XML-Diff Algorithms

Daniel Hottinger Franziska Meyer

Department of Computer Science
Swiss Federal Institute of Technology Zurich

Summer Semester, 2005

Outline

1. Motivation
   - Detecting Changes
   - Goals
2. Implementation In Java
   - Fast Match / Edit Script
3. Evaluation Of XML-Diff Algorithms
   - Criteria
   - Algorithm Ideas
   - Detection Of Moves

Detecting Changes
Unstructured Text

Problem solved:

- Line-by-line diff using the Longest Common Sub-sequence (LCS) algorithm
- Edit script (qed commands, unified diff, . . .)
- GNU Diffutils
- SCM systems such as: CVS, Subversion, git, Bitkeeper, . . .

```
3,5c3,5
< <person >
< <firstname>Fransizka</firstname>
< <lasname>Meyer</lasname>
--
> <person female="true">
> <firstname>Franziska</firstname>
> <lastname>Meyer</lastname>
8c8
< <firstname def="ghi">Daniel</firstname>
--
> <firstname>Daniel</firstname>
12,13c12,13
< <firstname abc="def">Paul</firstname>
< <lastname>Sevinç</lastname>
--
> <firstname>Paul</firstname>
> <lastname abc="def">Sevinç</lastname>
```

Detecting Changes

Why XML Is Different

- Defaults
- Attributes
- Canonical XML
- Moves
- Tags as Unit
- Encoding, CDATA, . . .

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<people>
 <person >
  <firstname>Fransizka</firstname>
  <lasname>Meyer</lasname>
 </person>
 <person>
  <firstname def="ghi">Daniel</firstname>
  <lastname>Hottinger</lastname>
 </person>
 <person>
  <firstname abc="def">Paul</firstname>
  <lastname>Sevinç</lastname>
 </person>
</people>
```

Detecting Changes

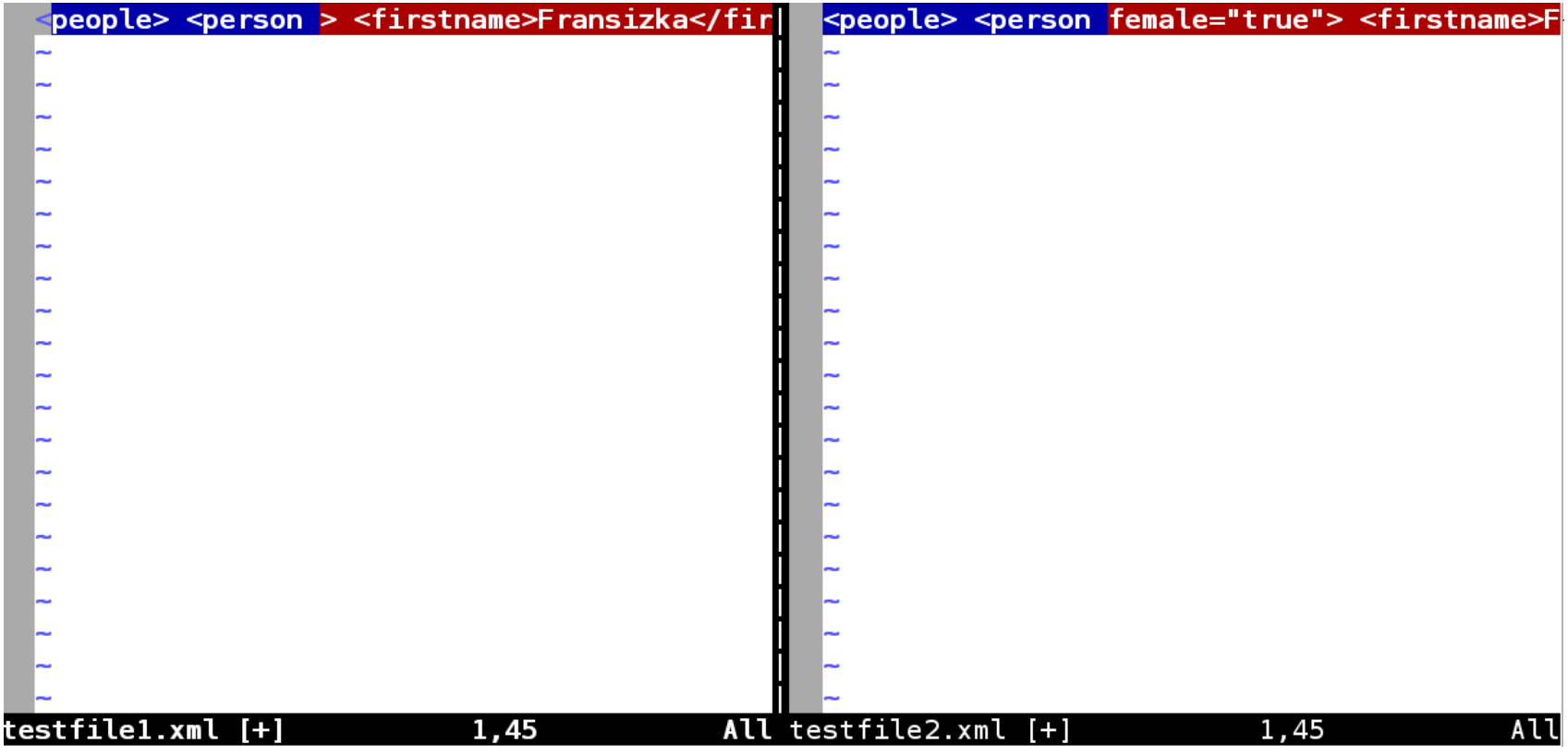
Text-based Tools Will Not Work

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<people>
 <person >
  <firstname>Fransizka</firstname>
  <lasname>Meyer</lasname>
 </person>
 <person>
  <firstname def="ghi">Daniel</firstname>
  <lastname>Hottinger</lastname>
 </person>
 <person>
  <firstname abc="def">Paul</firstname>
  <lastname>Sevinç</lastname>
 </person>
</people>
~
~
~
~
~
~
~
testfile1.xml                 1,1           All
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<people>
 <person female="true">
  <firstname>Franziska</firstname>
  <lastname>Meyer</lastname>
 </person>
 <person>
  <firstname>Daniel</firstname>
  <lastname>Hottinger</lastname>
 </person>
 <person>
  <firstname>Paul</firstname>
  <lastname abc="def">Sevinç</lastname>
 </person>
</people>
~
~
~
~
~
~
~
testfile2.xml                 1,1           All
```

