Web Data Extraction

Craig Knoblock
University of Southern California

This presentation is based on slides prepared by Ion Muslea and Kristina Lerman
Extracting Data from Semi-structured Sources

NAME Casablanca Restaurant
STREET 220 Lincoln Boulevard
CITY Venice
PHONE (310) 392-5751
Approaches to Wrapper Construction

- Manual Wrapper Construction
- Learning-based Wrapper Construction
- Automatic Wrapper Construction
Grammar Induction Approach

- Pages automatically generated by scripts that encode results of db query into HTML
  - Script = grammar
- Given a set of pages generated by the same script
  - Learn the grammar of the pages
    - Wrapper induction step
  - Use the grammar to parse the pages
    - Data extraction step
RoadRunner: Towards Automatic Data Extraction from Large Web Sites by Crescenzi, Mecca, & Merialdo
RoadRunner Overview

- Automatically generates a wrapper from large web pages
  - Pages of the same *class*
  - No dynamic content from javascript, ajax, etc
- Infers source schema
  - Supports nested structures and lists
  - Extracts data from pages
- Efficient approach to large, complex pages with regular structure
Example Pages

- Compares two pages at a time to find similarities and differences
- Infers nested structure (schema) of page
- Extracts fields
### Extracted Result

**Total number of SCHEMAS found: 1**

**Schema Number 1:**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>Database Primer</td>
<td>1998</td>
<td>$20</td>
<td></td>
<td>This book introduces the reader to the theory and technology...[TRUNCATED]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paul Jones</td>
<td>XML at Work</td>
<td>1999</td>
<td>$30</td>
<td></td>
<td>A comprehensive description of XML and all related standards...[TRUNCATED]</td>
</tr>
<tr>
<td>HTML and Scripts</td>
<td>First Edition, Paperback</td>
<td>1993</td>
<td>$30</td>
<td></td>
<td>A useful HTML handbook, with a good tutorial on the use of...[TRUNCATED]</td>
</tr>
<tr>
<td>JavaScripts</td>
<td>HTML</td>
<td>2000</td>
<td>$50</td>
<td></td>
<td>A must in every Webmaster's bookshelf...</td>
</tr>
</tbody>
</table>
Union-Free Regular Expression (UFRE)

- Web page structure can be represented as Union-Free Regular Expression (UFRE)
  - UFRE is Regular Expressions without *disjunctions*
  - If $a$ and $b$ are UFRE, then the following are also UFREs
    - $a \cdot b$
    - $(a)^+$
    - $(a)^?$
Web page structure can be represented as Union-Free Regular Expression (UFRE)

- UFRE is Regular Expressions without disjunctions
- If $a$ and $b$ are UFRE, then the following are also UFREs
  - $a.b \rightarrow$ string fields
  - $(a)^+ \rightarrow$ lists (possibly nested)
  - $(a)? \rightarrow$ optional fields
- Strong assumption that usually holds
Approach

- Given a set of example pages
- Generate the *Union-Free Regular Expression* which contains example pages
- Find the least upper bounds on the RE lattice to generate a wrapper in *linear time*
- Reduces to finding the least upper bound on two UFREs
Matching/Mismatches

Given a set of pages of the same type

- Take the first page to be the *wrapper* (UFRE)
- Match each successive sample page against the wrapper
- *Mismatches* result in generalizations of wrapper
  - String mismatches
  - Tag mismatches
Matching/Mismatches

Given a set of pages of the same type

- Take the first page to be the *wrapper* (UFRE)
- Match each successive sample page against the wrapper
- *Mismatches* result in generalizations of wrapper
  - String mismatches
    - Discover fields
  - Tag mismatches
    - Discover optional fields
    - Discover iterators
Example Matching

- Wrapper (initially Page 1):

01: <HTML>
02: Books of:
03: <B>
04: John Smith
05: </B>
06: <UL>
07: <LI>
08-10: <I>Title:</I>
11: DB Primer
12: </LI>
13: <LI>
14-16: <I>Title:</I>
17: Comp. Sys.
18: </LI>
19: </UL>
20: </HTML>

terminal tag search and
square matching

- Sample (Page 2):

