Finding the Best Probabilistic Schema for an XML Corpus

S. Abiteboul, Y. Amsterdamer, D. Deutch, T. Milo, P. Senellart

Webdam Seaminar, 4 March 2011

XML Schemas vs Probabilistic XML

XML Schema

- Compact description of a (possibly infinite) set of XML documents
- Nondeterministic generator

XML Schemas vs Probabilistic XML

XML Schema

- Compact description of a (possibly infinite) set of XML documents
- Nondeterministic generator

Probabilistic XML

- Compact description of a (possibly infinite) probability distribution over XML documents
- Probabilistic generator

XML Schemas vs Probabilistic XML

XML Schema

- Compact description of a (possibly infinite) set of XML documents
- Nondeterministic generator

Probabilistic XML

- Compact description of a (possibly infinite) probability distribution over XML documents
- Probabilistic generator

Question

Can we transform an XML schema into a probabilistic XML document by learning the optimal probabilities (w.r.t. a corpus)?

Same-Structure Inference Problem

Given:

- an XML corpus, i.e., multiset of documents $\{|d_1 \dots d_n|\}$
- an XML Schema schema, i.e., a top-down deterministic tree automaton with primary keys and foreign keys
- Some class of probabilistic distributions (e.g., Gaussian, uniform...) for data values

Find the best probabilistic XML generator, as a recursive Markov chain [Benedikt et al., 2010] (i.e., probabilistic tree automaton) extended with:

- Continuous probability distributions [Abiteboul et al., 2010]
- Long-distance constraints

... that has the same structure as the schema.

Motivating Applications

- Sampling of XML documents similar to a corpus: testing
- Analysis of a corpus, and display to a user
- Evaluating the respective quality of two XML schemas
- Concise summary of a corpus, on which statistics can be gathered (e.g., through aggregate queries)

Outline

Introduction

Basic Case

Adding Constraints

Conclusion

Basic Setting: No Constraints

- We have: a top-down tree automaton and a corpus
- We want: a probabilistic tree automaton with the same structure that maximizes the likelihood of the corpus

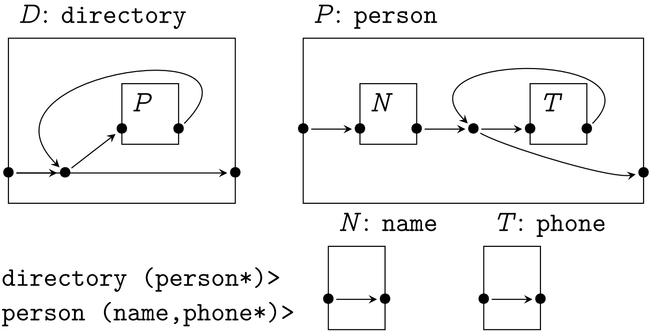
Basic Setting: No Constraints

- We have: a top-down tree automaton and a corpus
- We want: a probabilistic tree automaton with the same structure that maximizes the likelihood of the corpus

```
<!ELEMENT directory (person*)>
<!ELEMENT person (name,phone*)>
```

Basic Setting: No Constraints

- We have: a top-down tree automaton and a corpus
- We want: a probabilistic tree automaton with the same structure that maximizes the likelihood of the corpus



```
<!ELEMENT directory (person*)>
<!ELEMENT person (name,phone*)>
```

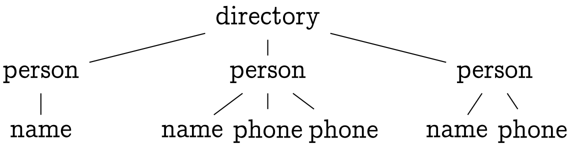
Algorithm

Very simple algorithm:

1. For each document of the corpus, run the automaton it, and for each state encountered:
   1.1 Increment a counter for the state
   1.2 Increment a counter for the outgoing transition
2. Normalize each transition counters by the counter of the incoming state: this gives a transition probability

Complexity linear in the size of the automaton and the corpus.