Detecting Changes

Text-based Tools Will Not Work



Goals

- Define a set of comparison criteria
- Compare different algorithms
- Implementation of a good algorithm in Java

Implementation In Java

- Generic data model
  - Original tree should be preserved (context of change)
  - It should be possible to look up a change for a given node
  - Full support for all XML features
  - We use DOM to represent XML trees
- First choice: Fast Match / Edit Script
  - Good performance
  - Reference implementation in Python, but
    - Script crashes sometimes
    - Implementation not performant
    - Parts in C, not 64-bit clean
- Port to Java is not trivial

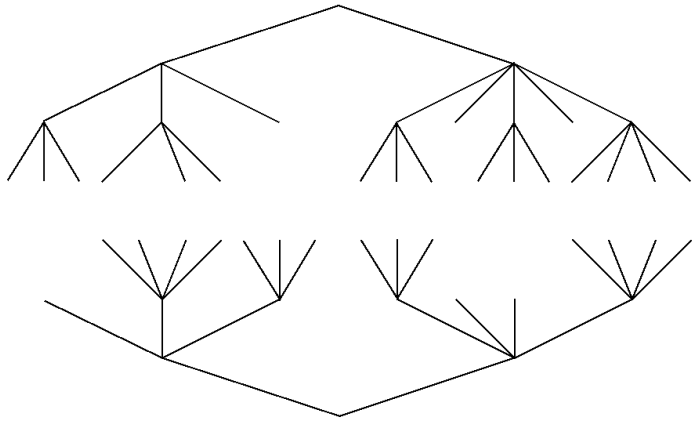
Implementation In Java

Fast Match / Edit Script (Simplified)

