Similarity of DTDs Based on Edit Distance and Semantics

Ales Wojnar, Irena Mlynkova, Jiri Dokulil

ales.wojnar@gmail.com, irena.mlynkova@mff.cuni.cz, jiri.dokulil@mff.cuni.cz

Charles University
Faculty of Mathematics and Physics
Department of Software Engineering
Prague, Czech Republic

Introduction

- XML = a standard for data representation and manipulation

  ⇒ used in most areas of IT
- Possible optimization: exploitation of similarity of XML data
  - Structural/semantic similarity
  - Typical applications: clustering, dissemination-based applications, schema integration systems, data warehousing, e-commerce, semantic query processing, …

    ⇒ Amount of approaches to similarity evaluation is high
  - Problem: persisting open issues to be solved

Goals of the Paper

- Our focus: similarity of XML schemas
  - XML documents = trees
  - XML schemas = regular expressions
    - More complex problem
- Disadvantage to be solved:
  - Emphasis on semantic similarity
  - Structural similarity is analyzed trivially
    - Comparison of leaf nodes / direct child nodes
- Our aims:
  - Emphasis on structural similarity
  - Preservation of exploitation of semantic similarity

Motivation

- Structural similarity of XML documents = minimum tree edit distance
  - XML documents D_A and D_B = labelled trees T_A and T_B
  - Number of operations to transform T_A to T_B
- Basic tree edit operations: Relabeling, InsertNode, DeleteNode
  - XML data: sharing, repetitions, recursion, …
  - ⇒ XML documents with the same DTD can have different structure
- XML tree edit operations: InsertTree, DeleteTree
- Problem:
  - XML schema = general graph (cycles, shared fragments)
  - How to incorporate semantics of element/attribute names?

Main Body of Algorithm

Input: DTD_A, DTD_B

Output: Edit distance between DTD_A and DTD_B

$T_A = \text{ParseXSD}(DTD_A)$;

$T_B = \text{ParseXSD}(DTD_B)$;

$Cost_{Graft} = \text{ComputeCost}(T_B)$;

$Cost_{Prune} = \text{ComputeCost}(T_A)$;

return $\text{EditDistance}(T_A, T_B, Cost_{Graft}, Cost_{Prune})$;

- **Classical tree edit approach**
  1. **DTDs are parsed + their trees are constructed**
  2. **Costs for inserting/deleting subtrees are computed**
  3. **Resulting minimal edit distance is evaluated**
     - **Dynamic programming**

DTD Tree Construction (1)

- DTD content models can be complex
  - Arbitrary combinations of operators (| , ( )) and cardinality constraints (? * +)
- Simplification rules:

$$\begin{array}{l} \text{I-a)}\ (e_1|e_2)^* \rightarrow e_1^*, e_2^* \\ \text{I-b)}\ (e_1, e_2)^* \rightarrow e_1^*, e_2^* \\ \text{I-c)}\ (e_1, e_2)? \rightarrow e_1?, e_2? \\ \text{I-d)}\ (e_1, e_2)^+ \rightarrow e_1^+, e_2^+ \\ \text{I-e)}\ (e_1|e_2) \rightarrow e_1?, e_2? \end{array}$$

$$\begin{array}{ll} \text{II-a)}\ e_1^{++} \rightarrow e_1^+ & \text{II-b)}\ e_1^{**} \rightarrow e_1^* \\ \text{II-c)}\ e_1^*? \rightarrow e_1^* & \text{II-d)}\ e_1?^* \rightarrow e_1^* \\ \text{II-e)}\ e_1^{+*} \rightarrow e_1^* & \text{II-f)}\ e_1^{*+} \rightarrow e_1^* \\ \text{II-g)}\ e_1?^+ \rightarrow e_1^* & \text{II-h)}\ e_1^+? \rightarrow e_1^* \\ \text{II-i)}\ e_1?? \rightarrow e_1^? & \end{array}$$

- Cardinality constraints are connected to single elements, no usage of | operator
  - A slight information loss

