Learning Twig and Path Queries

Sławek Staworko\textsuperscript{1} \quad Piotr Wieczorek\textsuperscript{2}

\textsuperscript{1}Mostrare Project
INRIA Lille – Nord Europe
University of Lille

\textsuperscript{2}Institute of Computer Science
University of Wrocław

March 27, 2012
ICDT 2012
Motivation

The challenge

- XML is text based for easy and direct access to its contents
- XPath and XQuery might be too formal/difficult for unknowledgeable users

XML query inference

- User provides an XML document with annotations (selected nodes, etc.)
- A learning algorithm returns a query addressing the user’s needs

Learning: a popular and interesting topic

- Languages of words [Gold '67], Word patterns [Anguin '79], Regular languages [Anguin '87, Oncina and Garcia '94, de la Higuera '97, ...]
- Schemas for XML [Bex et al. '10, ...]
- Query Automata [Carme et al. '07] and tree transducers [Lemay et al. '10]
- XQuery [Morishima et al. '04] and web extraction patterns [Raeymaekers et al. '08]
Overview

1. Preliminaries

2. What learning queries means?

3. Negative results

4. Learning algorithms
Basic notions
XML, Examples, and Unary Twig queries (tree patterns)
XML, Examples, and Unary Twig queries (tree patterns)

- **Positive examples (required nodes):**
  - library
  - book
  - title
  - author
  - Grundrisse
  - K. Marx

- **Negative examples (forbidden nodes):**
  - library
  - book
  - title
  - author
  - Capital
  - K. Marx

- **Positive examples (required nodes):**
  - library
  - collection
  - title
  - author
  - Manifesto
  - K. Marx

- **Negative examples (forbidden nodes):**
  - library
  - book
  - title
  - author
  - F. Engels
  - Conditions
XML, Examples, and Unary Twig queries (tree patterns)

/library/*[author = 'K. Marx']/title
XML, Examples, and Unary Twig queries (tree patterns)
Boolean queries

Positive and Negative Examples:

+  
  offer  
  item  
  type  descr
  For sale  Audi A4
  Wanted  MacBook

-  
  offer  
  item  
  type  descr
  Wanted  3D Puzzle
  For sale  Eee PC

Output Query:

offer  
item  
type  For sale
Boolean queries

Positive and Negative Examples:

```
+ offer
  | item
  |   type descr
  |   For sale
  |   Audi A4

- offer
  | item
  |   type descr
  |   Wanted
  |   MacBook

+ offer
  | list
  |   item
  |   type descr
  |   For sale
  |   3D Puzzle

+ offer
  | list
  |   item
  |   type descr
  |   For sale
  |   Eee PC
```

Output Query:

```
offer
  | list
  |   item
  |   type descr
  |   For sale
  |   Eee PC
```

(Path query)
What is learnability of queries?
Learning model

When is a class of queries $Q$ learnable from examples?

There is an algorithm learner that takes a set of examples and returns a query in $Q$.

1. learner works in polynomial time.
2. learner is sound i.e., it returns a query consistent with the set of examples (a query that selects all positive examples and none of negative examples);
   If no such query exists, then learner returns a null value.
Learning model

When is a class of queries \( Q \) learnable from examples?

There is an algorithm \textit{learner} that takes a set of examples and returns a query in \( Q \).

1. \textit{learner} works in polynomial time.

2. \textit{learner} is sound i.e., it returns a query consistent with the set of examples (a query that selects all positive examples and none of negative examples);
   If no such query exists, then \textit{learner} returns a null value.

\textbf{Bad news: Intractability of the consistency problem}

Deciding whether there exists a Twig query consistent with given a set of positive and negative examples, is \textit{NP-complete}.
Dealing with the bad news

1. Approximate learning
The result query may select some negative examples and omit some of the positive ones.

2. Richer query classes
Consistency for unions of Twig queries is in PTIME.

3. Use positive examples only
In the presence of positive examples only, consistency for Twig queries is trivial (the universal query selects all nodes).
Dealing with the bad news

1. Approximate learning
   The result query may select some negative examples and omit some of the positive ones.

2. Richer query classes
   Consistency for unions of Twig queries is in PTIME.

3. Use positive examples only
   In the presence of positive examples only, consistency for Twig queries is trivial (the universal query selects all nodes).
Learning from positive examples only

When is a class of XML queries learnable from positive examples?
There is an algorithm learner that takes a set of positive examples and returns a query.