Fast Match

- Let $S_1 \leftarrow \text{leaves}(T_1)$
- Let $S_2 \leftarrow \text{leaves}(T_2)$
- Let $lcs \leftarrow LCS(S_1, S_2)$
- Add each pair of nodes $(x, y) \in lcs$ to matching M
- For each unmatched node $x \in S_1$, if there is an unmatched node $y \in S_2$ such that $equal(x, y)$
  - Add (x, y) to M
  - Mark x and y as "matched"
- Repeat with in-order traversal of inner nodes

Example:



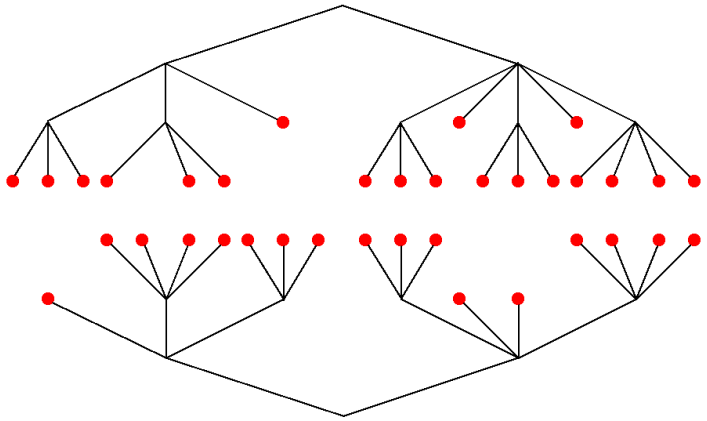
Implementation In Java

Fast Match / Edit Script (Simplified)

Fast Match

- Let $S_1 \leftarrow \text{leaves}(T_1)$
- Let $S_2 \leftarrow \text{leaves}(T_2)$
- Let $lcs \leftarrow LCS(S_1, S_2)$
- Add each pair of nodes $(x, y) \in lcs$ to matching M
- For each unmatched node $x \in S_1$, if there is an unmatched node $y \in S_2$ such that $equal(x, y)$
  - Add (x, y) to M
  - Mark x and y as "matched"
- Repeat with in-order traversal of inner nodes

Example:



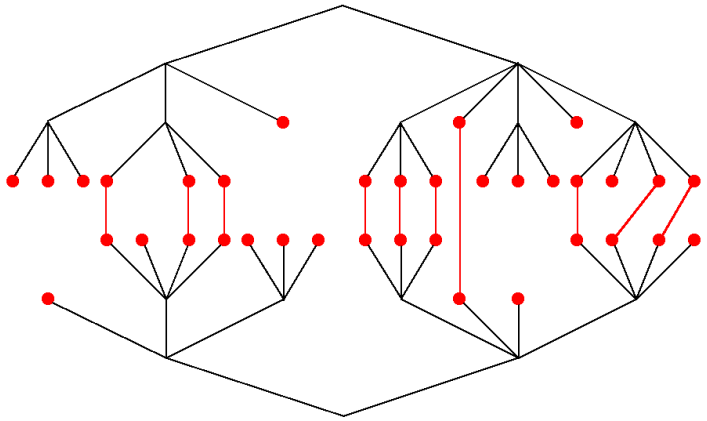
Implementation In Java

Fast Match / Edit Script (Simplified)

Fast Match

- Let $S_1 \leftarrow \text{leaves}(T_1)$
- Let $S_2 \leftarrow \text{leaves}(T_2)$
- Let $lcs \leftarrow LCS(S_1, S_2)$
- Add each pair of nodes $(x, y) \in lcs$ to matching M
- For each unmatched node $x \in S_1$, if there is an unmatched node $y \in S_2$ such that $equal(x, y)$
  - Add (x, y) to M
  - Mark x and y as "matched"
- Repeat with in-order traversal of inner nodes

