Efficient String-based XML-Prefiltering

Stefan Böttcher, Rita Hartel, Steffen Weber
Motivation

/site/people/person[@id="person0"]/name/text()

/site/people/person/@id
/site/people/person/name#

XML-Filter

Better total run-time!
Motivation

/site/people/person[@id="person0"]/name/text()

factor 74 smaller XML data

factor 13 better run-time

XQuery
Result

XQuery
Result

Better total run-time!
String Search Algorithms
(Boyer-Moore / Commentz-Walter) as an alternative to SAX parsing

P: <subtitle>
T: <site><books><book id="42"><title>X</title><subtitle>...

Only 20 of the 53 characters of the text have to be read!
(10 of them to verify the match)
String Search Algorithms (Boyer-Moore / Commentz-Walter) as an alternative to SAX parsing

Only 20 of the 53 characters of the text have to be read!
(10 of them to verify the match)
String Search Algorithms
(Boyer-Moore / Commentz-Walter) as an alternative to SAX parsing

Only 20 of the 53 characters of the text have to be read!
(10 of them to verify the match)
String Search Algorithms
(Boyer-Moore / Commentz-Walter) as an alternative to SAX parsing

Only 20 of the 53 characters of the text have to be read!
(10 of them to verify the match)
String Search Algorithms
(Boyer-Moore / Commentz-Walter) as an alternative to SAX parsing

P: <subtitle>
T: <site><books><book id="42"><title>X</title><subtitle>...

Only 20 of the 53 characters of the text have to be read!
(10 of them to verify the match)
String Search Algorithms
(Boyer-Moore / Commentz-Walter) as an alternative to SAX parsing

Only 20 of the 53 characters of the text have to be read!
(10 of them to verify the match)
String Search Algorithms
(Boyer-Moore / Commentz-Walter) as an alternative to SAX parsing

Only 20 of the 53 characters of the text have to be read!
(10 of them to verify the match)
String Search Algorithms
(Boyer-Moore / Commentz-Walter) as an alternative to SAX parsing

• Boyer-Moore / Commentz-Walter algorithms:
  – Run-time is sub-linear (on average)!
  – worst-case $O(n)$
  – best-case $O(n/m)$

Only 20 of the 53 characters of the text have been read!
(10 of them to verify the match)

searching multiple patterns at a time?
⇒ Commentz-Walter algorithm
String Filtering for XML-Filtering (1)

Projection path /a/b#

Example XML file

```xml
<a>
  <b id="1">11</b>
  <c><b id="2">22</b></c>
</a>
```

Example Schema

```xml
<!ELEMENT a (b|c)*)
<!ELEMENT b (#PCDATA)>
<!ATTLIST b id CDATA #REQUIRED>
<!ELEMENT c (b,b?)>
```

Why it is NOT SUFFICIENT to search only for the following tags:

```xml
<a>  <b> </a>  <b> </b>
</a>  </b>
```

**Wrong Output**

```xml
<a><b id="1">11</b><b id="2">22</b></a>
```

**Problem:** output contains **b-element** in line 3!
String Filtering for XML-Filtering (2)

Projection path /a/b#

Example XML file

```xml
<a>
  <b id="1">11</b>
  <c><b id="2">22</b></c>
</a>
```

Example schema

```xml
<!ELEMENT a (b|c)*>  
<!ELEMENT b (#PCDATA)>  
<!ATTLIST b id CDATA #REQUIRED>  
<!ELEMENT c (b,b?)>  
```

Tags to be searched

```
<a>  
<b>  
<c>  
</a>  
</b>  
</c>
```

Output

```
<a><b id="1">11</b> <c><b id="2">22</b></c> </a>
```

Solution of the problem: Analysis of the schema!
Interpretation as an Automaton
for the given example schema and given projection path /a/b#

example XML file

```xml
<a>
  <b id="1">11</b>
  <c><b id="2">22</b></c>
</a>
```

Output

```xml
<a><b id="1">11</b></a>
```
Frontier-Patterns and Jump-Offsets for the given example schema and given projection path /a/b#.

Why do no closing paranthesis occur in Frontier-Pattern?
1. attributes
2. empty Tags
3. XML allows whitespace

