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

ILP 2015

Motivation

- DISPONTE semantics: “DIstribution Semantics for Probabilistic ONTologiEs”
- 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.
- DISPONTE applies the distribution semantics of probabilistic logic programming to description logics
- BUNDLE is a system for reasoning on DISPONTE KBs using Binary Decision Diagrams (BDDs)

DISPONTE

- Atomic choice: a pair (E_i, k), where E_i is the ith probabilistic axiom and $k \in \{0, 1\}$ indicates whether E_i is chosen to be included in a world ($K = 1$) or not ($K = 0$).
- Selection σ: set of one atomic choice for each probabilistic axiom.
- σ identifies a world w_σ
- $P(w_\sigma) = \prod_{(E_i,1)\in\sigma} p_i \prod_{(E_i,0)\in\sigma} (1 - p_i)$
- Probability of a query Q given a world w: $P(Q|w) = 1$ if $w \models Q$, 0 otherwise
- Probability of Q
  $P(Q) = \sum_w P(Q, w) = \sum_w P(Q|w)P(w) = \sum_{w:w\models Q} P(w)$

Inference and Query answering

- The probability of a query Q can be computed according to the distribution semantics by first finding the explanations for Q in the knowledge base
- Explanation: subset of axioms of the KB that is sufficient for entailing Q
- All the explanations for Q must be found, corresponding to all ways of proving Q
- Probability of $Q \rightarrow$ probability of the DNF formula

$$F(Q) = \bigvee_{e \in E_Q} \left(\bigwedge_{F_i \in e} X_i \right)$$

where E_Q is the set of explanations and X_i is a Boolean random variable associated to axiom F_i

- We exploit Binary Decision Diagrams for efficiently computing the probability of the DNF formula

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:

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

- $P(Q) = 0.4 \times 0.6 \times (1 - 0.3) + 0.3 \times 0.6 = 0.348$

BUNDLE

- Binary decision diagrams for Uncertain reasoNing on Description Logic thEories
- BUNDLE performs inference over DISPONTE knowledge bases.
- It exploits an underlying ontology reasoner able to return all explanations for a query, such as **Pellet** [Sirin et al, WS 2007]
- Explanations for a query in the form of *a set of sets of axioms*.
- Then DNF formula built and converted to BDDs for computing the probability

EDGE

- Em over bDds for description loGics paramEter learning
- EDGE is inspired to EMBLEM [Bellodi and Riguzzi, IDA 2013]
- Takes as input a DL theory and a number of examples that represent queries.
- The queries are concept assertions and are divided into:
  1. positive examples;
  2. negative examples.
- EDGE computes the explanations of each example using BUNDLE, that builds the corresponding BDD.
  - For negative examples, EDGE computes the explanations of the query, builds the BDD and then negates it.
- Then EM cycle.

EM cycle of EDGE

1. Expectation takes as input a list of BDDs and computes:
   - Expected counts of random variables: $E[c_{i0}|Q]$ and $E[c_{i1}|Q]$ for all axioms F_i, where c_{ix} is the number of times the binary variable X_i takes value x $\in \{0, 1\}$
   - Expected counts are computed by traversing twice the BDDs
   - $E[c_{i0}] = \sum_Q E[c_{i0}|Q]$ and $E[c_{i1}] = \sum_Q E[c_{i1}|Q]$
2. Maximization computes the parameters values for the next EM iteration by relative frequency.
   - parameters π_i represent $P(X_i) = 1$ for all axioms F_i
   - π_i = $E[c_{i1}]$ / ($E[c_{i0}]$ + $E[c_{i1}]$)

- The EM algorithm is guaranteed to find a local maximum of the probability of the examples.

