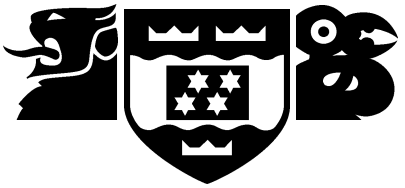
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematical and Computing Sciences
Computer Science

Learning Information Extraction Patterns from Tabular Web Pages without Manual Labelling

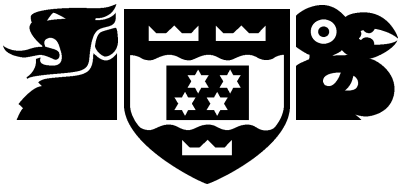
Xiaoying Gao, Mengjie Zhang, Peter Andreae

Technical Report CS-TR-03/3
March 2003

School of Mathematical and Computing Sciences
Victoria University
PO Box 600, Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

Learning Information Extraction Patterns from Tabular Web Pages without Manual Labelling

Xiaoying Gao, Mengjie Zhang, Peter Andreae

Technical Report CS-TR-03/3
March 2003

Abstract

This paper describes a domain independent approach for automatically constructing information extraction patterns for semi-structured web pages. Given a randomly chosen page from a web site of similarly structured pages, the system identifies a region of the page that has a regular "tabular" structure, and then infers an extraction pattern that will match the "rows" of the region and identify the data elements. The approach was tested on three corpora containing a series of tabular web sites from different domains and achieved a success rate of at least 80%. A significant strength of the system is that it can infer extraction patterns from a single training page and does not require any manual labeling of the training page.

Keywords Information extraction; machine learning; wrapper; semi-structured data; automatic pattern generation.

Author Information

The authors are academic staff members in computer science, Victoria University of Wellington, New Zealand.

Learning Information Extraction Patterns from Tabular Web Pages without Manual Labelling

Xiaoying Gao, Mengjie Zhang, Peter Andreae

School of Mathematical and Computing Sciences
Victoria University of Wellington
P.O. Box 600, New Zealand
{*xgao,mengjie,pondy*}*@mcs.vuw.ac.nz*

Abstract

This paper describes a domain independent approach for automatically constructing information extraction patterns for semi-structured web pages. Given a randomly chosen page from a web site of similarly structured pages, the system identifies a region of the page that has a regular "tabular" structure, and then infers an extraction pattern that will match the "rows" of the region and identify the data elements. The approach was tested on three corpora containing a series of tabular web sites from different domains and achieved a success rate of at least 80%. A significant strength of the system is that it can infer extraction patterns from a single training page and does not require any manual labeling of the training page.

Keywords: Information extraction; machine learning; wrapper; semi-structured data; automatic pattern generation.

1 Introduction

Since the late 1990s, the World Wide Web (WWW) has been serving a huge, widely distributed, global information service centre for news, advertisements, consumer information, financial management, education, government, e-commerce, and many other information services. Almost all this information is presented for "human reading" only, as text in natural language, marked up using HTML or other markup languages. The primary tools for searching for information on the WWW are therefore based on text-based searches.

However, a significant fraction of the information on the WWW is quite structured and could readily be stored in traditional databases which would support more sophisticated search techniques taking advantage of the highly structured nature of the information. In fact, many web pages are constructed dynamically from databases, but the underlying structure is frequently hidden in the formatting for human readability. There are many domains where it would be useful to extract this structured information from many different web pages and make it available in a database through specialised search engines that can take advantage of the structure to support complex search and query facilities. This not only supports more powerful search by human users, but also automated search for knowledge-based information retrieval, information agents, knowledge discovery, and data mining applications.

There are many information extraction techniques that can "pull out" the elements of the structured information from web pages and construct databases of the information[4, 6, 7, 8, 11, 12, 15, 17]. These information extraction systems for online documents are often called *wrappers*. Ideally, these wrappers would constitute autonomous information extraction agents that could independently mine the web for information and construct databases to support complex search and analysis.

