Data Extraction from Web Data Sources

Jerome Robinson
Computer Science Department, University of Essex, Colchester, U.K.
robij@essex.ac.uk

Abstract

This paper provides an explanation of the basic data structures used in a new page analysis technique to create wrappers (data extractors) for the result pages produced by web sites in response to user qeries via web page forms. The key structure called a tpGrid is a representation of the web page, which is easier to analyse than the raw html code. The analysis looks for repetition patterns of sets of tagSets, which are defined in the paper.

1. Introduction

A *web data source* in the context of the current paper is a database backed web server which accepts queries in web page forms and returns the set of result items for that query in a web page. We therefore refer to query extraction from query result web pages, but the method of wrapper production and data extraction also works with any web page that displays one or more sets of items that form *collection* objects in the web page. (A collection is a set of items with similar appearance on screen).

The key component of a wrapper for a web data source is the Extractor program. It uses some description of web page format in order to identify and extract data items from the html code representing the web page. Wrapper production therefore requires a *page analysis* phase, before the extractor can be used for the first time. The purpose of page analysis is to create a suitable *page descriptor* for the wrapper to use repeatedly to extract data from future pages from that web site.

```
<HTML><HEAD><TITLE> Spa Guide </TITLE></HEAD>                   (S0)
<BODY> <H1> Nunohiki Spa </H1>                                  (S1)
 <EM> Location </EM>                                            (S2)
   <P> Kita-Saku district, Nagano pref. Japan </P>              (S3)
   <P> TEL: 0268–67–3467 </P>                                   (S4)
 <EM> Open Hours </EM>                                          (S5)
   <P> From 10.15 a.m to 11.30 p.m. </P>                        (S6)
 <EM> Overview </EM>                                            (S7)
   <P> This facility was established in 1987, and ...   <BR>   (S8)
 The hot spring is located at the front of ...                  (S9)
     <B> “SAWARA”, </B>                                         (S10)
     on the basement floor ... <BR>                             (S11)
     The effects of this hot spring are ... </P>                (S12)
</BODY></HTML>
```

Figure 1. An HTML document, showing textStrings and tagStrings

An HTML document contains only tags and text items. (Embedded program scripts are not part of the html code, and are removed before page structure analysis begins). A web page can therefore be modelled as the numbered sequence of tagString/textString pairs in its html code, where a textString is anything that is not inside angle brackets, so it is not a tag. A tagString is the sequence of tags that precede any textString. *For example*, the html document in Figure 1 shows each textString on a separate line. The lines have been labelled, in Figure 1, with the numbers of the textStrings S0, S1, S2, etc., in order to explain what is meant by a numbered sequence of text items in each html page.

The **tagString** preceding textString S0 is `<HTML><HEAD><TITLE>`. The tagString preceding textString S1 is `</TITLE></HEAD><BODY><H1>`. Each tagString/textString pair has the same number as the textString it contains. The numbered sequence of tagString/textString pairs for the whole HTML document in Figure 1 is shown in Figure 2.

0	<HTML><HEAD><TITLE>	Spa Guide
1	</TITLE></HEAD><BODY><H1>	Nunohiki Spa
2	</H1><EM>	Location
3	</EM><P>	Kita-Saku district, Nagano pref. Japan
4	</P><P>	TEL: 0268–67–3467
5	</P><EM>	Open Hours
6	</EM><P>	From 10.15 a.m to 11.30 p.m.
7	</P><EM>	Overview
8	</EM><P>	This facility was established in 1987, and ...
9	<BR>	The hot spring is located at the front of ...
10	<B>	“SAWARA”,
11	</B>	on the basement floor ...
12	<BR>	The effects of this hot spring are ...

Figure 2. The numbered list of tagString/ textString pairs

All textStrings are visible on the Web Page represented by an HTML document. So a search for repetition in text items in an html document is a suitable way to start the search for collection objects in a web page. Figure 1 shows a document that contains only one record, but the real web page listing spas has a similar record for each of a number of different spas. Each record is displayed on-screen in the same format as the other records, so that the web page structure is understandable to a person reading the page. The format is produced by tags in the html document, and we can discover repeating patterns in the sequence of tagStrings.

The task of searching for repetitive patterns is simplified by observing that w*ithin the local context of tagStrings, the* order *of tags is not important.* This is an experimental observation. So each tag**String** is represented by a **tagSet**. A tagSet is a *set* of tag *names* with a count value for each name that occurs in the corresponding tagString.