1. learner works in polynomial time.
2. learner is sound i.e., it returns a query consistent with the set of examples.

Is that sufficient?
No. A trivial algorithm always returning the universal query \( //\) satisfies 1 and 2.
Minimality

What if we would require learner to return a minimal query (w.r.t. containment) that is consistent with the input set of examples?

Incompatible with polynomiality of learner
A set of examples may have a minimal Twig query of exponential size.
Completeness

Identification in the limit (Gold '67)
A good learning algorithm should be able to infer any concept with a sufficiently rich set of examples.

When is a class of XML queries learnable from positive examples?
There is an algorithm learner that takes a set of positive examples \( S \) and returns a query.

1. learner works in polynomial time.
2. learner is sound i.e., it returns a query consistent with the set of examples.
3. learner is complete: for every query \( q \in Q \) there is a (polynomially sized) set of examples \( CS_q \) for which learner returns \( q \). Furthermore, \( CS_q \) is robust under inclusion of non-essential examples.

Often, \( CS_q \) is called the characteristic sample for \( q \) w.r.t. learner.
Learning Algorithms
Learning unary path queries

Examples:

```
   r
  /\  
 c  a
 / \ /  
 a  a  a
 / \ /   
 b  b  a
 / \ /    
 a  c  c
 /     /  
 a     b
```

Query inference:

```
Invariant: constructed query is consistent with the examples
```
Learning unary path queries

Examples:

Query inference:

Invariant: constructed query is consistent with the examples

Find the smallest selecting path $w_0$
Learning unary path queries

Examples:

Query inference:

Invariant: constructed query is consistent with the examples

Find the smallest selecting path $w_0$

1. Start with the universal query

2.
Learning unary path queries

Examples:

```
   r
 /   \
 c  a
|   |
a  b
|   |
a  c
|   |
a
```

Query inference:

```
   *  
 /   \
 *  a
|   |
*  b
|   |
*  
```

Invariant: constructed query is consistent with the examples

Find the smallest selecting path $w_0$

1. Start with the universal query

2. Saturate with common factors (found in $w_0$)

Sławek S. (Mostrare, INRIA Lille) Learning Twigs
Learning unary path queries

Examples:

```
  r
 /|
/ |\
 c a
/ |\
 a a
/ |\
 b b
/ |\
 a c
/ |\
 c b
/ |
 a
```

Query inference:

```
  *  
/ |
/ *  
/ a  
/ |\
/ b  
/ |\
/ *  
/ c  
```

Invariant: constructed query is consistent with the examples

1. Find the smallest selecting path w₀
2. Start with the universal query
3. Saturate with common factors (found in w₀)
Learning unary path queries

Examples:

```
r
  / \
 c   a
  |   |
 a   b
  |   |
 b   c
  |   |
 a   b
  |   |
 a   c
  |   |
 a
```

Query inference:

```
*     *     *     *
\|     \|     \|     \|
*     *     *     *
 a     a     a     a
\|     \|     \|     \|
 b     b     b     b
\|     \|     \|     \|
 c     c     c     c
```

Invariant: constructed query is consistent with the examples

Find the smallest selecting path $w_0$  
Start with the universal query  
Saturate with common factors (found in $w_0$)  
Specialize //-edges
Learning unary path queries

Examples:

\[
\begin{array}{c}
    r \\
    \downarrow \\
    c \\
    \downarrow \\
    a \\
    \downarrow \\
    a \\
    \downarrow \\
    b \\
    \downarrow \\
    a \\
    \downarrow \\
    c \\
    \downarrow \\
    b \\
    \downarrow \\
    a \\
\end{array}
\]

Query inference:

\[
\begin{array}{c}
    * \\
    \mid \\
    * \\
    \mid \\
    a \\
    \mid \\
    b \\
    \mid \\
    c \\
    \mid \\
    * \\
\end{array}
\]

Invariant: constructed query is consistent with the examples

1. Find the smallest selecting path \( w_0 \)
2. Start with the universal query
3. Saturate with common factors (found in \( w_0 \))
4. Specialize //edges
Learning unary path queries

