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
Problem solved:

- Line-by-line diff using the Longest Common Subsequence (LCS) algorithm
- Edit script (qed commands, unified diff, . . .)
- GNU Diffutils
- SCM systems such as: CVS, Subversion, git, Bitkeeper, . . .
Detecting Changes
Why XML Is Different

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

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

<?xml version="1.0" encoding="ISO-8859-1"?>
<people>
  <person female="true">
    <firstname>Fransizka</firstname>
    <lastname>Meyer</lastname>
  </person>
  <person>
    <firstname>Daniel</firstname>
    <lastname>Hottinger</lastname>
  </person>
  <person>
    <firstname>Paul</firstname>
    <lastname>Sevinç</lastname>
  </person>
</people>

testfile1.xml 1,1 All testfile2.xml 1,1 All
## Detecting Changes

Text-based Tools Will Not Work

<table>
<thead>
<tr>
<th>testfile1.xml [+]</th>
<th>1.45</th>
</tr>
</thead>
<tbody>
<tr>
<td>All testfile2.xml [+]</td>
<td>1.45</td>
</tr>
</tbody>
</table>
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 \( lcs \leftarrow \text{LCS}(S_1, S_2) \)
- 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 \( \text{equal}(x, y) \)
  - 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 $lcs \leftarrow \text{LCS}(S_1, S_2)$
- 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 $\text{equal}(x, y)$
  - Add $(x, y)$ to $M$
  - Mark $x$ and $y$ as “matched”
- Repeat with in-order traversal of inner nodes

**Example:**

[Diagram of two trees with matching nodes highlighted]
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:
Fast Match

- Let $S_1 \leftarrow \text{leaves}(T_1)$
- Let $S_2 \leftarrow \text{leaves}(T_2)$
- Let $lcs \leftarrow \text{LCS}(S_1, S_2)$
- 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 $\text{equal}(x, y)$
  - 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
Fast Match

- Let $S_1 \leftarrow \text{leaves}(T_1)$
- Let $S_2 \leftarrow \text{leaves}(T_2)$
- Let $lcs \leftarrow \text{LCS}(S_1, S_2)$
- 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 $\text{equal}(x, y)$
  - Add $(x, y)$ to $M$
  - Mark $x$ and $y$ as “matched”

- Repeat with in-order traversal of inner nodes
**Implementation In Java**

**Fast Match / Edit Script (Simplified)**

**Fast Match**

- Let $S_1 \leftarrow \text{in-order}(T_1)$
- Let $S_2 \leftarrow \text{in-order}(T_2)$
- Let $lcs \leftarrow \text{LCS}(S_1, S_2)$
- 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 $\text{equal}(x, y)$
  - Add $(x, y)$ to $M$
  - Mark $x$ and $y$ as “matched”
- Repeat with in-order traversal of inner nodes

**Example:**

![Diagram showing matching nodes and inner nodes traversal]
Fast Match

- Let $S_1 \leftarrow \text{in-order}(T_1)$
- Let $S_2 \leftarrow \text{in-order}(T_2)$
- Let $lcs \leftarrow \text{LCS}(S_1, S_2)$
- 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 $\text{equal}(x, y)$
  - Add $(x, y)$ to $M$
  - Mark $x$ and $y$ as “matched”
- Repeat with in-order traversal of inner nodes
Fast Match

- Let \( S_1 \leftarrow \text{in-order}(T_1) \)
- Let \( S_2 \leftarrow \text{in-order}(T_2) \)
- 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 \( \text{equal}(x, y) \)
  - Add \((x, y)\) to \( M\)
  - Mark \( x \) and \( y \) as “matched”
- Repeat with in-order traversal of inner nodes
Implementation In Java
Fast Match / Edit Script  (Simplified)

Fast Match

- Let $S_1 \leftarrow \text{in-order}(T_1)$
- Let $S_2 \leftarrow \text{in-order}(T_2)$
- 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:

Insert | Delete
1      | 1
2      | 2
3      | 3
4      | 4
5      | 5
6      | 6
7      | 7
8      | 8

LCS

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

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