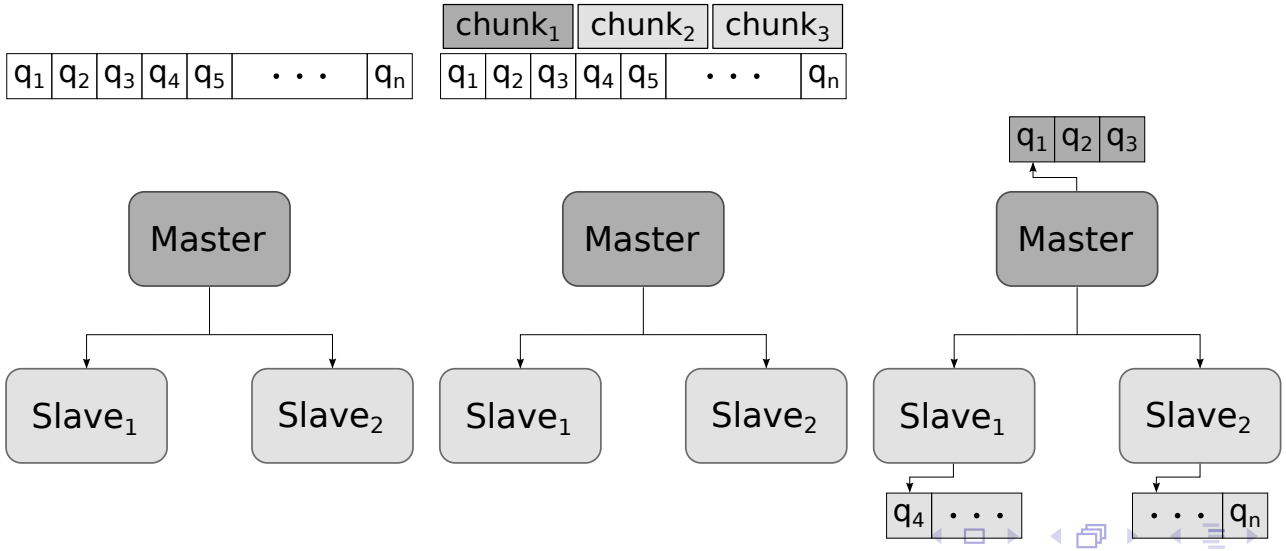
    negative examples information that we regard as false for which we would like to get low probability.
- **Compute the explanations** of each example using BUNDLE
- **Enter the EM cycle**
  - **Expectation**
    - computes the expected counts by traversing twice the BDDs
  - **Maximization**
    - computes maximum likelihood parameters from the expected counts
- The EM algorithm is guaranteed to find a local maximum of the probability of the examples

EDGE^{MR}: Distributed Parameter Learning by MapReduce

- Initialization: the data is replicated among all workers. Each worker parses its copy of the probabilistic knowledge base and stores it in main memory.
- Query resolution: the examples are divided among all the workers Each worker generates its private subset of BDDs and keeps them in memory. Two different scheduling techniques can be applied for this operation.
- Expectation-Maximization: during the Expectation step all the workers traverse their BDDs and calculate their local set of expectations. Then the master gathers all the expectations from the slaves, aggregates them and performs Maximization

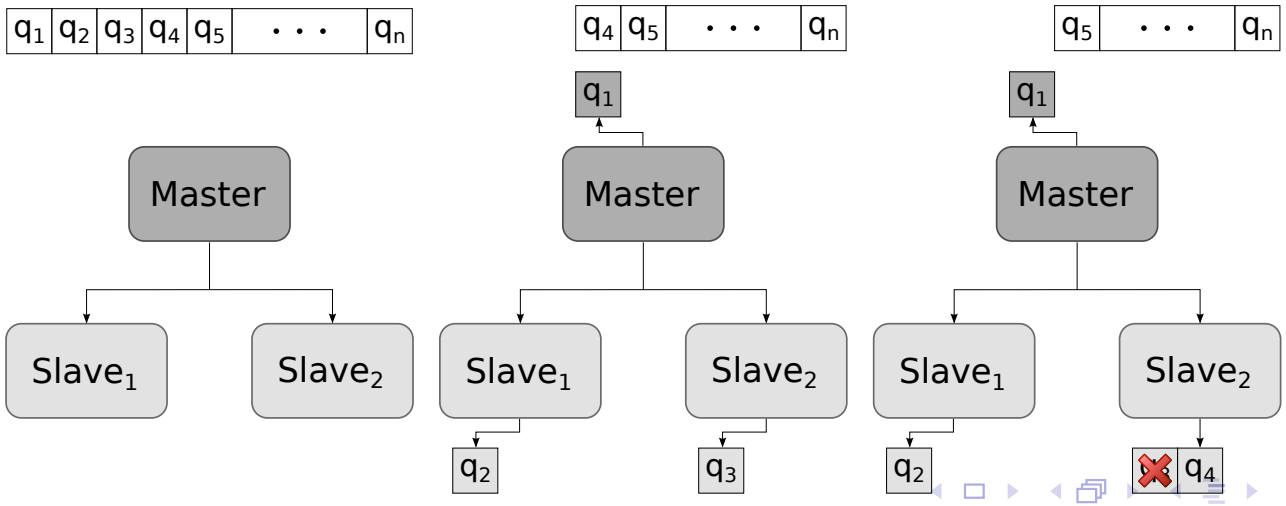
Scheduling Techniques
Single-step scheduling

- N slaves, the master divides the total number of queries into $N + 1$ chunks
- The master begins to compute its queries while, for each other chunk of queries, it sends the chunk to the corresponding slave.
- Then the master waits for the results from the slaves. When the slowest slave returns its results to the master, EDGE^{MR} proceeds to the EM cycle.