Most existing wrappers are based on techniques that match information extraction patterns or templates to a web page, and use the patterns to identify the significant data elements. The web pages need to be at least semi-structured so that all the relevant pages from the site share at least similar formatting. Typically, a single pattern will work for many pages from the same site, but combining information from pages on many sites may require different patterns for each site. The bottleneck for developing such wrappers is the construction (and maintenance) of the patterns.

Constructing patterns by hand is tedious and error prone, and it is often not feasible to keep up with the dynamism of the Internet — new sites must constantly be incorporated, and existing sites may often change their format to improve their services to human readers which will mean that existing patterns must be modified to deal with the updated format. An alternative approach is to use machine learn-

ing technology to generate patterns automatically. Typically, such systems use some example pages from a site to infer a pattern that will work on all pages from that site. These techniques allow patterns to be generated and maintained more efficiently than with manual pattern constructions. However, most of the techniques require that the example pages are manually labelled and tagged to identify the target elements on the example pages. Although this is much simpler than constructing patterns by hand, it means that the process still requires non-trivial human intervention, which limits the autonomy of an information extraction agent.

The goal of our research is to develop domain independent techniques that can autonomously construct information extraction patterns for semi-structured web pages without requiring manual labelling or tagging. This paper describes one such technique [5] that works on a class of web pages that can be characterised as having a tabular format. The technique requires uses a single example page, and requires no labeling or tagging.

We have evaluated the technique on a variety of web sites, and also compared its effectiveness with an existing set of algorithms[10] that build patterns from a set of web pages that have been manually labeled and tagged. For web pages that have a tabular format, we show that our algorithm is more effective than Kushmerick's system, and much more effective than any of his individual algorithms, even though it uses unlabelled training examples. On the sites with non-tabular web pages, our algorithm performs much less well.

The paper is organised as follows. Section 2 describes what we mean by semi-structured web pages and introduces the task of constructing information extraction patterns for such pages. Section 3 outlines the language we have developed for representing patterns and section 4 describes our algorithm for building patterns from unlabelled, untagged web pages. Section 5 summarised our evaluation experiments, and section 6 presents our conclusions and future work.

2 Background

There is a spectrum of kinds of information on the WWW. Much of the information on the web is presented as free text; automatically extracting information from these pages would require natural language processing. Some information is fully structured in some kind of database format, which makes automatic extraction and analysis of the information straightforward. There is a growing use of XML to provide fully structured, machine interpretable data of this form. However, between these two extremes, there is much information that is presented in a semi-structured form, where the information has a fairly regular and identifiable structure, but is not as precise, complete, or formal as the fully structured pages. Some examples are online classified advertisements, online product catalogs, and telephone books. The information typically consists of a collection of entities, each described by a set of "fields". The information is usually laid out on the web page in regular way to make it easy for a human reader to parse the information.

Many of these pages are generated using predefined templates, either generated by hand (for example by using form-based page editors) or are automatically generated by local search engines. Commonly, these pages use HTML markup to present the information in the form of tables or lists so that a reader can easily recognise the corresponding elements of the information. For effective information extraction, we must construct patterns that can also parse the files and identify the entities and fields.