Example Run

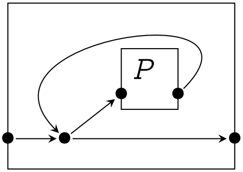


directory

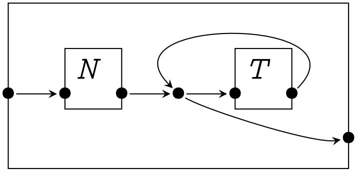
person person person

name name phone phone name phone

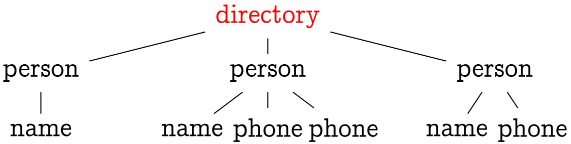
D: directory

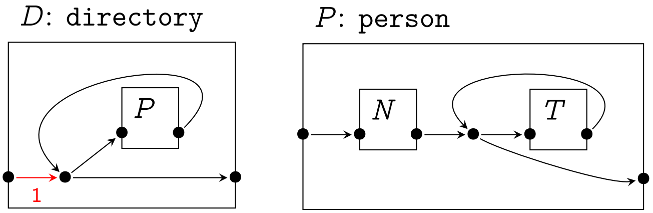


P: person

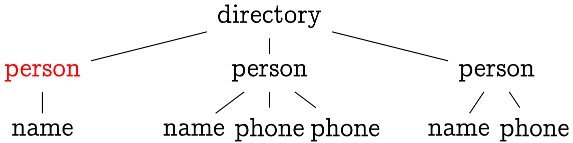


Example Run


directory
person
person
person
name
name
phone
phone
name
phone


D: directory
P
1
P: person
N
T

Example Run



directory

person

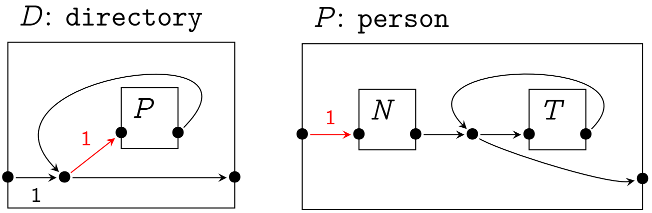
person

person

name

name phone phone

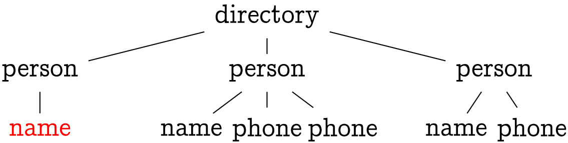
name phone



D: directory

P: person

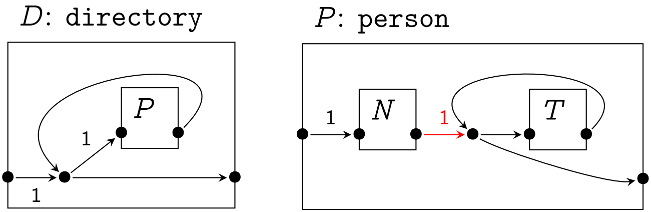
Example Run



directory

person person person

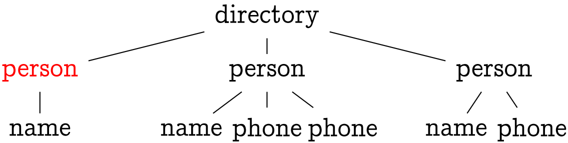
name name phone phone name phone



D: directory

P: person

Example Run

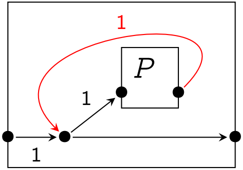


directory

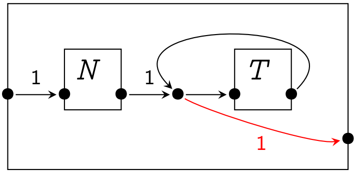
person person person

name name phone phone name phone

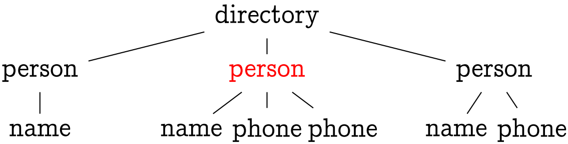
D: directory



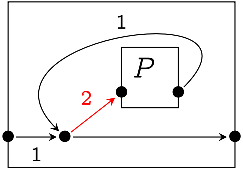
P: person



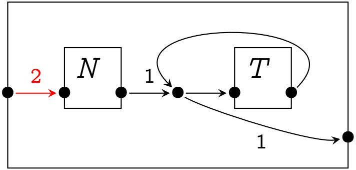
Example Run



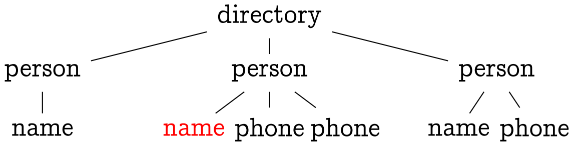
D: directory

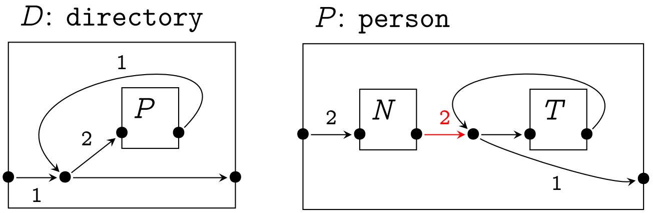


P: person



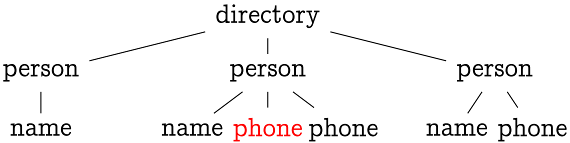
Example Run


directory
person
person
person
name
name phone phone
name phone


D: directory

P: person

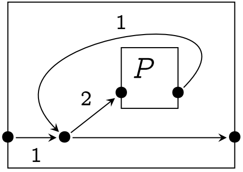
Example Run



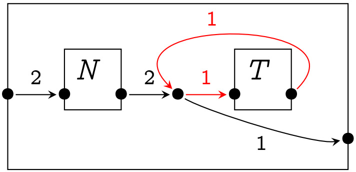
directory

person person person

name name phone phone name phone

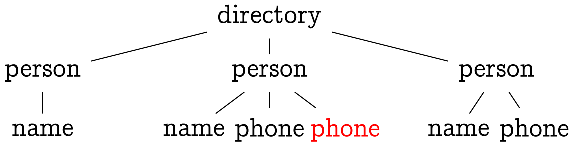


D: directory



P: person

Example Run

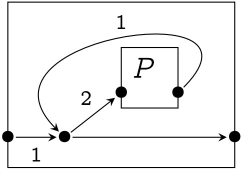


directory

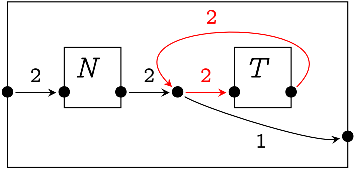