01: <HTML>
02: Books of:
03: <B>
04: Paul Jones
05: </B>
06: <IMG src=.../>
07: <UL>
08: <LI>
09-11: <I>Title:</I>
12: XML at Work
13: </LI>
14: <LI>
15-17: <I>Title:</I>
18: HTML Scripts
19: </LI>
20: </UL>
21-23: <I>Title:</I>
24: Javascript
25: </LI>
26: </UL>
27: </HTML>

- Wrapper after solving mismatches:

<HTML>
Books of:<B>PCDATA</B>
( <IMG src=.../> )?
<UL>
( <LI><I>Title:</I>PCDATA</LI> )+
</UL></HTML>
String Mismatches: Discovering Fields

- String mismatches are used to discover fields of the document
- Wrapper is generalized by replacing “John Smith” with #PCDATA

```html
<HTML>Books of: <B>John Smith
→ <HTML> Books of: <B>#PCDATA
```
Example Matching

- Wrapper (initially Page 1):
01:  <HTML>
02:  Books of:
03:  <B>
04:  John Smith
05:  </B>
06:  <UL>
07:  <LI>
08-10:  <I>Title:</I>
11:  DB Primer
12:  </LI>
13:  <LI>
14-16:  <I>Title:</I>
17:  Comp. Sys.
18:  </LI>
19:  </UL>
20:  </HTML>

- Sample (Page 2):
01:  <HTML>
02:  Books of:
03:  <B>
04:  Paul Jones
05:  </B>
06:  <IMG src=.../>
07:  <UL>
08:  <LI>
09-11:  <I>Title:</I>
12:  XML at Work
13:  </LI>
14:  <LI>
15-17:  <I>Title:</I>
18:  HTML Scripts
19:  </LI>
20:  </UL>
21-23:  <I>Title:</I>
24:  Javascript
25:  </LI>
26:  </UL>
27:  </HTML>

- Wrapper after solving mismatches:

```
<HTML>Books of:<B><PCDATA</B>
( <IMG src=.../> )?
<UL>
 ( <LI><I>Title:</I><PCDATA</LI> )+
</UL></HTML>
```
Tag Mismatches: Discovering Optionals

- First check to see if mismatch is caused by an iterator (described next)
- If not, could be an optional field in wrapper or sample
- Cross search used to determine possible optionals
- Image field determined to be optional:
  - ( <img src=.../> )?
Example Matching

Wrapper (initially Page 1):

01:  <HTML>
02:  Books of:
03:  <B>
04:  John Smith
05:  </B>
06:  <UL>
07:  <LI>
08-10:  <I>Title:</I>
11:  DB Primer
12:  </LI>
13:  <LI>
14-16:  <I>Title:</I>
17:  Comp. Sys.
18:  </LI>
19:  </UL>
20:  </HTML>

terminal tag search and square matching

String Mismatch

Sample (Page 2):

01:  <HTML>
02:  Books of:
03:  <B>
04:  Paul Jones
05:  </B>
06:  <IMG src=.../>
07:  <UL>
08:  <LI>
09-11:  <I>Title:</I>
12:  XML at Work
13:  </LI>
14:  <LI>
15-17:  <I>Title:</I>
18:  HTML Scripts
19:  </LI>
20:  <LI>
21-23:  <I>Title:</I>
24:  Javascript
25:  </LI>
26:  </UL>
27:  </HTML>

String Mismatch

String Mismatch

String Mismatch

Wrapper after solving mismatches:

<HTML>Books of:<B>#PCDATA</B>
 ( <IMG src=.../> )?
<UL>
 ( <LI><I>Title:</I>#PCDATA</LI> )+
</UL></HTML>
Tag Mismatches: Discovering Iterators

- Assume mismatch is caused by repeated elements in a list
  - End of the list corresponds to last matching token: </LI>
  - Beginning of list corresponds to one of the mismatched tokens: <LI> or </UL>
  - These create possible “squares”
- Match possible squares against earlier squares
- Generalize the wrapper by finding all contiguous repeated occurrences:
  - ( <LI><I>Title:</I>#PCDATA</LI> )+
Example Matching

- Wrapper (initially Page 1):

01: <HTML>
02: Books of:
03: <B>
04: John Smith
05: </B>
06: <UL>
07: <LI>
08-10: <I>Title:</I>
11: DB Primer
12: </LI>
13: <LI>
14-16: <I>Title:</I>
17: Comp. Sys.
18: </LI>
19: </UL>
20: </HTML>