For example: The tagStrings shown in Figure 2 are too simple to illustrate the idea of tagSets, so a typical tagString from a real web page is now used:

```
</td></tr></table><table valign="top" width="757"
border="0" cellspacing="0" cellpadding="0"><tr><td
valign="top" width="146"><table border="0"
cellspacing="0" cellpadding="0" width="146"><tr><td
width="146"><img src="/images/spacer.gif" width=1
height=10 border="0" alt="></td></tr><tr><td
align="right" width="146" valign="middle"><a
class=header href="http://www.bl.uk/index.shtml">
```

This tagString precedes a textString in the html document. Ignoring tag attributes (inside the angle brackets) shows that it contains the following **tag names**:

</td></tr></table><table><tr><td><table><tr><td><img></td></tr><tr><td><a>

Start and end tags (such as td and /td) are treated as different tag names. Counting the number of times each tag name occurs in the tagString produces the tagSet:

</td – 2> </tr – 2> </table – 1> <table – 2> <tr – 3> <td – 3> <img – 1> <a – 1>

In order to facilitate tagSet comparisons, we represent each tagSet as a vector of count values. The number of elements in the vector is the number of tag names in the whole html document. There were 41 tag names in the document from which this tagString was taken, so its tagSet is represented as:

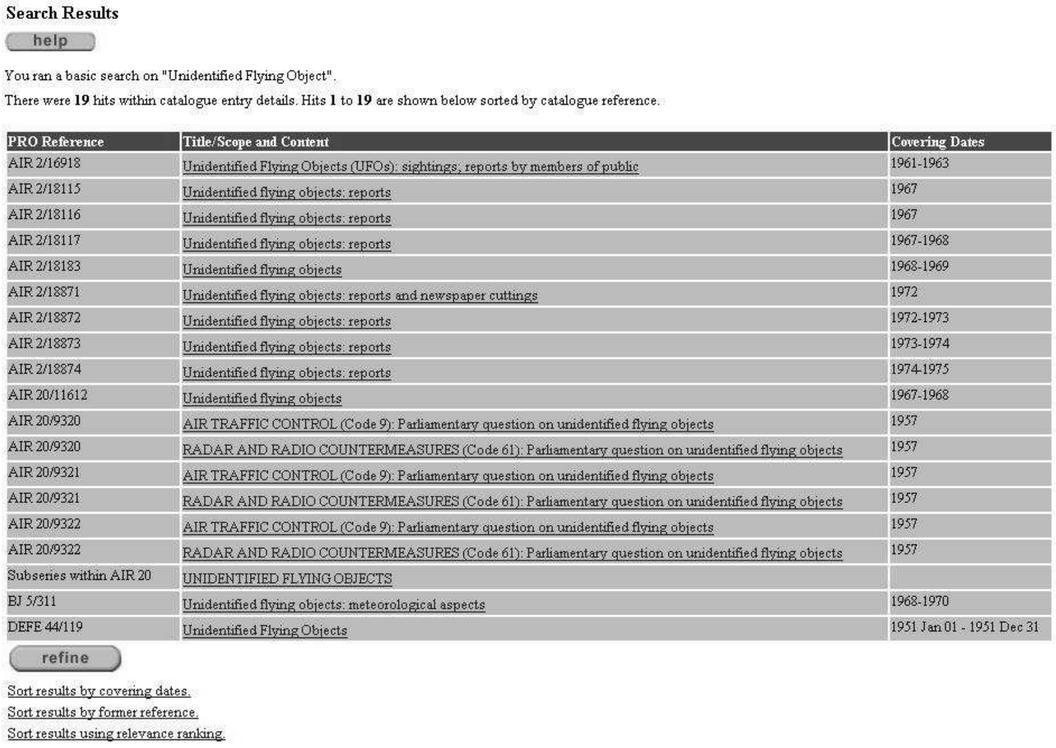
0,0,0,0,0,0,0,0,2,3,3,1,1,0,0,2,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Which is a 41-element vector of count values. The 41 tag names were:

html, head, title, /title, meta, link, /head, body, table, tr, td, a, img, /a, br, /td, /tr, /table, SPAN, div, /div, form, center, input, /center, hr, strong, /strong, p, font, /font, select, option, /select, b, /span, /FORM, blockquote, /blockquote, /body, /html.

2. Automatic wrapper production example

The web page shown in Figure 3 is the result of a search query on the UK Public Records Office catalogue. The query sought any reports on ‘unidentified flying object’.



Search Results

You ran a basic search on "Unidentified Flying Object".

There were **19** hits within catalogue entry details. Hits **1** to **19** are shown below sorted by catalogue reference.