person person person

name name phone phone name phone

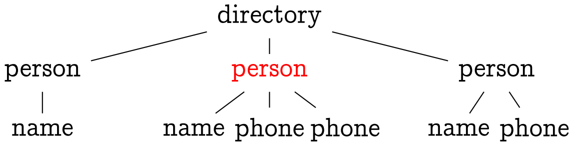
D: directory



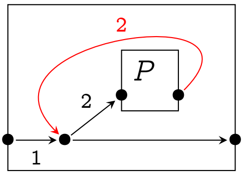
P: person



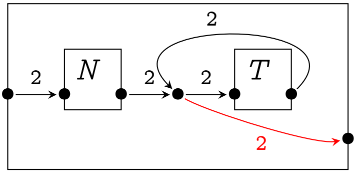
Example Run



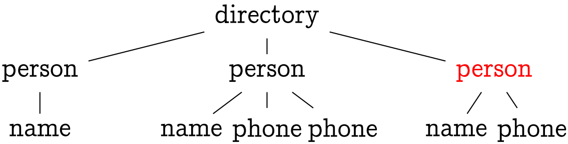
D: directory



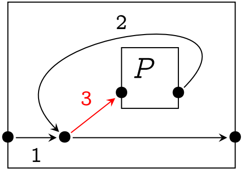
P: person



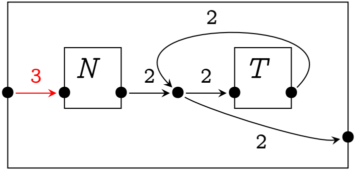
Example Run



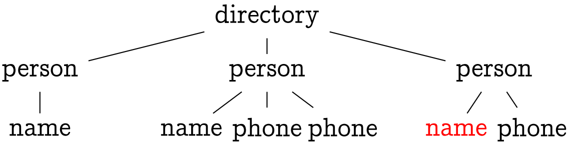
D: directory



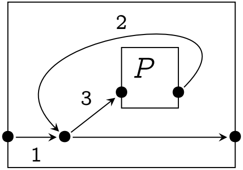
P: person



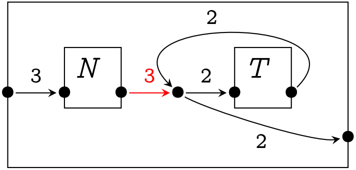
Example Run



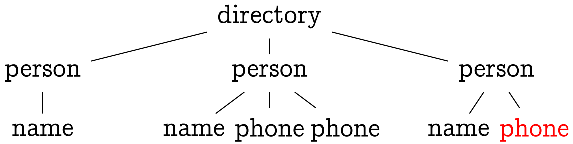
D: directory

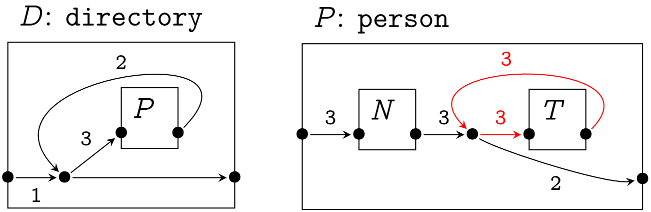


P: person

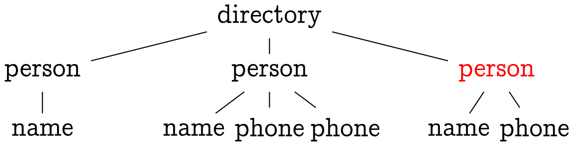


Example Run

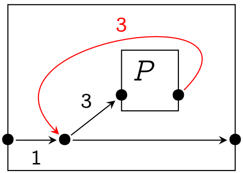

directory
person
person
person
name
name
phone
phone
name
phone


D: directory
2
P
3
1
P: person
3
N
3
3
T
3
2

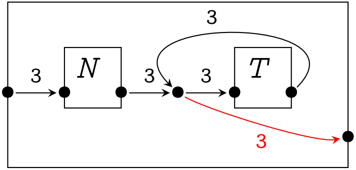
Example Run



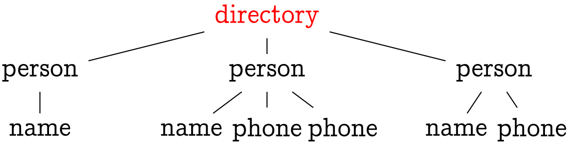
D: directory

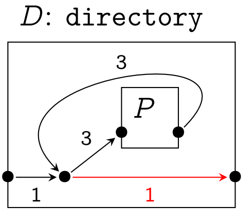


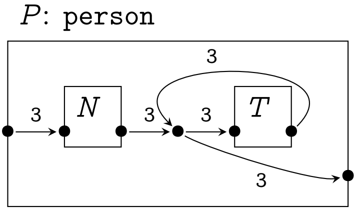
P: person



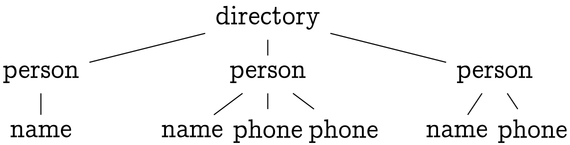
Example Run


directory
person
person
person
name
name
phone
phone
name
phone

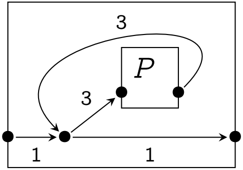

D: directory
P
3
3
1
1


P: person
N
T
3
3
3
3
3

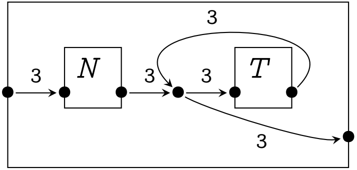
Example Run



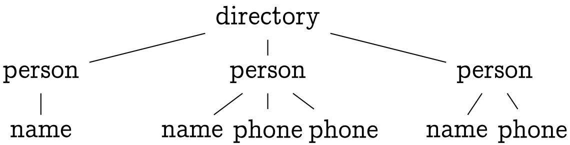
D: directory



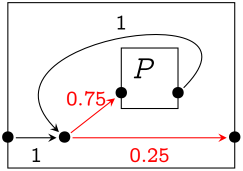
P: person



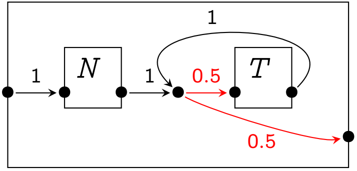
Example Run


directory
person
person
person
name
name phone phone
name phone

D: directory



P: person



Correctness of The Algorithm

Proposition

The probabilities assigned by the algorithm optimizes the likelihood of the corpus.