---

<table>
<thead>
<tr>
<th>$q$</th>
<th>Frontier-Pattern[$q$]</th>
<th>Action[$q$]</th>
<th>Jump-Offset[$q$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>{$a$}</td>
<td>nop</td>
<td>0</td>
</tr>
<tr>
<td>$q_1$</td>
<td>{$a$, $b$, $c$}</td>
<td>copy tag</td>
<td>0</td>
</tr>
<tr>
<td>$\hat{q}_1$</td>
<td>{}</td>
<td>copy tag</td>
<td>0</td>
</tr>
<tr>
<td>$q_2$</td>
<td>{$b$}</td>
<td>copy on</td>
<td>0</td>
</tr>
<tr>
<td>$\hat{q}_2$</td>
<td>{$a$, $b$, $c$}</td>
<td>copy off</td>
<td>0</td>
</tr>
<tr>
<td>$q_3$</td>
<td>{$c$}</td>
<td>nop</td>
<td>10</td>
</tr>
<tr>
<td>$\hat{q}_3$</td>
<td>{$a$, $b$, $c$}</td>
<td>nop</td>
<td>0</td>
</tr>
</tbody>
</table>
XML Schema (XSD)
Rule Graph & Larger Jump-Offsets

```xml
<!ATTLIST b id CDATA #REQUIRED>
...<c><b id=""/></c>...

10

<xsd:attribute name="id" type="xsd:int" use="required"/>
...<c><b id="1"/></c>...

11

<xsd:element name="b" type="xsd:int"/>
...<c><b id="1">1</b></c>...

15

Minimum lengths
- int: 1
- string: 0
- date: 10
- ...
```
Recursive Schema (1) and new projection path /a/b/b# 

New example XML file

```xml
<a>
  <b id="1">\b id="11"/</b>
  <c><b id="2">22</b></c>
</a>
```

New example schema

```xml
<!ELEMENT a (b|c)*/>
<!ELEMENT b (b?)>
<!ATTLIST b id CDATA #REQUIRED>
<!ELEMENT c (b,b?)>
```

The diagram shows the automata construction process for the given schema. The automaton transitions are labeled with the corresponding XML elements and attributes, demonstrating how the system skips until the corresponding `</b>` tag is encountered due to the new projection path /a/b/b#.
Recursive Schema (2) 
and another new projection path \(/a/b//b\)

Automata expansion at run-time: adapt automaton to the depth of the structure of the XML stream
Attribute Filters
and another new projection path /a/b[@id=1]#

If [@id=1] is true
<b> → ‘copy on’
</b> → ‘copy off’

If [@id=1] is false
<b> ... </b> → ‘nop’

2 active pairs of states for b:
If filter is true  <b> ... </b> - with Action ‘copy tag’ or ‘copy on’/‘copy off’
If filter is false  <b> ... </b> - with Action ‘nop’  (→ i.e., nothing is copied)
Automata Construction (Overview)

1. Rule Graph
2. Element Templates
3. Projection Paths
4. Automaton

Schema
Automata Construction (1)

Element templates

Element template a

Element template b

Element template c

![XML Element Templates](image)
Automata Construction (2)

projection path /a/b/b#
Automata Construction (2)
different projection path /a/b//b

- element template a
- element template b
- element template c
Benchmarks

• **System configuration**
  – Intel Core 2 Duo T9400 and 4 GB main memory
  – Gentoo Linux 64 Bit with Kernel 2.6.37
  – Java 6.0.23

• **Approach taken**
  – 2 GB XMark
  – XMark-XSD generated from official XMark-DTD and augmented by missing data types
  – Excluded disk from being bottleneck
**Size of Projected Documents**

- Positive effect of attribute filters („AF“)
- **Q14:** needs content of all description-elements

---

Stefan Böttcher, Rita Hartel, Steffen Weber (University of Paderborn, Germany): Efficient String-based XML Prefiltering
• **Q5**: only one projection path, long tag names like `closed_auctions`
• **Q10**: 12 projection paths
Comparison with Previous C++-based SMP

Runtime Comparison (2 GB XMark)

- StringFilter 2.0 DTD
- StringFilter 2.0 XSD
- StringFilter 1.0
- SMP
Scalability

![Scalability of StringFilter (XMark Query 1)](image)

- Runtime (Seconds)
- Memory Usage (MB)
- Size of XMark Document (MB)

Stefan Böttcher, Rita Hartel, Steffen Weber (University of Paderborn, Germany): Efficient String-based XML Prefiltering
Runtime in combination with Qizx

For XMark documents > 1 GB, evaluation of some XQuery queries is only possible with our StringFilter!
Scalability in Combination with Qizx

Scalability of StringFilter + Qizx (XMark Q1)

- Runtime (Seconds)
- Size of XMark Document (MB)
- StringFilter + Qizx-filtered
- Qizx

Stefan Böttcher, Rita Hartel, Steffen Weber (University of Paderborn, Germany): Efficient String-based XML Prefiltering
Comparison with XML Filters Evaluated on SAX

- StringFilter is faster than query evaluation using SAX-based XML parsing
- Query evaluation on compressed XML may be competitive ➔ next talk
Summary

- **Faster XQuery query evaluation**
  - String search algorithms skip irrelevant sub-strings
  - Automaton contains currently relevant sub-string matches

- **StringFilter 2.0**
  - Uses XML Schema (XSD)
  - Supports recursive schemas
  - Supports attribute filters
Conclusions (1) – Possible Improvements

1. **String-Search algorithms**: fewer character comparisons

   - Possible Improvements
     1. String-Search algorithms: fewer character comparisons
     2. Restrictions for data types: larger Jump-Offsets

   - Exclude these comparisons by schema analysis

2. **Restrictions for data types**: larger Jump-Offsets

   ```xml
   <xsd:simpleType name="plz">
     <xsd:restriction base="xsd:integer">
       <xsd:minInclusive value="1000"/>
       <xsd:minInclusive value="99999"/>
     </xsd:restriction>
   </xsd:simpleType>
   ```

   3 characters more than without data type restriction
Conclusions (2) – Possible Extensions

3. Better addressing of following-sibling axes

4. StringFilter as a StAX-Parser
   - High performance by String search algorithms
   - ”SkipToMatchingEnd“
   - Projection paths as filters
Thank you!