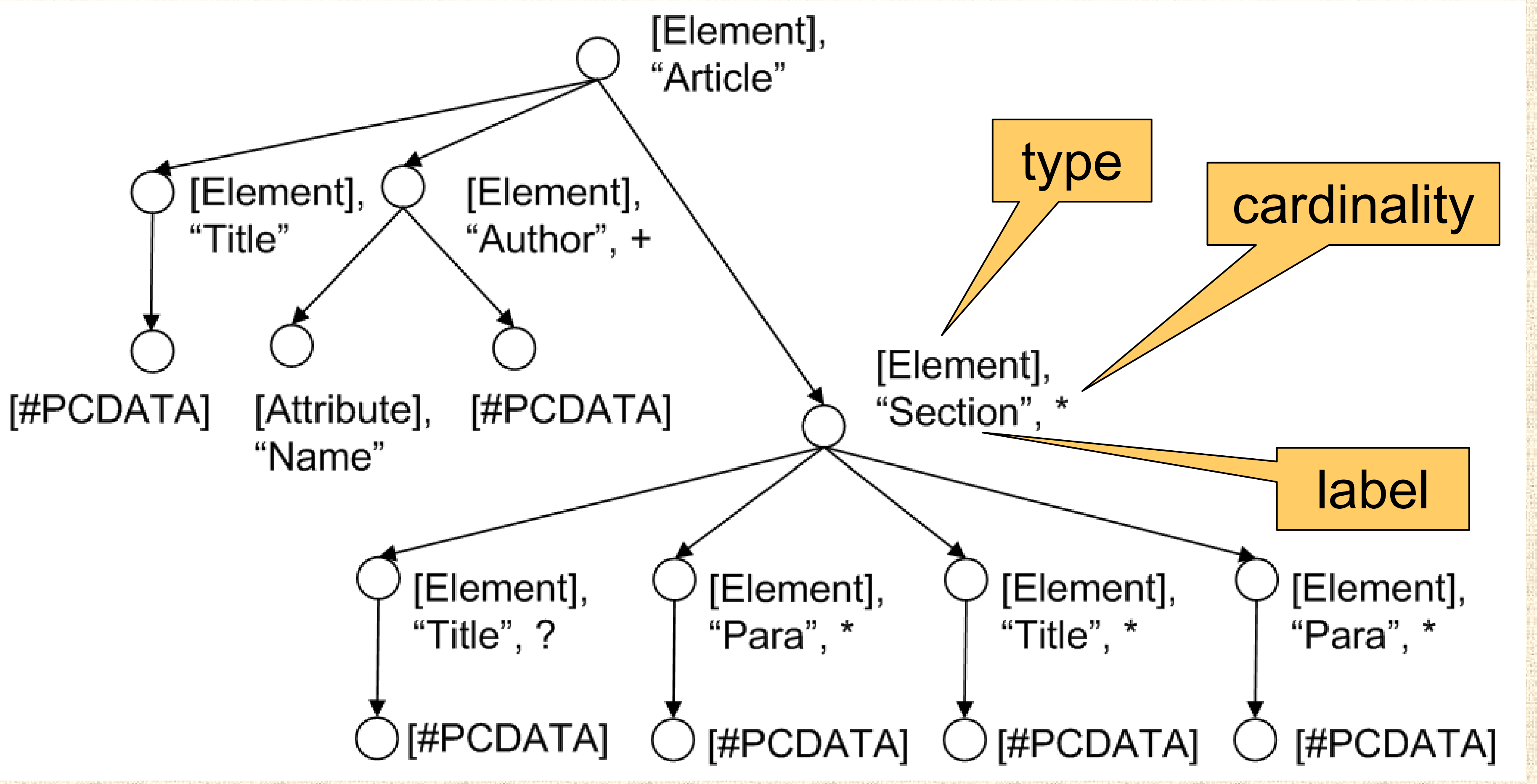
DTD Tree Construction (2)

- DTD = general directed graph
- Shared elements
  - Undirected cycles
  - Solution: Creating of a separate copy of shared fragment for each sharer
- Repeatable elements
  - Directed cycles
    - The same approach would lead to infinitely deep trees
  - Solution: Statistical analyses of XML data (“the depth of real-world XML data is < 10 on average”)

  $\Rightarrow$ ∞ can be modelled with a constant

Example:

<!ELEMENT Article (Title, Author+, Section+)>
<!ELEMENT Section (Title?, (Para|(Title?, Para+)+)*)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Para (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ATTLIST Author CDATA Name REQUIRED>



Tree Edit Operations

- Same as for XML trees: Relabeling, InsertNode, DeleteNode, InsertTree, DeleteTree
- Transformation of T_A to T_B: various sequences of operations
- Optimization: allowable sequences

relaxed

  - Tree T may be inserted only if tree similar to T occurs in T_B
  - Tree T may be deleted only if tree similar to T occurs in T_A
  - Tree that has been inserted via the InsertTree may not subsequently have additional nodes inserted
  - Tree that has been deleted via the DeleteTree may not previously have had nodes deleted