Proposition

The probabilistic generator constructed by the algorithms terminates with probability 1.

Correctness of The Algorithm

Proposition

The probabilities assigned by the algorithm optimizes the likelihood of the corpus.

Proposition

The probabilistic generator constructed by the algorithms terminates with probability 1.

Actually, results (kind of) known in the literature about probabilistic context-free grammars.

Outline

Introduction

Basic Case

Adding Constraints

Conclusion

Constraints

In XML Schema:

- xs:ID: global unary primary keys, only one kind per document
- xs:IDREF: global unary foreign keys, only one kind per document
- <xs:key>, <xs:unique>: primary keys, possibly local to a subtree, non-necessary unary
- <xs:keyref>: foreign keys, possibly local to a subtree, non-necessary unary

We consider for now:

- Global unary primary keys and foreign keys (extension to local should be possible)
- Domains of finite size: together with primary keys, act as a constraint!

Semilinear Sets

$$(A_1 \dots A_k) \in \bigcup_{i=1}^{p} \{v_{i0} + \alpha_1 v_{i1} + \cdots + \alpha_{n_1} v_{in_i} \mid \alpha_1 \dots \alpha_{n_1} \in \mathbb{N}\}$$

Interesting properties:

- The constraints on the number of times a tree automaton enters each particular state are given by a semilinear set (Parikh's theorem)
- Key constraints can be represented by semilinear sets
- Testing if the intersection of two semilinear sets is empty can be done in NP (integer programming)

How to Adapt the Algorithm

- For each transition, first check if the current document + the automaton + the constraints has a possible continuation (test the intersection of the semilinear sets)
- If only one transition from a given state, do not increment the counter of the