PRO Reference	Title/Scope and Content	Covering Dates
AIR 2/16918	Unidentified Flying Objects (UFOs): sightings; reports by members of public	1961-1963
AIR 2/18115	Unidentified flying objects: reports	1967
AIR 2/18116	Unidentified flying objects: reports	1967
AIR 2/18117	Unidentified flying objects: reports	1967-1968
AIR 2/18183	Unidentified flying objects	1968-1969
AIR 2/18871	Unidentified flying objects: reports and newspaper cuttings	1972
AIR 2/18872	Unidentified flying objects: reports	1972-1973
AIR 2/18873	Unidentified flying objects: reports	1973-1974
AIR 2/18874	Unidentified flying objects: reports	1974-1975
AIR 20/11612	Unidentified flying objects	1967-1968
AIR 20/9320	AIR TRAFFIC CONTROL (Code 9): Parliamentary question on unidentified flying objects	1957
AIR 20/9320	RADAR AND RADIO COUNTERMEASURES (Code 61): Parliamentary question on unidentified flying objects	1957
AIR 20/9321	AIR TRAFFIC CONTROL (Code 9): Parliamentary question on unidentified flying objects	1957
AIR 20/9321	RADAR AND RADIO COUNTERMEASURES (Code 61): Parliamentary question on unidentified flying objects	1957
AIR 20/9322	AIR TRAFFIC CONTROL (Code 9): Parliamentary question on unidentified flying objects	1957
AIR 20/9322	RADAR AND RADIO COUNTERMEASURES (Code 61): Parliamentary question on unidentified flying objects	1957
Subseries within AIR 20	UNIDENTIFIED FLYING OBJECTS	
BJ 5/311	Unidentified flying objects: meteorological aspects	1968-1970
DEFE 44/119	Unidentified Flying Objects	1951 Jan 01 - 1951 Dec 31

Sort results by covering dates.
Sort results by former reference.
Sort results using relevance ranking.

Figure 3. A results page from http://catalogue.pro.gov.uk/

Each result is displayed on a separate line in the web page. The task, when creating a wrapper (data extractor program), is to find the set of results in the html document and find a way for the extractor program to identify each of the fields in each record in order to create a database table containing all the result records on the page. A *page analysis* operation is needed, to create a page description for the extractor to use. That page analysis process is now described. The method proceeds, as explained above, by first converting the html document into a numbered sequence of tagSet/textString pairs.

The first 25 textStrings in Figure 3’s html document are:

S0: Search Results
S1: You ran a basic search on "unidentified AND flying AND object" .
S2: &nbsp;
S3: There were 19 hits within catalogue entry details. Hits 1 to 19 are shown below sorted by catalogue reference.
S4: PRO Reference
S5: Title/Scope and Content
S6: Covering Dates
S7: AIR 2/16918
S8: Unidentified Flying Objects (UFOs): sightings; reports by members of public
S9: 1961-1963
S10: AIR 2/18115
S11: Unidentified flying objects: reports
S12: 1967
S13: AIR 2/18116
S14: Unidentified flying objects: reports
S15: 1967
S16: AIR 2/18117
S17: Unidentified flying objects: reports
S18: 1967-1968
S19: AIR 2/18183
S20: Unidentified flying objects
S21: 1968-1969
S22: AIR 2/18871

S23: Unidentified flying objects: reports and newspaper cuttings
S24: 1972

There are 69 textStrings (and therefore tagSet/textString pairs) in the *whole* document. In the 69 tagSets that occur in these pairs, there are only 12 *different* tagSets, because some of the tagSets are re-used in many different tagSet/textString pairs. The 12 *Distinct TagSets* are shown in Figure 4.

```
  T0: 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
  T1: 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0
  T2: 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0
  T3: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0
  T4: 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0
  T5: 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0
  T7: 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0
  T8: 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0
  T9: 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0
 T10: 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0
 T67: 0, 0, 0, 0, 0, 0, 0, 2, 3, 0, 0, 0, 3, 2, 2, 1, 1, 3, 2, 0, 0, 0, 0, 0
 T68: 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0
```

Figure 4. The Distinct TagSets in Figure 3's html document

Each row in that table represents a different tagSet. Each tagSet is a vector of 24 count values, for each of the 24 tag names found in the whole of the html document. The 24 tag names are: html, head, meta, /head, body, table, tbody, tr, td, p, b, /p, a, img, /a, font, /font, /td, /tr, /tbody, /table, /b, /body, /html.

```
T0:  0
T1:  1
T2:  2
T3:  3
T4:  4
T5:  5, 6
T7:  7
T8:  8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 50, 53, 56, 59, 62, 65
T9:  9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66
T10: 10,13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49, 52, 55, 58, 61, 64
T67: 67
T68: 68, 69
```

Figure 5. A tpGrid. It shows the itemSet associated with each Distinct TagSet in Fig 4

Each row is numbered with the number of the tagSet/textString pair in which the tagSet was first used in the html code. TagSet10 is the third row from the end. All tagSet/textString pairs from 11 to 66 (in the sequence of numbered pairs that represent the html code) must contain one of the tagSets already used, because the next tagSet/textString pair to contain a new tagSet is the one containing textString 67

An ***itemSet*** is the set of one or more tagSet/textString pairs associated with each distinct tagSet. If we represent each item by its number (in the sequence of numbered pairs) and display the itemSet associated with each of the 12 distinct tagSets shown in Figure 4, we obtain a ***tagSet Progression Grid*** (tpGrid) shown in Figure 5. This is the abstraction of the html document that we use to analyse page structure for repetitive features. It is much easier to see repetition in this grid structure than in the raw html code.