Scheduling Techniques
Dynamic scheduling

- Handling each query may require a different amount of time
- Chunk dimension established by the user
- At first each machine is assigned a chunk of query in order
- Then if the master ends handling its chunk it just takes the next one, instead, if a slave ends handling its chunk, it asks the master for another one
- When all the queries are evaluated, EDGE^{MR} starts the EM cycle



Class Expression Learning

Definition

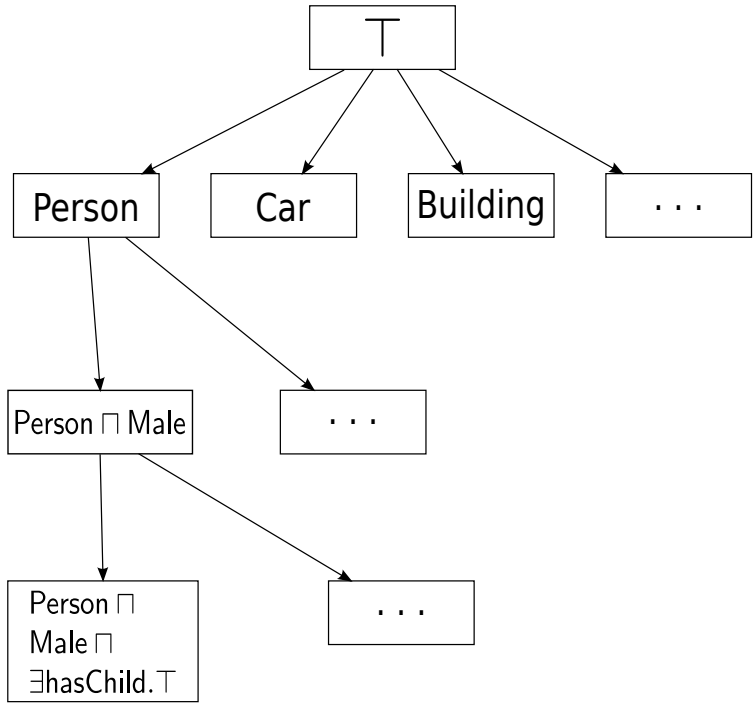
- Given a knowledge base $\mathcal{K}$
- Given a Target class contained in $\mathcal{K}$
- Given $R_{\mathcal{K}}(C)$ a retrieval function of the reasoner that returns all the instances of C

The class learning consists in finding an expression C such that:

$$R_{\mathcal{K}}(\text{Target}) = R_{\mathcal{K}}(C)$$

CELOE: Class Expression Learning for Ontology Engineering

- **C**lass **E**xpression **L**earning for **O**ntology **E**ngineering.
- Generates a set of Class Expressions C_i ordered according to an heuristics.
- Provides a semi-automatic approach to add axioms of the form $Target \equiv C_i$ o $Target \sqsubseteq C_i$.
- Combination of a refinement operator and a search algorithm
- Top-down
- Inside **DL-Learner** project



Put it all together!

- Our recipe:
  - BUNDLE: for Inference
  - EDGE: for parameter learning
  - CELOE: for candidate axioms generation

Put it all together!

- Our recipe:
  - BUNDLE: for Inference
  - EDGE: for parameter learning
  - CELOE: for candidate axioms generation

Put it all together!

- Our recipe:
  - BUNDLE: for Inference
  - EDGE: for parameter learning
  - CELOE: for candidate axioms generation

$\Downarrow$

LEAP

LEAP: LEArning Probabilistic description logics

- **Structure and parameter learner** for probabilistic KBs under DISPONTE
- Exploits two algorithms:

  CELOE (Class Expression Learning for Ontology Engineering) for learning the structure of a KB

  EDGE for learning the parameters of a probabilistic KB

LEAP: algorithm

1. Input: a knowledge base $\mathcal{K}$, a set of examples that represent **queries**

LEAP: algorithm

1. Input: a knowledge base $\mathcal{K}$, a set of examples that represent **queries**
2. Run CELOE to obtain a set C of candidate probabilistic subclass-of axioms

LEAP: algorithm

1. Input: a knowledge base $\mathcal{K}$, a set of examples that represent **queries**
2. Run CELOE to obtain a set C of candidate probabilistic subclass-of axioms
   - For each axiom, the probability is initialized to the accuracy returned by CELOE

LEAP: algorithm

1. Input: a knowledge base $\mathcal{K}$, a set of examples that represent **queries**
2. Run CELOE to obtain a set C of candidate probabilistic subclass-of axioms
   - For each axiom, the probability is initialized to the accuracy returned by CELOE
3. Run EDGE to compute the initial log-likelihood LL of $\mathcal{K}$