- Otherwise, proceed as usual
- Optimality results (for this class of probabilistic generator with continuation tests) still hold!

Remark

Only works for binary choices; but possible to transform n-ary choices into binary ones.

Outline

Introduction

Basic Case

Adding Constraints

Conclusion

In brief

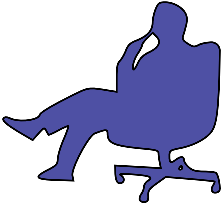
- Work in progress
- Optimal probabilistic generator with respect to an XML schema and corpus
- Effective algorithm to compute it!
- Nondeterministic polynomial time: maybe not so bad, especially if we assume constraints are small

What Remains to Be Done

- Check everything really works ☺
- Adding data value generators: not obvious which kind of dependencies among data values should be preserved
- Detailed complexity analysis
- Implementation

What Remains to Be Done

- Check everything really works ☺
- Adding data value generators: not obvious which kind of dependencies among data values should be preserved
- Detailed complexity analysis
- Implementation



- More general constraint language? Things will become undecidable pretty fast.
- What about probabilistic constraints?
- What if the tree automaton is non-deterministic?
- What if we also want to discover the schema?
- Compelling application?

Merci.

Webdam

Serge Abiteboul, T-H. Hubert Chan, Evgeny Kharlamov, Werner Nutt, and Pierre Senellart. Aggregate queries for discrete and continuous probabilistic xml. In *Proc. ICDT*, Lausanne, Switzerland, March 2010.

Michael Benedikt, Evgeny Kharlamov, Dan Olteanu, and Pierre Senellart. Probabilistic XML via Markov chains. *Proceedings of the VLDB Endowment*, 3(1):770–781, September 2010. Presented at the VLDB 2010 conference, Singapore.