Example:



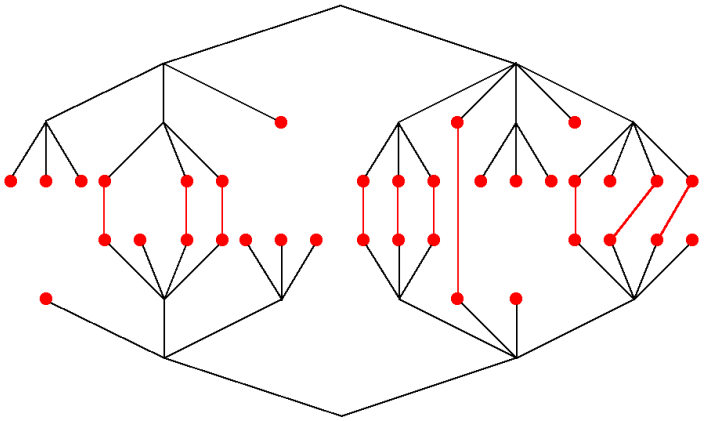
Implementation In Java

Fast Match / Edit Script (Simplified)

Fast Match

- Let $S_1 \leftarrow \text{leaves}(T_1)$
- Let $S_2 \leftarrow \text{leaves}(T_2)$
- Let $lcs \leftarrow LCS(S_1, S_2)$
- Add each pair of nodes $(x, y) \in lcs$ to matching M
- For each unmatched node $x \in S_1$, if there is an unmatched node $y \in S_2$ such that $equal(x, y)$
  - Add (x, y) to M
  - Mark x and y as "matched"
- Repeat with in-order traversal of inner nodes

Example:



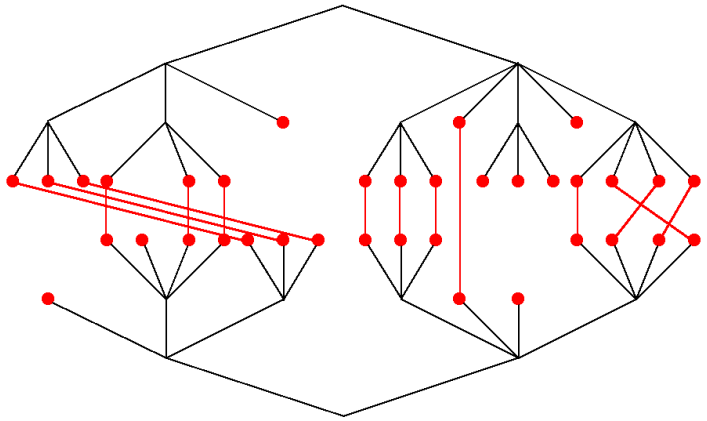
Implementation In Java

Fast Match / Edit Script (Simplified)

Fast Match

- Let $S_1 \leftarrow \text{leaves}(T_1)$
- Let $S_2 \leftarrow \text{leaves}(T_2)$
- Let $lcs \leftarrow LCS(S_1, S_2)$
- Add each pair of nodes $(x, y) \in lcs$ to matching M
- For each unmatched node $x \in S_1$, if there is an unmatched node $y \in S_2$ such that $equal(x, y)$
  - Add (x, y) to M
  - Mark x and y as “matched”
- Repeat with in-order traversal of inner nodes

Example:



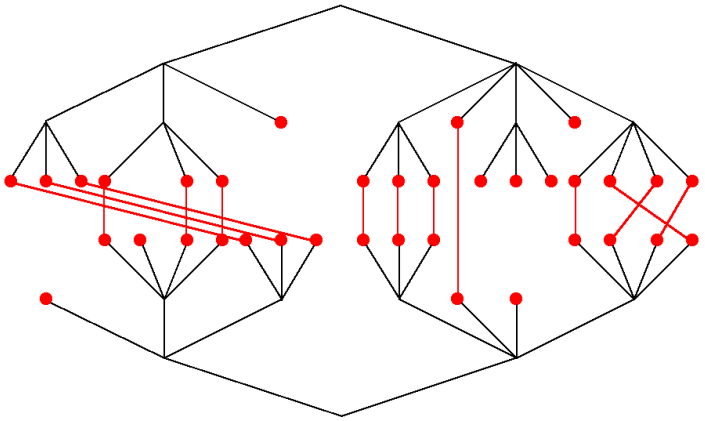
Implementation In Java

Fast Match / Edit Script (Simplified)