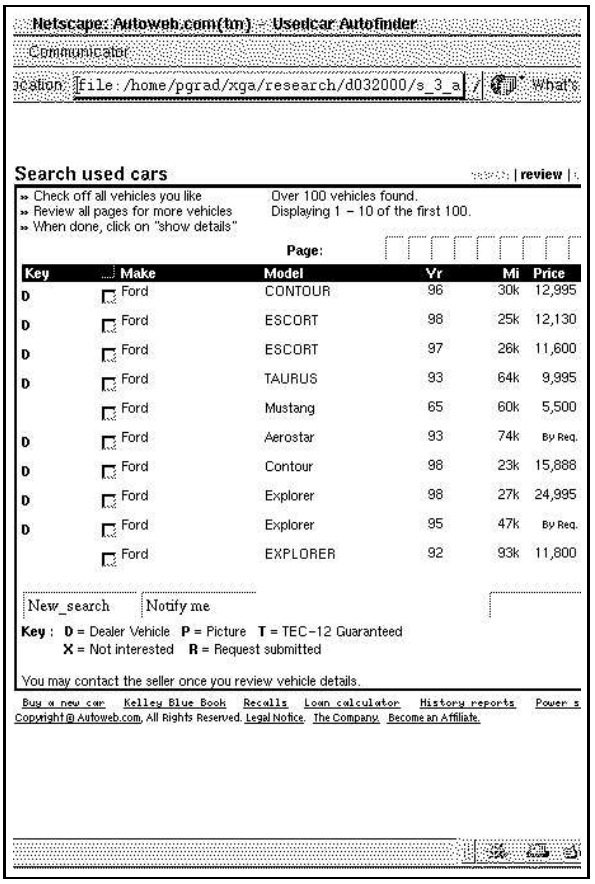
Figure 1 shows a piece of a sample web page for car advertisements and a fragment of its source file. For this web page, an information extraction system might need to extract the values of the attributes "Key","Make", "Model", "Year", "Miles", and "Price" for each car advertisement, for example, D (Dealer), Ford, CONTOUR, 96, 30k, and $12,995. We refer to the values of such kinds of attributes *knowledge units* in this paper. An information extraction pattern to match this kind of page must identify the pieces of text constituting the knowledge units, and also the elements of the HTML structure that separate the knowledge units. An example pattern is shown in Figure 2, using the pattern language described in the next section. The task of the pattern building algorithm is to infer such a pattern from the source file of a Web page.

Semi-structured data, by definition, is presented in a relatively regular format. Our algorithm is based on the premise that by identifying the regularities in a page, the structure can be automatically detected, and a pattern can be constructed. There are two kinds of regularities that the algorithm can exploit:

- Regularities in the markup: For example, tables in HTML contain rows that are marked by `TR` tags, and contain cells that are marked with `TD` or `TH` tags.

- Text similarities: The text in the corresponding knowledge units of different entities are often similar. For example, "prices" often start with a "$" and are followed by 2 to 8 numbers, "car makes" and "car models" are often a single word with 3 to 12 characters, and the first letter is often capitalised.

We focus on pages that have a tabular structure in which the information is structured in the form of a sequence of entity descriptions, where each entity is described by a sequence of values. Pages containing HTML tables in which each row describes an entity are one example of pages with tabular structure, but other formatting markup, such



(a)

```
<tr> <td bgcolor="#f1f1f1" width="60">
<font color="#cc0000" face="arial, helvetica" size="-1">
<b>D</b>
  
  
</font>
<font face="arial, helvetica" size="-1">
</font>
 
</td>
<td bgcolor="#f1f1f1" align="right">
<input type="Checkbox" name="986169" value="1" >
</td>
<td valign=top align=left bgcolor="#f1f1f1" width="110">
<font size=-1 face="arial, helvetica" color="#000000">Ford</font>
</td>
<td valign=top align=left bgcolor="#f1f1f1" width="120">
<font size=-1 face="arial, helvetica" color="#000000">CONTOUR</font>
</td>
<td valign=top align=center bgcolor="#f1f1f1" width="30">
<font size=-1 face="arial, helvetica" color="#000000">96</font></td>
<td valign=top align=right bgcolor="#f1f1f1" width="55">
<font size=-1 face="arial, helvetica" color="#000000">
30k
</font></td>
<td valign=top align=right bgcolor="#f1f1f1" width="45">
<font size="-1" face="arial, helvetica" color="#000000">
$12,995
</font>
</td>
</tr>
. . . . . .
<tr>
<td bgcolor="#ffffee" width="60">
<font color="#cc0000" face="arial, helvetica" size="-1">
<b>P</b>
. . . . . .
<font size="-1" face="arial, helvetica" color="#000000">
12,130
</font>
</td>
</tr>
. . . . . .
```

(b)

Figure 1. (a) A sample piece of web page; (b) A fragment of the source file of the page.

as nested lists, or sections and paragraphs may also be used to construct tabular structured pages.