In order to avoid confusion it is important to distinguish the meaning of figures 4 and 5. Figure 4 shows all the Distinct TagSets in the html document. Figure 5, on the other hand, does not show any tagSets. Although the rows in Figure 5 are labelled with tagSet numbers, which *correspond to* those for the same rows in Figure 4, each row in Figure 5 shows the ***itemSet*** associated with each tagSet.

The *row cluster* T8, T9, T10 in Figure 5 reveals the position of the collection of 3-field result records in the web page. It also provides all the information an extractor program needs to accurately find and extract the data item for each field in each result record.

Figure 5 contains all the text items visible on the web page. Each text item is identified by its position number in sequential order. Rows T8, T9, T10 contain items 8 to 66. This cluster of long rows corresponds to the set of 3-field results in the web page. By constructing the tpGrid we have found the position of the results set in the web page, discovered that result records contain three fields, identified distinct tagSets that mark the start and end of the results section of the html document, and found other tagSets that uniquely label each field's data items so that accurate extraction of data items into correctly formatted records is ensured.

The way to discover and interpret patterns in a tpGrid (e.g Figure 5) is to follow the **TRAIL** of item numbers. If consecutive item numbers were linked with a line the result would be the Trail. It shows the order in which tagSets are used in the web page by following the sequence of numbered tagSet/textString pairs that correspond to the html document. *For example,* in Figure 5, tagSet T0 occurs only once in the whole document: before textString 0. Similarly we see that tagSets T1, T2, T3, T4, T7 and T67 each appear only once in the entire html document (preceding textStrings 1, 2, 3, 4, 7 and 67, respectively). TagSet T5, in contrast, has an itemSet containing two items: 5 and 6. This means that the two tagSet/textString pairs numbered 5 and 6 both contain the same tagSet, T5.

TagSets T8, T9 and T10 have large itemSets. A large itemSet is shown as a long row of integers. Each integer represents by number a numbered tagSet/textString pair that contains the same tagSet as all the other items in the row. TagSet T8 occurs before twenty textStrings, namely textStrings 8, 11, 14, 17, 20, 23, etc. Notice that the sequence of item numbers forms an arithmetic progression with common difference 3. This is because result records contain three fields. **The Trail** follows repeated vertical sequences of three items while it is inside the row cluster. TextStrings 8 to 66 on the web page use only tagSets 8,9 and 10, and the use of those three tagSets is ***cyclic***.

Each of the three fields in each record has a distinct tagSet to identify items belonging to that field. However, tagSet T8 does not identify first field items, as might be supposed. Instead, the row cluster representing a set of results has a characteristic structure in tpGrids, which is shown in Figure 6, where Trail entry and exit points are shown by arrows.

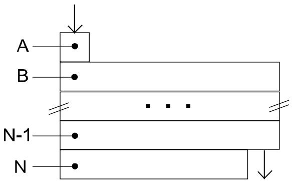


Figure 6. The characteristic structure of a result set row cluster in tpGrids.

Rows T7 to T10 in Figure 5 are an example. The tagSet before the first record (row **A** in figure 6) is different from the tagSet (row **N**) before each subsequent record in the result set. **A** is tagSet T7 and **N** is tagSet T10 in figure 5. So item 7 is the first field data for the first record and item 10 is the first field of the second record. The first field items of all the other records are in row T10. The reason for this structure is that the tagSet before the whole set of results uses different tags from that which just separates records within the collection of results.

3. The Data Extraction Algorithm

The tpGrid (Figure 5) provides the information needed by an extractor (wrapper) program to extract data from result pages from this web site. The extractor program proceeds as follows:

1. Search for tagSet T7 which occurs only once in the html document. It marks the start of the result set. TagSet T7 is immediately followed by the data item for the first field of the first record. Records have three fields.

2. TagSet T67 follows the last data item in the result set. It can be used either to isolate the results section from the html document, or just as a stopcode: stop extracting data items when tagSet T67 is encountered.

3. After the first field of the first record, a *field2* data item is preceded by tagSet T8; a *field3* data item is preceded by tagSet T9, and a *field1* item is preceded by tagSet T10. Each data item is, in effect, labelled in the html code by the tagSet that precedes it. In result sets whose field order can vary between records, this unique labelling is a significant benefit. And also when field values are missing from result records.

4. The extractor continues to extract data as long as the specified tagSets are found. Pages which contain no data (because the query produced no results) cause no problem because the tagSet labels for data items will not be present either. So the extractor recognizes an empty result set.

4. Rearranging tpGrids

The tpGrid representing a web page allows collections of result records to be identified as blocks of long rows (i.e. row clusters) with the structure shown in Figure 6. But sometimes items or whole rows in a tpGrid are *displaced* from their correct position in the row cluster to which they belong. A row is displaced if its tagSet happens to be used somewhere else in the web page, as well as in the part of the page containing the results. Individual items are displaced if an *insignificant* variation in their set of preceding tags gives them a different tagSet from other items in the same field position in records. Such displaced entities in the tpGrid can be automatically moved to their correct position (in order to reveal the repetition pattern). The process involves Trail-following and tagSet comparison, as now illustrated by means of an example.