Fast Match

- Let $S_1 \leftarrow leaves(T_1)$
- Let $S_2 \leftarrow leaves(T_2)$
- Let $lcs \leftarrow LCS(S_1, S_2)$
- Add each pair of nodes $(x, y) \in lcs$ to matching M
- For each unmatched node $x \in S_1$, if there is an unmatched node $y \in S_2$ such that $equal(x, y)$
  - Add (x, y) to M
  - Mark x and y as "matched"
- Repeat with in-order traversal of inner nodes

Example:



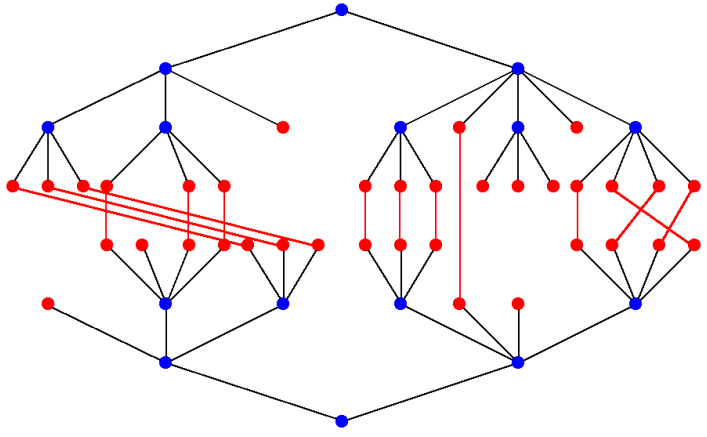
Implementation In Java

Fast Match / Edit Script (Simplified)

Fast Match

- Let $S_1 \leftarrow$ in-order(T_1)
- Let $S_2 \leftarrow$ in-order(T_2)
- Let $lcs \leftarrow LCS(S_1, S_2)$
- Add each pair of nodes $(x, y) \in lcs$ to matching M
- For each unmatched node $x \in S_1$, if there is an unmatched node $y \in S_2$ such that $equal(x, y)$
  - Add (x, y) to M
  - Mark x and y as "matched"
- Repeat with in-order traversal of inner nodes

Example:



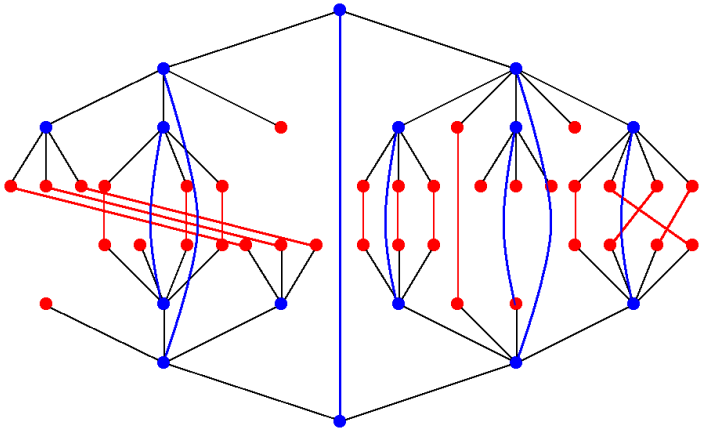
Implementation In Java

Fast Match / Edit Script (Simplified)

Fast Match

- Let $S_1 \leftarrow$ in-order(T_1)
- Let $S_2 \leftarrow$ in-order(T_2)
- Let $lcs \leftarrow LCS(S_1, S_2)$
- Add each pair of nodes $(x, y) \in lcs$ to matching M
- For each unmatched node $x \in S_1$, if there is an unmatched node $y \in S_2$ such that $equal(x, y)$
  - Add (x, y) to M
  - Mark x and y as "matched"
- Repeat with in-order traversal of inner nodes