The key idea of the algorithm is to find repeated components ("rows") of the page that match each other well, and construct a generalised pattern that matches all the rows, and picks out the corresponding textual elements of each row that are assumed to be the desired knowledge units.

Although semi-structured pages, especially tabular pages, have regularities, the regularities are often not precise or consistent, so that constructing an effective information extraction pattern is not trivial. For example,

- Most browsers tolerate missing closing tags in many situations, so that a page may have (say) `</TR>` and `</TD>` tags in some parts of the table, but not have them in others.

- Because the pages are intended for human reading, it is quite feasible for rows of a table (or lists in a paragraph) to have different numbers of cells. Where knowledge units are missing for some entities, a page may use an empty value such as ` ` in some places, and a completely missing cell or item in another.

- The knowledge units might be presented in different forms in different places.

The example page fragment below illustrates some of these irregularities:

```
Row 1: <TR>
       <TD>Ford<TD>Telstar</TD>
       <TD><FONT color=red>2001</FONT></TD>
       <TD>$5k</TD>
Row 2: <TR BGCOLOR=\#CCCCC>
       <TD>Toyota</TD><TD>Camry</TD>
       <TH Align=Left>$20,000<br>o.n.o.</TD></TR>
```

Notice the missing `</TD>` and `</TR>` in the first row, the missing year value and a mistake using `<TH>` instead of `<TD>` in the second row, the different markup of the price fields, and the different form of the text of the prices.

The follow sections describe our solution to the problem of identifying the regularities without being confused by the irregularities.

```
tag(any, any, any)
tag(word, b, any)
tag(word, b, any)
opt(tag(any, any, any))
text(any, any, any))
opt(tag(any, any, any))
text(any, any, any)
text(any, any, any)
tag(word, e, ”</font>”)
tag(word, b, ”<font face= “arial, helvetica” size= “-1”>”)
tag(word, e, ”</font>”)
text(any, any, any)
tag(word, e, ”</td>”)
tag(word, b, any)
tag(word, b, any)
tag(word, e, ”</td>”)
tag(word, b, any)
tag(word, b, ”<font size=-1 face=“arial, helvetica” color=“#000000”>”)
text(any, any, any))
tag(word, e, ”</font>”)
tag(word, e, ”</td>”)
tag(word, b, any)
tag(word, b, ”<font size=-1 face=“arial, helvetica” color=“#000000”>”).
text(any, any, any)
tag(word, e, ”</font>”)
tag(word, e, ”</td>”)
tag(word, b, any)
tag(word, b, ”<font size=-1 face=“arial, helvetica” color=“#000000”>”)
text(any, any, any)
tag(word, e, ”</font>”)
tag(word, e, ”</td>”)
tag(word, b, any)
tag(word, b, ”<font size=-1 face=“arial, helvetica” color=“#000000”>”)
text(any, any, any)
tag(word, e, ”</font>”)
tag(word, e, ”</td>”)
tag(word, b, any)
tag(word, b, ”<font size=-1 face=“arial, helvetica” color=“#000000”>”)
opt(tag(any, any, any))
opt(text(any, any, any))
alt([tag(any, any, any), text(any, any, any)])
opt(tag(any, any, any))
opt(tag(any, any, any))
tag(line, e, ”</tr>”).
```

Figure 2. An example pattern.

3 Pattern Language and Representation

The first requirement for building patterns is a language for expressing the patterns. This language must be rich enough to express the generalisations that capture the regularities in a semi-structured web page, but constrained enough to enable ready inference of the patterns from a single example page.

Our focus on HTML documents with a tabular structure informs the design of the language. We assume that the part of a page containing the knowledge units consists of a sequence of rows, where each row contains the information about a single entity. The target pattern must match each row in the page. We further asssume that the knowledge units in a row are contained in text strings that are separated by formatting tags that provide the structure, so that the pattern must distinguish the knowledge units from the structuring tags. The pattern must cover all the variations in both the knowledge units and the structuring tags from row to row.