An IEEE Search results web page is shown in Figure7 (from http://www.computer.org). Each result record is shown in a white rectangle and contains a fairly large number of potential field items. The tpGrid for this page, before repositioning displaced rows is shown in Figure8, and after rearrangement in Figure 9.

The ten result records visible in Figure 7 are shown in Figure 9 as the 10-column row cluster between rows T41 and T51. By following the Trail through this cluster we observe that each record has eleven fields, because of the eleven items in each column in the cluster of long rows.

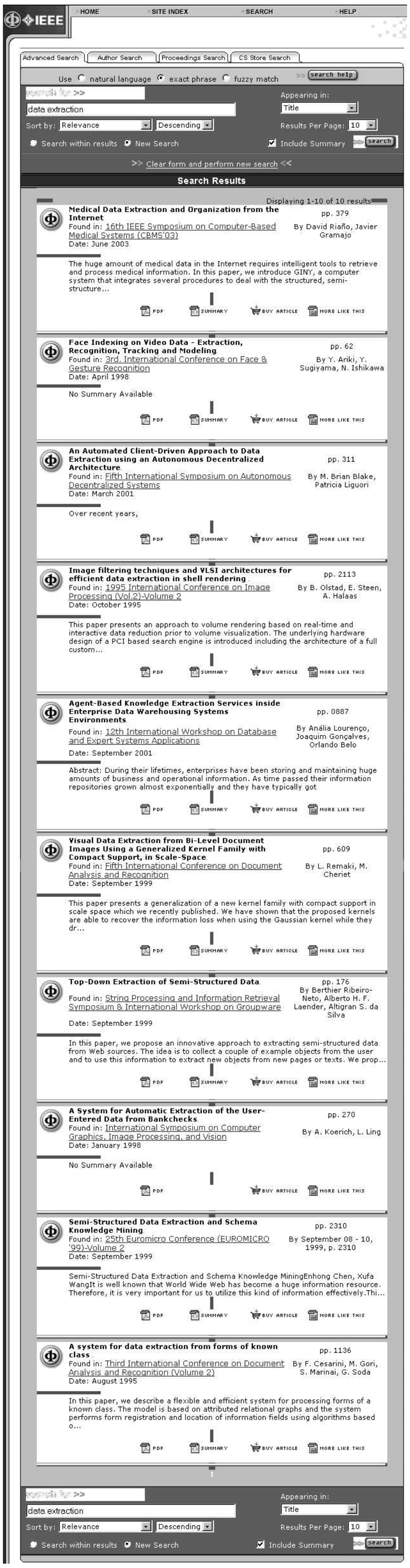
Item 127 is out of place. It should be in the space above it, in tagSet T50's row of items. The eleven fields in other columns associate a particular tagSet with each different field. So we expected item 127 to have tagSet T50, but instead it has new tagSet T127.

Comparing tagSets T50 and T127:

```
 50:0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,0,0,0,...,0,0,0
127:0,0,0,0,0,0,0,0,0,0,0,2,1,1,1,2,0,0,0,...,0,0,0
```

The 3 extra tags, absent from tagSet 127 are: a, img, /a, so there is an image hyperlink missing. This absence can be seen in Figure 7: in the third record (white rectangle) from the end, the usual row of four icons has only three. Missing images and hyperlinks are common variations found between records in result sets, so they are automatically recognised as *insignificant variation* between tagSets.

The BigBook standard web page from the RISE repository [11] produces a tpGrid that is structurally analogous to Figure 9, although the Web Page looks very different from Figure 7. Our analysis of the BigBook page structure can be seen at web site [10].



Use ○ natural language ● exact phrase ○ fuzzy match

data extraction

Appearing in: Title

Sort by: Relevance Descending

Results Per Page: 10

○ Search within results ● New Search

☑ Include Summary

Displaying 1-10 of 10 results

Medical Data Extraction and Organization from the Internet
Found in: 16th IEEE Symposium on Computer-Based Medical Systems (CBMS'03)
Date: June 2003
pp. 379
By David Riaño, Javier Gramajo
The huge amount of medical data in the Internet requires intelligent tools to retrieve and process medical information. In this paper, we introduce GINY, a computer system that integrates several procedures to deal with the structured, semi-structure...

Face Indexing on Video Data - Extraction, Recognition, Tracking and Modeling
Found in: 3rd. International Conference on Face & Gesture Recognition
Date: April 1998
pp. 62
By Y. Ariki, Y. Sugiyama, N. Ishikawa
No Summary Available

An Automated Client-Driven Approach to Data Extraction using an Autonomous Decentralized Architecture
Found in: Fifth International Symposium on Autonomous Decentralized Systems
Date: March 2001
pp. 311
By M. Brian Blake, Patricia Liguori
Over recent years,

Image filtering techniques and VLSI architectures for efficient data extraction in shell rendering
Found in: 1995 International Conference on Image Processing (Vol.2)-Volume 2
Date: October 1995
pp. 2113
By B. Olstad, E. Steen, A. Halaas
This paper presents an approach to volume rendering based on real-time and interactive data reduction prior to volume visualization. The underlying hardware design of a PCI based search engine is introduced including the architecture of a full custom...