Example:



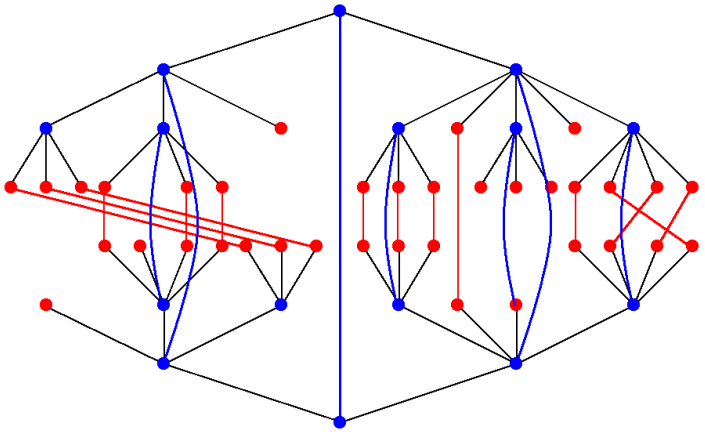
Implementation In Java

Fast Match / Edit Script (Simplified)

Fast Match

- Let $S_1 \leftarrow$ in-order(T_1)
- Let $S_2 \leftarrow$ in-order(T_2)
- Let $lcs \leftarrow LCS(S_1, S_2)$
- Add each pair of nodes $(x, y) \in lcs$ to matching M
- For each unmatched node $x \in S_1$, if there is an unmatched node $y \in S_2$ such that $equal(x, y)$
  - Add (x, y) to M
  - Mark x and y as "matched"
- Repeat with in-order traversal of inner nodes

Example:



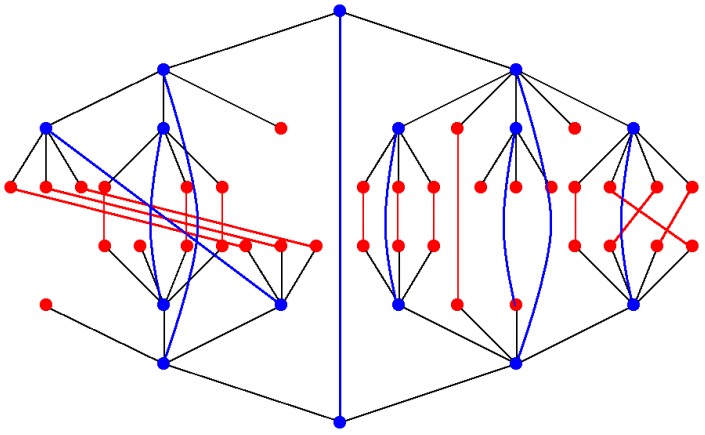
Implementation In Java

Fast Match / Edit Script (Simplified)

Fast Match

- Let $S_1 \leftarrow$ in-order(T_1)
- Let $S_2 \leftarrow$ in-order(T_2)
- Let $lcs \leftarrow LCS(S_1, S_2)$
- Add each pair of nodes $(x, y) \in lcs$ to matching M
- For each unmatched node $x \in S_1$, if there is an unmatched node $y \in S_2$ such that $equal(x, y)$
  - Add (x, y) to M
  - Mark x and y as “matched”
- Repeat with in-order traversal of inner nodes

Example:



Implementation In Java

Problems / Benefits

- Problems
  - The algorithm only works if all steps are implemented
    - Hard to trace bugs
  - Special handling of different node types
  - Some cases are undefined in the original paper
  - Readability of the code is important
  - We do not want to update T_1 except for inserts

Implementation In Java

Problems / Benefits

- Problems
  - The algorithm only works if all steps are implemented
    - Hard to trace bugs
  - Special handling of different node types
  - Some cases are undefined in the original paper
  - Readability of the code is important
  - We do not want to update T_1 except for inserts
- Benefits
  - AMD Athlon 64 3200+
  - Two 820 KB XML files (37500 Tags, 6400 Attributes)