Given these assumptions, we can represent a web page as a sequence of (concrete) *tokens*, where each token is either an HTML tag (a *concrete tag token*) or a string of the characters between HTML tags (a *concrete text token*). A pattern for a set of web pages must match against each subsequence of concrete tokens that corresponds to a row.

3.1 Pattern Language

The basic unit of the pattern language is an *abstract token*. Each abstract token is a generalisation of tokens in the page. *Generalised text tokens* are generalisations of knowledge units; *generalised tag tokens* are generalisations of HTML tags forming the structure of the row. An abstract token may be a simple generalised token, an *optional token* (for matching row elements that may be missing), or a *disjunctive token* (`alt`, for matching more complex row elements that cannot be represented by single tokens). A pattern consists of a sequence of abstract tokens. Figure 3 gives a formal specification of the syntax of the pattern language.

Pattern ::=	[*AbstractToken*, *AbstractToken*, . . . *AbstractToken*]
AbstractToken ::=	*GeneralisedToken* OR
	opt(*GeneralisedToken*) OR
	alt(*GeneralisedToken*,. . . , *GeneralisedToken*)
GeneralisedToken ::=	*Head*(*Parameter*, *Parameter*, *Parameter*)

Figure 3. Syntax of the Pattern language.

The four parts of a generalised token capture the properties that a page element must have in order to match the token. The Head of a generalised token specifies whether it is a tag token or a text token. The three parameters of a token capture different aspects of the generalisation of the elements the token will match. The semantics of the three parameters are different for the two categories of tokens. For convenience, we will use *tag token* and *text token* to refer to the generalised tokens only.

3.1.1 Representation of Tag Tokens

The first parameter of a tag token captures the role or *level* of the tag. Different HTML tags play different kinds of roles. Some of them specify the appearance of individual words, for example, to change the font and size. Others specify the structure of larger level units, for example, `<br>` separates text into lines and `<p>` separates text into paragraphs. To represent generalised tokens that may match different HTML tags, we have found it useful to capture the different kinds of roles in terms of the *level* of the page structure at which the tag operates. We identifed four useful levels: *page*, *paragraph*, *line* and *word*. Most HTML tags operate on one of these levels. Some tags have more than one role and can be used in different levels, in which case we classify the tag according to its most commonly used

role. For example, <H2> is most often used to begin a major section of an HTML document and is therefore at the *page* level, even though it also can be used (incorrectly) to emphasise a lower level element such as a cell in a table. Tags that do not fit any of the four levels are classified as *other*.

The second parameter of a tag token captures whether the tag is an opening or closing tag. Most HTML tags appear as pairs of an opening and closing tag, for example, <B> and </B>. The *open* or *close* state is extracted as an important feature for guiding information extraction.

The third parameter of a tag token consists of the string of characters of one of the tags from which the tag token was generalised.

The syntax for tag tokens is shown in Figure 4.

TagToken ::=	tag(*level*, *open/close*, *tag_string*):
level ::=	page | para | line | word | other
open/close ::=	o | c
tag_string ::=	character string

Figure 4. Syntax of Tag Tokens

3.1.2 Representation of Text Tokens

HTML tags are from a well defined, constrained language and therefore using categories of tags to represent the generalisations is feasible. The text strings that will constitute the extracted knowledge units are not from a well defined language, so we need an alternative approach for the generalised text tokens that represent generalisations of concrete text tokens. We also want the system to be domain independent, so we cannot build in any properties of text strings that are specific to knowledge units from particular domains.

We use two domain independent properties of text strings to capture some of the generalisations. The first parameter of an generalised text token is the *length* of the text string — the number of characters in the string. The second parameter (*format*) is the sequence of character categories in the string. We distinguish several categories of characters: uppercase alphabetic, lowercase alphabetic, digits, and punctuation. Each category in a sequence of categories represents any number of adjacent characters of the given category. Any character that is not in one of the four categories, such as $, is represented by itself in the format. For example, the format for "$5000 Australian" is "$npcl", where "n" represents digits, "p" represents punctuation, "c" represents capital letters, and "l" represents lower case letters.