Agent-Based Knowledge Extraction Services inside Enterprise Data Warehousing Systems Environments
Found in: 12th International Workshop on Database and Expert Systems Applications
Date: September 2001
pp. 0887
By Anália Lourenço, Joaquim Gonçalves, Orlando Belo
Abstract: During their lifetimes, enterprises have been storing and maintaining huge amounts of business and operational information. As time passed their information repositories grown almost exponentially and they have typically got

Visual Data Extraction from Bi-Level Document Images Using a Generalized Kernel Family with Compact Support, in Scale-Space
Found in: Fifth International Conference on Document Analysis and Recognition
Date: September 1999
pp. 609
By L. Remaki, M. Cheriet
This paper presents a generalization of a new kernel family with compact support in scale space which we recently published. We have shown that the proposed kernels are able to recover the information loss when using the Gaussian kernel while they dr...

Top-Down Extraction of Semi-Structured Data
Found in: String Processing and Information Retrieval Symposium & International Workshop on Groupware
Date: September 1999
pp. 176
By Berthier Ribeiro-Neto, Alberto H. F. Laender, Altigran S. da Silva
In this paper, we propose an innovative approach to extracting semi-structured data from Web sources. The idea is to collect a couple of example objects from the user and to use this information to extract new objects from new pages or texts. We prop...

A System for Automatic Extraction of the User-Entered Data from Bankchecks
Found in: International Symposium on Computer Graphics, Image Processing, and Vision
Date: January 1998
pp. 270
By A. Koerich, L. Ling
No Summary Available

Semi-Structured Data Extraction and Schema Knowledge Mining
Found in: 25th Euromicro Conference (EUROMICRO '99)-Volume 2
Date: September 1999
pp. 2310
By September 08 - 10, 1999, p. 2310
Semi-Structured Data Extraction and Schema Knowledge MiningEnhong Chen, Xufa WangIt is well known that World Wide Web has become a huge information resource. Therefore, it is very important for us to utilize this kind of information effectively.Thi...

A system for data extraction from forms of known class
Found in: Third International Conference on Document Analysis and Recognition (Volume 2)
Date: August 1995
pp. 1136
By F. Cesarini, M. Gori, S. Marinai, G. Soda
In this paper, we describe a flexible and efficient system for processing forms of a known class. The model is based on attributed relational graphs and the system performs form registration and location of information fields using algorithms based o...

data extraction

Appearing in: Title

Sort by: Relevance Descending

Results Per Page: 10

○ Search within results ● New Search

☑ Include Summary

Figure 7. IEEE Search results web page

```
T0: 0
T1: 1
T2: 2,3,4,  27,  173
T5: 5
T6: 6,  152
T7: 7,8,9,10,  13,14,15,16, 18,  153,154,155,156,  159,160,161,162,164
T11: 11,  157
T12: 12,  158
T17: 17,  163
T19: 19,  165
T20: 20,  166
T21: 21,22,23,24,25,  167,168,169,170,171
T26: 26,  172
T28: 28,  174
T29: 29
T30: 30
T31: 31
T32: 32
T33: 33
T34: 34
T35: 35
T36: 36
T37: 37,38,46,48,57,59,68,70,79,81,90,92,101,103,112,114,123,125,134,136,145,147
T39: 39
T40: 40
T41: 41,52,63,74,85,96,107,118,129,140
T42: 42,45,53,56,64,67,75,78,86,89,97,100,108,111,119,122,130,133,141,144
T43: 43,54,65,76,87,98,109,120,131,142,176
T44: 44,55,66,77,88,99,110,121,132,143

T47: 47,58,69,80,91,102,113,124,135,146

T49: 49,60,71,82,93,104,115,126,137,148
T50: 50,61,72,83,94,105,116,138,149
T51: 51,62,73,84,95,106,117,128,139
T127: 127
T150: 150
T151: 151
T175: 175
T177: 177
T178: 178
```

Figure 8. The tpGrid before rearrangement, for the web page shown in Figure 7

5. Terminology

This section defines some of the key terminology associated with the ideas introduced during the discussion of examples above.

A **Pair Sequence** is a numbered sequence of tagSet/textString pairs. Each tagSet summarises a corresponding tagString in the html document. The Pair Sequence for a web page is an indexed version of the html document representing that page. It contains all the information in the html code (including the final tagString that has no textString after it, but is followed instead by the end of the document/file). It allow Ti/Si pairs to be identified by number, and accessed by number when required. So the Pair Sequence is a lookup table, accessed by item number. The reason it contains all the html code in the html document is that each tagSet summarises a corresponding tagString, so each numbered pair can be seen as a tuple: [< tagString || tagSet >, textString] whose first field is a tagSet and its corresponding tagString. So information from the raw html at any point in the document can be accessed by position number if or when required. The page analysis process described in the current paper uses only the tagSet part of the embedded < tagString || tagSet > tuple.