- Sample (Page 2):

01: <HTML>
02: Books of:
03: <B>
04: Paul Jones
05: </B>
06: <IMG src=.../>
07: <UL>
08: <LI>
09-11: <I>Title:</I>
12: XML at Work
13: </LI>
14: <LI>
15-17: <I>Title:</I>
18: HTML Scripts
19: </LI>
20: <LI>
21-23: <I>Title:</I>
24: Javascript
25: </LI>
26: </UL>
27: </HTML>

- Wrapper after solving mismatches:

<HTML>Books of:<B>#PCDATA</B>
( <IMG src=.../> )?
<UL>
( <LI><I>Title:</I>#PCDATA</LI> )+
</UL></HTML>
Internal Mismatches

- Generate *internal mismatch* while trying to match square against earlier squares on the *same page*
  - Solving internal mismatches yield further refinements in the wrapper
    - List of book editions
    - *<I>Special!</I>*
Recursive Example

- Wrapper (initially Page 1):
01-05: <HTML>Books of:<B>John Smith</B><UL>
07:  <LI>
08:  <B>Computer Systems</B><P>
12:  </B><P>
13:  </P></LI>
14:  </LI></UL>06:  <UL>
07:  <LI>
08:  <B>Database Primer</B><P>
20-22:  <I>Special!</I><P>
23:  </B><P>
24:  </B><P>
26:  </B><P>
27:  </P><P>
28:  </LI></UL>09-30:  </UL></HTML>

- Wrapper after solving mismatches:
<HTML>Books of:<B>&#PCDATA</B><UL>
</LI><LI>&#PCDATA</LI> (<B>&#PCDATA</B>
 ("Special!")?)<P>
</B><P></LI></UL>
</HTML>
Discussion

- Assumptions:
  - Pages are well-structured
  - Structure can be modeled by UFRE (no disjunctions)
- Search space for explaining mismatches is huge
  - Uses a number of heuristics to prune space
    - Limited backtracking
    - Limit on number of choices to explore
    - Patterns cannot be delimited by optionals
  - Will result in pruning possible wrappers
Limitations

- Learnable grammars
  - Union-Free Regular Expressions (RoadRunner)
    - Variety of schema structure: tuples (with optional attributes) and lists of (nested) tuples
    - Does not efficiently handle disjunctions – pages with alternate presentations of the same attribute
  - Context-free Grammars
    - Limited learning ability
- User needs to provide a set of pages of the same type
Inferlink Web Extraction Software
Inferlink Web Extraction Software

- Two phase processing
  - Step 1: Cluster the pages based on the layout of the pages
  - Step 2: Build a template to extract the data for each cluster
Inferlink Web Extraction Software: Clustering

- **Cluster**
  - Based on the visible text
  - Page is broken into chunks
    - These are continuous blocks of text
  - Search for common visible chunks
    - Remove chunks that occur in all pages
    - Remove chunks that occur in less than 10 pages
  - Greedy algorithm to cluster the pages based on the remaining chunks
    - Sort by the size of the clusters created by each chunk
Inferlink Web Extraction Software: Template Learning

- Input: cluster \{Pi\}
- Select 5 random pages to build a template
  - Tokenize on space & punctuation
  - Start with n-grams of tuples of size n, n=6
    - Find those n-grams that occur on all pages
    - Keep only those n-grams that occur exactly once per pages
    - Decompose pages based on these n-grams
    - Run algorithm recursive on decomposed page
  - Repeat above for size n-1 down to n=2
  - Construct template based on the decomposition
Discussion

- Inferlink approach solves some of the key limitations of Roadrunner
  - Pages do not all have to be of the same type
  - Multiple optionals would be treated as different page types
  - Scales well with complex pages
Demonstration
Web Data Extraction Software

- Beautiful Soup
  - [http://www.crummy.com/software/BeautifulSoup/](http://www.crummy.com/software/BeautifulSoup/)
  - Python library to manually write wrappers
- Jsoup
  - [http://jsoup.org/](http://jsoup.org/)
  - Java library to manually write wrappers
- ScrapyngHub
  - [http://scrapynghub.com/](http://scrapynghub.com/)
  - Portia provides a wrapper learner
- Others
  - Tell us if you find a good one!