As in the tag tokens, the third parameter of a text token is the the string of characters of one of the text strings from which the token was generalised. The syntax of text tokens is summarised in Figure 5.

TextToken ::=	text (*length*, *format*, *text_string*)
length ::=	number
format ::=	*CharCategory**
CharCategory ::=	c | l | n | p | *character*
text_string ::=	character string

Figure 5. Syntax of Text Tokens.

3.2 Wild Cards

In both tag tokens and text tokens, any of the three parameters can have the value any, which acts as a "wild card" that matches any value, allowing larger generalisations. For example, the tag token tag(line, any, any) will match any tag that labels a *line* page structure such as <tr>, </tr>
; the tag token tag(any, any, any) will match any HTML tag at all. The text token text(any, ''c'', any) would match any capitalised word of any length.

3.3 Optional Tokens and Disjunctive Tokens

Although generalised tokens with wild cards can capture many generalisations of web page elements, some web pages contain some variations that cannot be represented. We therefore add two further kinds of tokens:

- *optional tokens*,
  represented by opt(⟨*GeneralisedToken*⟩),
  to represent tokens that may or may not be present in the page, and
- *disjunctive tokens*,
  represented by alt(⟨*GeneralisedToken*⟩, ⟨*GeneralisedToken*⟩, . . . , ⟨*GeneralisedToken*⟩),
  to represent disjunctive generalisations to handle mismatch tokens (including incorrectly written tokens) by giving a list of interchangeable tokens.

4 Learning Patterns from HTML Source Files

Given a representation language for expressing patterns, the second requirement for building patterns is an algorithm that will infer patterns from example pages. Traditional machine learning algorithms for inferring patterns require a set of training examples that have been marked up or labeled, usually by hand; our goal is to infer a pattern from a single unlabeled example page. Our approach exploits the repetitive structure of the pages to automatically identify sets of

corresponding elements that can then be generalised to obtain a generic pattern.

We have developed a two-stage, domain-independent, pattern learning algorithm. The first stage is a regular structure detection algorithm based on the Smith-Waterman algorithm [16]. The second stage is a pattern induction algorithm that generalises sets of elements identified in the first stage. This section briefly outlines the Smith-Waterman algorithm, then describes our two new algorithms.

4.1 Smith-Waterman Algorithm

The Smith-Waterman algorithm [16] was designed to identify maximally similar subsequences in two linear molecular structures. The algorithm works on any sequences of units from a finite alphabet, given a measure of the similarity of the units. The algorithm can handle both missing units and mismatching units. For example, given

S_1 = "A-B-A-U-G-C-C-A-U-U-G-A-C-G-G"
S_2 = "C-D-A-G-C-C-U-C-G-C-U-U-A-G"

The algorithm will identify the following maximally similar subsegments:

"-G-C-C-A-U-U-G-"
"-G-C-C - U-C-G-"

Note that this pair contains both an internal deletion (of A in S_1) and a mismatch (U to C), as well as dropping the beginnings and ends of the original sequences. The algorithm uses different penalties for different kinds of mismatch and dynamic programming to build the match with the optimal score.

Given two molecular sequences $A = a_1 a_2 \ldots a_n$ and $B = b_1 b_2 \ldots b_m$, this algorithm finds the maximum similarity of a pair of subsequences. The algorithm builds a matrix H in which H_{ij} is the maximum similarity of two subsequences ending in a_i and b_j. The algorithm computes the H_{ij} as follows:

For $0 \leq i \leq n$ and $0 \leq j \leq m$, set $H_{i,0} = H_{0,j} = 0$
For i and j increasing from 1 to n and m

$$H_{i,j} = \max\{H_{i-1,j-1} + s(a_i, b_j),$$
$$\max_{k \geq 1}\{H_{i-k,j} - W_k\},$$
$$\max_{l \geq 1}\{H_{i,j-l} - W_l\},\ 0\}$$

where $s(a_i, b_j)$ is the similarity comparison function which specifies the degree of similarity between sequence element a_i and b_j: $s(a_i, b_j) = 1$ if a_i matches b_j and $s(a_i, b_j) = -1/3$ if they do not match. $W_k = 1.0 + 1/3 * k$ is the weight for deletion of length k. The largest value in the matrix ($Max\{H_{ij}\}$) represents the ends of the best matching subsequences, and the subsequences are found by tracing back to an element where $H_{ij} = 0$.