Potential data items (PDIs) are the numbered items in the Pair Sequence. A basic goal of page analysis is to discover which of the PDIs contain data items. Data is contained in tag attributes that are hyperlinks to files, as well as the textString. The repetition of tagSets reveals collections of PDI records whose fields are each a [< tagString || tagSet >, textString] tuple, rather than just a textString. So a sequence of field

```
T0: 0
T1: 1
T2: 2,3,4,27,173
T5: 5
T6: 6,152
T7: 7,8,9,10,13,14,15,16,18,153,154,155,156,159,160,161,162,164
T11: 11,157
T12: 12,158
T17: 17,163
T19: 19,165
T20: 20,166
T21: 21,22,23,24,25,167,168,169,170,171
T26: 26,172
T28: 28,174
T29: 29
T30: 30
T31: 31
T32: 32
T33: 33
T34: 34
T35: 35
T36: 36
T37: 37,38
T39: 39
T40: 40
T41: 41,52,63,74,85, 96,107,118,129,140
T42: 42,53,64,75,86, 97,108,119,122,141
T43: 43,54,65,76,87, 98,109,120,131,142
T44: 44,55,66,77,88, 99,110,121,132,143
T42: 45,56,67,78,89,100,111,122,130,144
T37: 46,57,68,79,90,101,112,123,125,145
T47: 47,58,69,80,91,102,113,124,135,146
T37: 48,59,70,81,92,103,114,125,134,147
T49: 49,60,71,82,93,104,115,126,137,148
T50: 50,61,72,83,94,105,116,    138,149
T51: 51,62,73,84,95,106,117,128,139
T127:                       127
T150: 150
T151: 151
T175: 175
T177: 177
T178: 178
```

Figure 9. The tpGrid after moving displaced rows, for the web page shown in Figure 7

items may be identified in the tagString preceding each textString. Each single field in the discovered repetition pattern may thus be subsequently expanded into several fields (the extra fields are URLs extracted from tag attributes in the tagString of the PDI).

6. Automatic page format change detection

One of the key problems that has limited interest in using web data sources is their brittleness of access. A web site only has to change the layout of its results page and a wrapper will no longer extract data correctly.

A system is needed that *monitors* page format, so that any change is detected *before* a query to the site fails. This is easily achieved with tpGrids, because a grid is a fingerprint for page format. Therefore a query sent repeatedly to a web site will generate the same structures in the tpGrid for the results page. So by monitoring the grids from such test queries it is possible to detect page format changes as soon as they occur, and also to distinguish between changes that will affect data extraction and those involving other parts of the page. This fingerprint for page format is a valuable resource to ensure reliable data access.

7. Discussion and Experimental Evaluation

The page analysis technique has been applied to a large number of web site result pages in order to investigate its effectiveness. Examples of the range of different pages successfully wrapped by the page analyser can be seen at our web site. Our ultimate goal is fully automatic page analysis (and hence wrapper production) because this enables new applications for autonomous agents. But many of the current uses for wrappers (such as data integration systems, shopping agents and recommender systems, for example) probably *want* some minimal human involvement in order to provide confidence that completely accurate data extraction is achieved but also to specify the schema for the extracted data. Not all available fields are wanted. They could be extracted and then discarded or ignored later. Or there may be a wish to divide one textSring into more than one field during extraction. (This can alternatively be done later by processing the extracted database table).

If the page analysis process is used with human input to resolve ambiguities then it is effective at wrapping the majority of web pages seen so far. (We exclude, of course, plain text documents studied in NLP Information Extraction research. Our system looks for repetition patterns in html tags). Pages with few results provide limited repetition information. This problem can be solved by comparing the tpGrids for two or more web pages from the same web site (to find the results by distinguishing constant from variable page components).

Nested iterations were discussed briefly in [12]. Further examples are shown in our web site [10]. Pages whose data set has a tree structure have a tpGrid results section that is a nested version of Figure 6.

Our work is continuing by examining further web site results pages, to discover and resolve any problems. A comparison with previous work by processing the 'standard' web results pages in the RISE repository [11] will be published in due course. We are investigating the use of various knowledge sources in order to automate the process of resolving ambiguities. The use of repetition patterns of tagSets alone has been very effective at discovering page structure and identifying tagSet labels for data items, to be used by the extractor. But there is a lot of other information that can be used as well, to advance the aim of fully automatic wrapper production for *any* site. Many web sites have completely regular result sets (in terms of tagSet repetition) even though some fields may be missing. Such sites are automatically wrapped just by tpgrid analysis.