Similarity of Element/Attribute Names

	*	+	?	none
*	1	0.9	0.7	0.7
+	0.9	1	0.7	0.7
?	0.7	0.7	1	0.8
none	0.7	0.7	0.8	1

$$\mathrm{Sim}(e_1, e_2) = \mathrm{Max}(\mathrm{SemanticSim}(e_1, e_2), \mathrm{SyntacticSim}(e_1, e_2)) \times \alpha + \mathrm{CardinalitySim}(e_1, e_2) \times \beta$$

- $\alpha + \beta = 1$ and $\alpha, \beta \geq 0$
- **SemanticSim: distance of labels of e_1, e_2 in thesaurus**
- **SyntacticSim: edit distance of labels of e_1, e_2**
- **CardinalitySim: cardinality compatibility table**

Cost of Tree Edit Operations

- **Inserting/deleting tree *T*:**
  - **Single InsertTree/DeleteTree … a combination of InsertTree/DeleteTree and Insert/Delete**
  - **Which is the best?**
- **Idea:**
  - **Pre-computed: $Cost_{Graft}(T)$, $Cost_{Prune}(T)$ for each subtree T**
  - **Dynamic programming: finds the optimal sequence of edit operations**
- **Classical approach for tree edit distance**
  - **See the paper for details…**

Complexity

- **Classical edit distance: $O(|T_A||T_B|)$**
- **Construction of DTD tree T_A: $O(|T_A|)$**
- **SyntacticSim: $O(|T_A||T_B||\Omega|)$**
  - **Evaluated for each pair of element/attribute names**
  - **Ω = maximum length of element/attribute label**
- **CardinalitySim: $O(|T_A||T_B|)$**
- **SemanticSim: $O(|T_A||T_B||\Sigma|)$**
  - **Σ = size of the thesaurus**

$\Rightarrow O(|T_A||T_B||\Sigma|)$

Experiments (1): Real-World Data

	c1	c2	c3	c4	c5	tv	np
c1	1	0.57	0.43	0.19	0.71	0.08	0.42
c2	0.57	1	0.57	0.45	0.48	0.10	0.11
c3	0.43	0.57	1	0.39	0.36	0.01	0.13
c4	0.19	0.45	0.39	1	0.21	0.00	0.00
c5	0.71	0.48	0.36	0.21	1	0.00	0.11
tv	0.08	0.10	0.01	0.00	0.00	1	0.00
np	0.42	0.11	0.13	0.00	0.11	0.00	1

c1, … c5 = customer
tv = TV schedule
np = newspaper

- **Expectable results:**
  - **Customers have higher similarity (0.44 on average) than distinct objects**
  - **c1 and np are structurally similar $\Rightarrow$ have higher similarity**
- **If we set α = 0 (switch off the semantic evaluation) the values are less precise**
  - **The trend between same and distinct objects is the same**
- **Not surprising – real-world data are simple**

Experiments (2): Semantic Similarity

- Motivation: Synthetic data ⇒ better demonstration of results
- Testing set: 3 DTDs with the same structure (PERSON, USER, AAA)
  - PERSON and USER have similar meaning of element/attribute names
  - AAA has no meaning of element/attribute names

Semantic similarity	✓	×
PERSON x USER	0.92	0.40
PERSON x AAA	0.33	0.33

⇒ More precise results

- At the cost of searching the thesaurus

Experiments (3): Edit Operations

- Question: Are InsertTree, DeleteTree useful also for DTDs?
- Testing set: 2 similar DTDs with shared fragments

Cost	1	5	10	100
USER1 x USER2	0.92	0.74	0.52	0.52

⇒ The DTDs were correctly identified as similar only when the costs were set sufficiently low, i.e. the operations were used

Conclusion

- Algorithm for evaluating XML schema similarity
  - Emphasis on structural level
  - Exploitation of semantics
- Combination of edit distance and semantic similarity
- Experiments:
  - Edit distance: Describes the structure more precisely
  - Semantic similarity: More precise results
    - At the cost of searching a thesaurus
- Future work:
  - Other edit operations
    - Moving a node or adding/deleting a non-leaf node
  - XML Schema constructs and “syntactic sugar”

Thank you