Implementation In Java

Problems / Benefits

- Problems
  - The algorithm only works if all steps are implemented
    - Hard to trace bugs
  - Special handling of different node types
  - Some cases are undefined in the original paper
  - Readability of the code is important
  - We do not want to update T_1 except for inserts
- Benefits
  - AMD Athlon 64 3200+
  - Two 820 KB XML files (37500 Tags, 6400 Attributes)
  - Python implementation: 9:50

Implementation In Java

Problems / Benefits

- Problems
  - The algorithm only works if all steps are implemented
    - Hard to trace bugs
  - Special handling of different node types
  - Some cases are undefined in the original paper
  - Readability of the code is important
  - We do not want to update T_1 except for inserts
- Benefits
  - AMD Athlon 64 3200+
  - Two 820 KB XML files (37500 Tags, 6400 Attributes)
  - Python implementation: 9:50
  - Java implementation: 0:20

Evaluation Of XML-Diff Algorithms

Criteria For Comparison

- Size of edit script
- Complexity
- Supported operations
  - Insert
  - Delete
  - Rename
  - Move/copy

Evaluation Of XML-Diff Algorithms

Example Criteria

- Insertion and deletion of tags and attributes
- Changing the order of attributes
- Support for namespaces
- Support for XML-Schema and/or DTD
  (no other schema languages mentioned in papers)

The same criteria are used for testing the implementation.

Algorithm Ideas

- FMES (already seen)
  - Matches nodes using LCS
- X-Diff
  - Unordered trees
  - Very strong assumptions
- XyDiff
  - Matches subtrees (uses ID attributes when available)
  - Tries to match parents of matched subtrees

Detection Of Moves

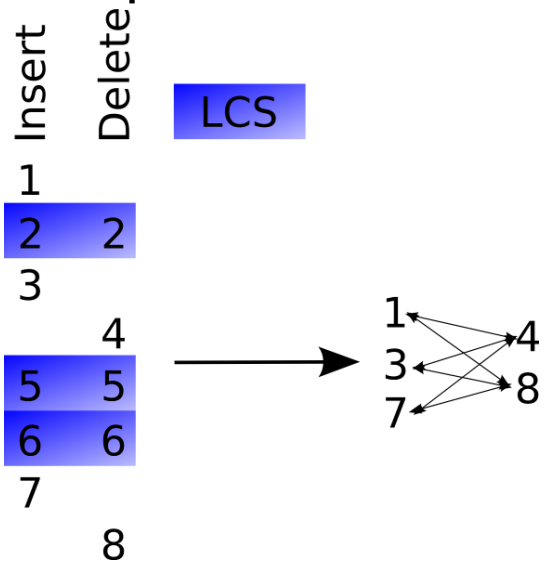
Is It Really NP-Hard?

- Cobéna et al. say that minimal edit scripts with move detection are NP-hard, see Zhang et al.
- Zhang's proof seems correct, but it tackles another problem . . .

Detection Of Moves

A Simple Idea

Compare all inserts with all deletes:




```
<tagA/>
<tagB/>   <tagB/>
<tagC/>
          <tagG/>
<tagE/>   <tagE/>
<tagF/>   <tagF/>
<tagG/>
          <tagH/>
```

Summary

Summary

- XML needs special diff algorithms
- Different algorithms implement different operations
- Detection of moves seems to be hard in general

Outlook

- There are more algorithms to implement
- We don’t know whether move detection is NP-hard or not

For Further Reading


J. W. Hunt and M. D. McIlroy.
An Algorithm for Differential File Comparison.


Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom.
Change detection in hierarchically structured information.


Grégory Cobéna, Talel Abdessalem, and Yassine Hinnach.
A comparative study for XML change detection.


Kaizhong Zhang, Jason T. L. Wang, and Dennis Shasha.
On the editing Distance between undirected acyclic Graphs.