Examples:

```
  r
   \   \   \
  c    a
 |     |
a     a
 |     |
b     b
 |     |
a     c
 |     |
c     b
 |     |
a
```

Query inference:

```
  *    *    *    *    *
   |    |    |    |    |
  *    *    *    *    *
   |    |    |    |    |
  a    a    *    *    *
   |    |    |    |    |
  b    b    a    a    *
   |    |    |    |    |
  c    c    b    b    *
   |    |    |    |    |
  c    c    c    c    *
```

Invariant: constructed query is consistent with the examples

Find the smallest selecting path $w_0$

Start with the universal query

Saturate with common factors (found in $w_0$)

Specialize $\supset\supset$-edges

Specialize root and leaf
Learning Boolean path queries
What is difficult about learning Boolean (path) queries?

Problem: Every tree offers several paths to choose from
Solution: Use all of them and infer sets (conjunctions) of queries
Learning conjunctions of Boolean path queries

Invariant: constructed query is satisfied in all trees

Initialize $w_0$

Start with the universal query

Saturate with common factors

Specialize $//\text{-edges}$

Specialize root and leaf

1. offer
   | item
   \- for-sale descr

2. offer
   | list
   | item
   \- for-sale descr

3. offer
   | list
   | item
   \- for-sale descr

4. offer
   | list
   | item
   \- for-sale descr

5. offer
   | list
   | item
   \- for-sale descr
Learning conjunctions of Boolean path queries

$w_0$

<table>
<thead>
<tr>
<th>Original Path Query</th>
<th>Inferred Path Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>offer/item/for-sale</td>
<td>offer/item/for-sale</td>
</tr>
<tr>
<td>offer/item/descr</td>
<td>offer/item/descr</td>
</tr>
<tr>
<td>offer/list/item/for-sale</td>
<td>offer/item/for-sale</td>
</tr>
<tr>
<td>offer/list/item/descr</td>
<td>offer/item/descr</td>
</tr>
<tr>
<td>offer/list/item/wanted</td>
<td>offer/item/*</td>
</tr>
</tbody>
</table>

Obtained conjunction of (minimal) path queries

offer/item/for-sale                      offer/item/descr
Learning Boolean Twig Queries
Weaving a Twig query with path queries

Fusing a path query $p$ into Twig query $f$

1. split of $p$ into some $p_1 \cdot p_2$
2. embed $p_1$ into $f$ and let $p_1$ end at $n$
3. attach $p_2$ at $n$

\[\begin{array}{c}
\text{Fusion} \\
\text{a} & * & a \\
\text{b} & a & c \\
\end{array}\rightarrow
\begin{array}{c}
\text{a} & a \\
\text{b} & a & c \\
\end{array}\]
Learning Boolean Twig queries

Learning a conjunction of Boolean path queries yields:

\[ \text{dblp}$\star$/$\text{author}$, \text{dblp}$\star$/$\text{title}$, \text{dblp}$\star$/$\text{url}$\]

Iteratively fuse these queries, at each step taking a minimal consistent query.
Learning Unary Twig Queries
Learning unary Twig queries
Learning unary Twig queries
Learning unary Twig queries
Learning unary Twig queries
Learning unary Twig queries
Learning unary Twig queries (cont’d.)

```
library
  collection
    title
      capital
    author
      marx
  book
    title
      manifesto
    author
      marx
      engels
```
Learning unary Twig queries (cont’d.)
Learning unary Twig queries (cont’d.)
Learning unary Twig queries (cont’d.)

```
library
  collection
    title
    
    capital
    
    author
    marx

book
  title
  
  author
  marx

manifesto
  
  author
  engels
```

```
library
  *
  
  title

library
  *
  
  title

library
  *
  
  title

  author
  marx
```
Conclusions and what’s in the paper

Learnability (in the limit) of Twig queries

- Unfeasible in the presence of positive and negative examples
- Feasible in the presence of positive examples only

Anchored Twig queries

- //edge cannot be incident to * unless it is the root a leaf
- * may be a non-selecting leaf node only if it is incident to //edge

Two essential properties of Anchored Twig queries that enable our algorithms

P1 Containment of queries can be tested with embeddings (and thus is in PTIME)
P2 Existence of polynomially sized match sets (for completeness) [Miklau, Suciu, 2004]