EDGE^{MR}

- Distributed Parameter Learning by MapReduce
- Same approach as EMBLEM^{MR}, [Chu et al, NIPS 2006]: expectations are computed separately for each examples and then aggregated in the Reduce phase
- $n + 1$ workers from 0 to n. Worker 0 is the "master", the others the "slaves"
- The Map function is performed by all workers; the Reduce function by the master (the "reducer")
- The input KB T is replicated among all workers, the examples E are evenly divided among the $n + 1$ workers

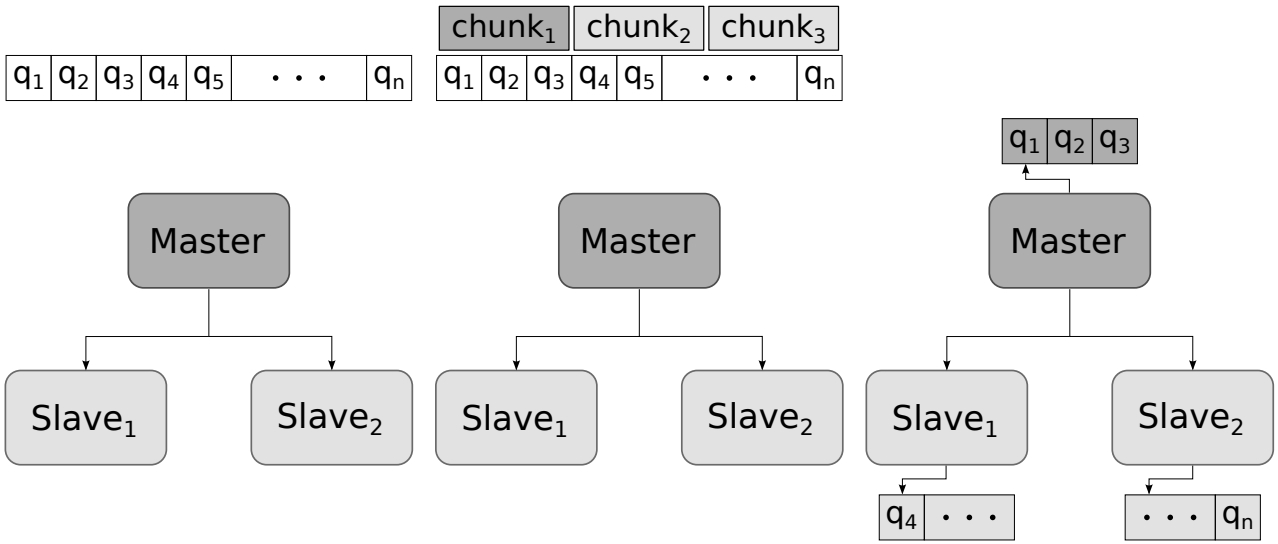
EDGE^{MR}

- Each worker builds the BDDs for its examples. All the mappers stay active keeping the BDDs in memory
- The Expectation step is executed in parallel by sending the current values of the parameters to each mapper *m*, which computes the expectations for each of its examples
- The vector of expectations are sent back to the master that aggregates by component-wise sum them and performs Maximization

Scheduling Techniques

Single-step scheduling:

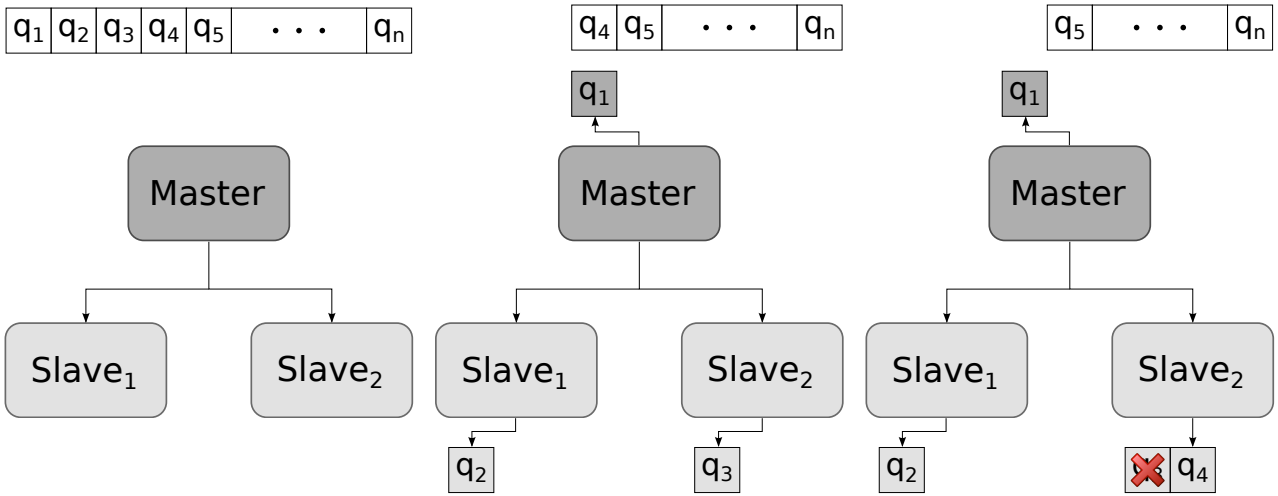
- n is the number of the slaves, the master divides the total number of queries into $n + 1$ chunks
- The master begins to compute its queries while, for each other chunk of queries, it send the chunk to the corresponding slave.
- Then the master waits for the results from the slaves. When the slowest 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
- At first each machine is assigned a query in order
- Then if the master ends handling a query it just takes the next one, instead, if a slave ends handling a query, it asks the master for a new query and the master sends the next query to the slave
- When all the queries are evaluated, EDGE^{MR} starts the EM cycle.



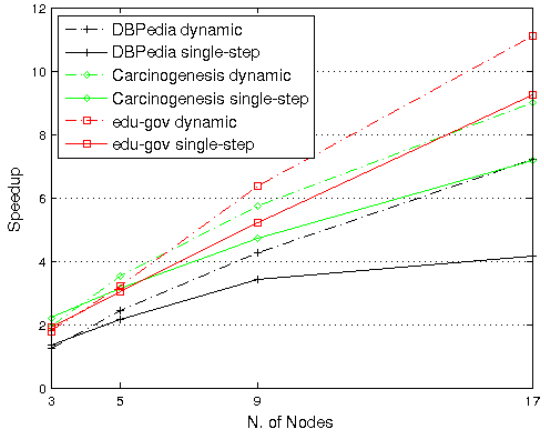
Experiments

- EDGE^{MR} has been implemented in Java using MPI
- Hardware: machines with an Intel Xeon Haswell E5-2630 v3 (2.40GHz) CPU with 2GB of memory allocated to the job.
- Datasets: Mutagenesis, Carcinogenesis, an extract of DBpedia and `education.data.gov.uk`
- Generation of positive and negative examples by sampling individuals and classes from the dataset
- five-fold cross-validation for each dataset and for each number of workers

Dataset	# of all axioms	# of probabilistic axioms	# of pos. examples	# of neg. examples	Fold size (MiB)
Carcinogenesis	74409	186	103	154	18.64
DBpedia	5380	1379	181	174	0.98
`education.data.gov.uk`	5467	217	961	966	1.03
Mutagenesis	48354	92	500	500	6.01

Experiments

Dataset	EDGE	EDGE^{MR}							
		Dynamic				Single-step			
		3	5	9	17	3	5	9	17
Carcinogenesis	847	441.8	241	147.2	94.2	384	268.4	179.2	117.8
DBpedia	1552	1259.8	634	364.6	215.2	1155.6	723.8	452.6	372.6
`education.data.gov.uk`	6924.2	3878.2	2157.2	1086	623.2	3611.6	2289.6	1331.6	749.4
Mutagenesis	1439.4	635.8	399.8	223.2	130.4	578.2	359.2	230	124.6



Speedup (ratio of the running time of 1 worker to the running time of n workers)

Conclusions and Future Work

- Conclusions
  - The speedup is significant even if it is sublinear
  - A certain amount of overhead (the resources, and thereby the time, spent for the MPI communications) is present.
  - Dynamic scheduling has generally better performance than single-step scheduling.
- Future work
  - use EDGE^{MR} for distributed structure learning in the LEAP system [Riguzzi et al, URSW 2014]
  - exploit Linked Open Data for completing the given data

THANKS FOR LISTENING AND ANY QUESTIONS ?