8. Related Work

Previously published work on the task of extracting data from *documents*, to create database tables, is divided into two separate research areas: Information Extraction and Wrapper Production. The former applies to plain text documents (rather than web pages) and uses Natural Language Processing techniques to obtain some understanding of the document in order to identify and extract substrings as data items. That is very different from Wrapper Production, which uses *the html framework* to divide up the document and recognise parts that are required data. This makes wrapper production *language independent* since the techniques apply to html rather than the language displayed on screen.

Previous work on wrapper production has included a lot of work using Machine Learning algorithms on html pages that have been previously annotated by hand to identify and describe the data items embedded in the document. This approach is clearly not an automatic technique, where a program finds the data and discovers its structure. Other researchers have tried to remedy that deficiency. Searching for repetition patterns has been tried in several ways. Repeating substrings in the html document have been sought, repeating subtrees in the html parse tree, and also repeating individual tags that recur at approximately equal intervals within the document character string. These methods experienced difficulties because of the ways they represented the document. In contrast, the tpGrid data structure represents a web page in terms of potential data objects and in a way that immediately reveals relevant features of page structure. This abstraction is a much easier thing to analyse than the document itself, and its item numbering system makes it a kind of index into parts of the original web page so that details can be obtained if required during analysis.

Another method [2] previously described for automatic wrapper production compared two or more query result pages from a web site. The purpose was to recognise constant parts that occur in all result pages from a site. Hence the other parts may contain data.

Domain-specific ontologies have also been used to support wrapper production techniques. These are specific to the kind of data to be extracted from web pages. The approach differs from other wrapper production strategies in considering the text of the data substrings rather than the html framework. It is thus language and domain dependent.

Our approach may compare web pages in order to distinguish between iterations that contain data and others that are part of the page structure. A simple ontology *for web result pages*, rather than data, is helpful to recognise parts of the web page such as the link to fetch the next page-full of query results.

9. Conclusions

Creating an extractor program for the result pages from a particular web site involves a page structure analysis process. An analyser program examines one or more pages from the web site and produces a form of page description to be used later by the extractor program on any query result pages obtained from that site.

The tpGrid is a useful object for analysis, supplementing information a program can obtain by directly examining the html code for the page. The grid finds data in the page, discovers its record structure and also identifies the tagSet 'labels' that an extractor program can use to recognise each type of data item to extract from other results pages from the same source.

The tpGrid is a fingerprint for the page structure of a particular web site. So it is easy to detect page format changes before they cause a wrapper to fail during extraction.

Since the method operates on repetition of items observed in the results page, it will not find data on pages that display few result items. For these, either a test query that generates more data is needed, or else the comparison of two or more results pages to distinguish between constant and variable parts of result pages.

The method of page analysis discussed in this paper is quick and simple, showing potential for use by 'robot' programs exploring the 'hidden web'. Techniques for automatic form-filling were discussed in [6].

10. References

[1] D. Buttler and L. Liu and C. Pu, *A Fully Automated Object Extraction System for the World Wide Web*. Proc. Intl. Conf. on Distributed Computing Systems, 2001. pp 361 - 371.

[2] V. Crescenzi, G. Mecca and P. Merialdo. *Roadrunner: Towards automatic data extraction from large web sites,* Proc 27th Very Large Databases Conference, VLDB'01, pages 109-118, 2001.

[3] David W. Embley, Y. S. Jiang, Yiu-Kai Ng, *Record-Boundary Discovery in Web Documents,* Proc. ACM SIGMOD Conference 1999, pages 467-478.

[4] Kushmerick, N. & Thomas, B. *Adaptive information extraction: Core technologies for information agents*. In Intelligent Information Agents R&D in Europe: An AgentLink perspective, Springer, 2003 (Klusch, Bergamaschi, Edwards & Petta, eds). LNCS 2586.

[5] Stephen W. Liddle, K. A. Hewett, and D. W. Embley, *An Integrated Ontology Development Environment for Data Extraction*, submitted, April 2003.

[6] S.W. Liddle, D.W. Embley, D.T. Scott, and S.H. Yau, *Extracting Data Behind Web Forms*, Proceedings of the Workshop on Conceptual Modeling Approaches for e-Business, Tampere, 2002.

[7] I. Muslea and S. Minton and G. Knoblock. A Hierarchical Approach to Wrapper Induction. Proc 3rd Conf on Autonomous Agents (1999).

[8] W.W. Cohen, W. Fan, *Learning Page-Independent Heuristics for Extracting Data from Web Pages*, Proc 8th International World Wide Web Conference, 1999.

[9] Stephen Soderland, *Learning Information Extraction Rules for Semi-structured and Free Text,* Machine Learning 34(1-3) pp 233-272, 1999.

[10] A substantial bibliography of relevant references can be found via the author's web site at http://www.page-info.info

[11] The RISE Repository of Online Information Sources Used in Information Extraction Tasks
http://www.isi.edu/info-agents/RISE/index.html

[12] J. Robinson, *Data Extraction from Web Database Query Result Pages via TagSets and Integer Sequences*, Proc IADIS WWW/Internet International Conference 2003.