Questions?
Thank you!

Questions?
Memory Usage

Memory Usage (without JVM)

<table>
<thead>
<tr>
<th>Qn</th>
<th>Memory Usage (MB)</th>
<th>RuleGraph</th>
<th>StringFilter</th>
<th>Created Node-Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory Usage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- RuleGraph
- StringFilter
- Created Node-Pairs
Example Schema
non-recursive

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="a">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="b"/>
        <xsd:element ref="c"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="b">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:int">
          <xsd:attribute name="id" type="xsd:int" use="required"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="c">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="b"/>
        <xsd:element minOccurs="0" maxOccurs="1" ref="b"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
Example Schema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="a">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="b"/>
        <xsd:element ref="c"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="b">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="b" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:int" use="required"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="c">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="b"/>
        <xsd:element minOccurs="0" maxOccurs="1" ref="b"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
Commentz-Walter Algorithm
„Boyer-Moore for multiple patterns“

<table>
<thead>
<tr>
<th>single pattern</th>
<th>forward comparison</th>
<th>backward comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiple patterns</td>
<td>Knuth-Morris-Pratt</td>
<td>Boyer-Moore</td>
</tr>
<tr>
<td></td>
<td>Aho-Corasick</td>
<td>Commentz-Walter</td>
</tr>
</tbody>
</table>

\[ P = \{ \text{<author, <title, <subtitle}>} \]
### Computation of the Jump-Offsets

#### Non-recursive schema

![Diagram of automata](image)

<table>
<thead>
<tr>
<th>edge is</th>
<th>destination state is relevant</th>
<th>destination state is irrelevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>between same pair of states</td>
<td>minimum length of text contents</td>
<td>minimum length of text content + 1 or + length of destination tag</td>
</tr>
<tr>
<td>others</td>
<td>0</td>
<td>length of destination tag</td>
</tr>
</tbody>
</table>

**Table:**

- **q0** is the initial state.
- **q5** is a terminal state.
- Transitions are labeled with the length of the edge and the label of the source state.
- The jump offsets are computed as follows:
  - For transitions between the same pair of states, the jump offset is the minimum length of text contents.
  - For other transitions, the jump offset is 0.
  - For transitions involving a terminal state, the jump offset is the length of the destination tag.

**Schema:**

- Nodes represent states, and edges represent transitions with labels.
- States are colored to distinguish between relevant and irrelevant destinations.
- Jump offsets are indicated by numbers next to the states.
Computation of the Jump-Offsets

Recursive schema

<table>
<thead>
<tr>
<th>edge is</th>
<th>destination state is relevant</th>
<th>destination state is irrelevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>between same</td>
<td>minimum length of text contents</td>
<td>minimum length of text content + 1 or + length of destination tag</td>
</tr>
<tr>
<td>pair of states</td>
<td></td>
<td></td>
</tr>
<tr>
<td>others</td>
<td>0</td>
<td>length of destination tag</td>
</tr>
</tbody>
</table>

Stefan Böttcher, Rita Hartel, Steffen Weber (University of Paderborn, Germany): Efficient String-based XML Prefiltering
Rule Graph
Recursive schema

Rule tree a
Rule tree b
Rule tree c
Automata Construction

Treatment of minOccurs / maxOccurs

```xml
<!ELEMENT c (b,b?)>
<xsd:sequence>
  <xsd:element ref="b"/>
  <xsd:element minOccurs="0" maxOccurs="1" ref="b"/>
</xsd:sequence>

<xsd:sequence>
  <xsd:element minOccurs="1" maxOccurs="2" ref="b"/>
</xsd:sequence>
```
Jump-Offsets

Characters skipped by Jump-Offsets

- Skipped Characters
- Total Jump-Offsets
- Actually Skipped Characters
Pattern-Matcher Cache

Pattern-Matcher Instances

- BM
- BM (without Cache)
- CW
- CW (without Cache)

Stefan Böttcher, Rita Hartel, Steffen Weber (University of Paderborn, Germany): Efficient String-based XML Prefiltering
Conclusions (3) - Possible Improvements

5. Relevant positions: more precise addressing

- /a/c/b/following-sibling::b#

- /a/c/b/following-sibling::d#

<!ELEMENT c (b?,d)>
Conclusions (4) – Possible Improvements

6. **Stricter mode**: if space is forbidden:

   `<b id="1">`   `<b/>`   `<b>`   `<b id="1">`