LEAP: algorithm

1. Input: a knowledge base $\mathcal{K}$, a set of examples that represent **queries**
2. Run CELOE to obtain a set C of candidate probabilistic subclass-of axioms
   - For each axiom, the probability is initialized to the accuracy returned by CELOE
3. Run EDGE to compute the initial log-likelihood LL of $\mathcal{K}$
4. For each $ax \in C$

LEAP: algorithm

1. Input: a knowledge base $\mathcal{K}$, a set of examples that represent **queries**
2. Run CELOE to obtain a set C of candidate probabilistic subclass-of axioms
   - For each axiom, the probability is initialized to the accuracy returned by CELOE
3. Run EDGE to compute the initial log-likelihood LL of $\mathcal{K}$
4. For each $ax \in C$
   1. Add the axiom ax to the knowledge base $\mathcal{K}$

LEAP: algorithm

1. Input: a knowledge base $\mathcal{K}$, a set of examples that represent **queries**
2. Run CELOE to obtain a set C of candidate probabilistic subclass-of axioms
   - For each axiom, the probability is initialized to the accuracy returned by CELOE
3. Run EDGE to compute the initial log-likelihood LL of $\mathcal{K}$
4. For each $ax \in C$
   1. Add the axiom ax to the knowledge base $\mathcal{K}$
   2. Run EDGE on the new KB to compute the new parameters and the new log-likelihood LL'

LEAP: algorithm

1. Input: a knowledge base $\mathcal{K}$, a set of examples that represent **queries**
2. Run CELOE to obtain a set C of candidate probabilistic subclass-of axioms
   - For each axiom, the probability is initialized to the accuracy returned by CELOE
3. Run EDGE to compute the initial log-likelihood LL of $\mathcal{K}$
4. For each $ax \in C$
   1. Add the axiom ax to the knowledge base $\mathcal{K}$
   2. Run EDGE on the new KB to compute the new parameters and the new log-likelihood LL'
   3. If $LL' > LL$ keep the axiom ax

LEAP: algorithm

1. Input: a knowledge base $\mathcal{K}$, a set of examples that represent **queries**
2. Run CELOE to obtain a set C of candidate probabilistic subclass-of axioms
   - For each axiom, the probability is initialized to the accuracy returned by CELOE
3. Run EDGE to compute the initial log-likelihood LL of $\mathcal{K}$
4. For each $ax \in C$
   1. Add the axiom ax to the knowledge base $\mathcal{K}$
   2. Run EDGE on the new KB to compute the new parameters and the new log-likelihood LL'
   3. If $LL' > LL$ keep the axiom ax
   4. Else remove it

LEAP^{MR}: Structure Learning with Distributed Parameter Learning

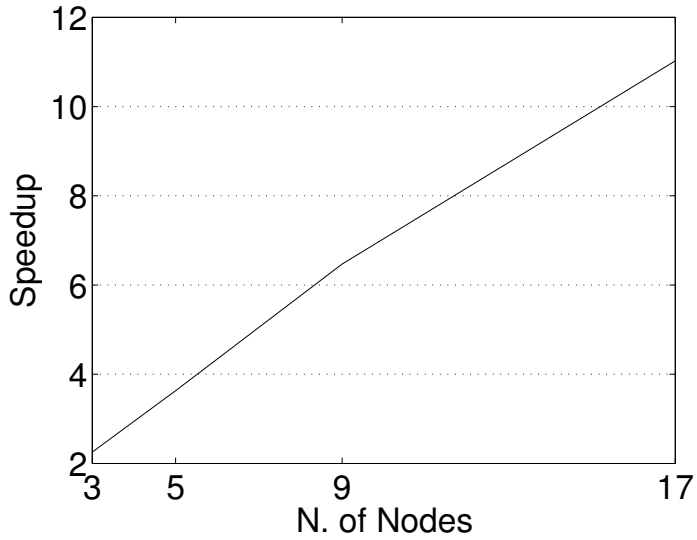
- Evolution of LEAP
- It adapts LEAP to perform EDGE^{MR}
- It exploits:
  - CELOE for learning the structure of a KB
  - EDGE^{MR} for learning the parameters of a probabilistic KB
- Same algorithm of LEAP, but it always uses EDGE^{MR} instead of EDGE
- The execution of CELOE is **not distributed**
- Future work
  - Distributing the scoring of the refinements generated during the execution of CELOE

Experiments

- LEAP^{MR} has been implemented in Java
- Cluster of 64-bit Linux machines with 8-cores Intel Haswell 2.40 GHz CPUs and 2 GB (max) Java memory allocated per node
- Dataset: Moral
  - 202 individuals and 4710 axioms (22 axioms are probabilistic)
- The experiments were performed on 1 (LEAP), 3, 5, 9 and 17 nodes

Results

- Speedup of LEAP^{MR} relative to LEAP for Moral KB:



- Speedup is significant but sublinear
- Sublinearity caused by MPI communication overhead

Thank you for listening!

Thank you for listening!