4.2 Regular Structure Detection Algorithm

Finding all the rows on a semi-structured web page involves identifying similar subsequences of tokens from the page. We developed a *regular structure detection algorithm* that identifies a region of regular structure on a page consisting of contiguous subsequences of tokens on a page that have similar structure. Each subsequence is assumed to be a row describing a single entity. One component of the algorithm is based on the Smith-Waterman algorithm.

The algorithm first uses a set of heuristics to find all plausible starts and ends of rows, and forms a set of candidate rows from the subsequences of tokens between all the start and end points. It then passes all adjacent pairs of candidate rows to a row similarity algorithm to find rows with very similar structure. When the similarity algorithm considers that two rows are sufficiently similar, it outputs a list of pairs of the corresponding tokens from the two rows. The regular structure detection algorithm then finds the largest sequence of successive pairs of matching rows, and collects all the rows from this region. It then forms lists of all the corresponding tokens of these rows to pass to the *pattern induction algorithm*.

4.2.1 Row Similarity Algorithm

The first step of the row similarity algorithm is to use some simple similarity criteria to rule out pairs of candidate rows that are clearly dissimilar. Pairs of rows that satisfy these criteria are then matched with our version of the Smith-Waterman algorithm to identify the best matching subsequences of each row. The critical modification to the Smith-Waterman algorithm is that the simple unit similarity function described above is replaced by the more complex *token similarity function* described below. Having found the best matching subsequences, the similarity algorithm applies two further row similarity criteria to determine whether the two candidate rows are sufficiently similar to be considered genuine rows. The measures of row similarity are described in section 4.2.3.

An example of the input and output of the algorithm is given in Figure 6 for two rows that are considered sufficiently similar. If the two rows were not sufficiently similar, then no list of token pairs would be produced. The tokens are indicated by angle brackets: ⟨ ⟩.

4.2.2 Token Similarity Function

The Smith-Waterman algorithm requires a similarity function that has the value 1.0 for identical elements and some negative value for completely dissimilar elements. For many domains, there will be no similarity values other than the two extreme values. On the other hand, in our web

Input:
row1: [⟨<TR>⟩⟨<TD>⟩⟨Ford⟩⟨</TD>⟩⟨<TD>⟩⟨Telstar⟩
⟨</TD>⟩⟨<TD>⟩⟨$5000⟩⟨</TD>⟩⟨</TR>⟩]
row2: [⟨<TR BGCOLOR=yellow>⟩⟨<TD>⟩⟨Toyota⟩⟨<TD>⟩⟨Camry⟩
⟨</TD>⟩⟨<TH>⟩⟨$9000⟩⟨</TD>⟩⟨</TR>⟩]

Output:
The two strings are suffi ciently similar
The associated token pairs:
(⟨<TR>⟩, ⟨<TR BGCOLOR=yellow>⟩),
(⟨<TD>⟩, ⟨<TD>⟩),
(⟨*Ford* ⟩, ⟨*Toyota*⟩),
(⟨</TD>⟩, missing unit),
(⟨<TD>⟩, ⟨<TD>⟩),
(⟨Telstar⟩, ⟨Camry⟩),
(⟨</TD>⟩, ⟨</TD>⟩),
(⟨<TD>⟩, ⟨<TH>⟩),
(⟨$5000⟩, ⟨$9000⟩),
(⟨</TD>⟩, ⟨</TD>⟩),
(⟨</TR>⟩, ⟨</TR>⟩).

Figure 6. Example output of row similarity algorithm

page domain, two tokens may be significantly similar without being identical, so that intermediate values will be important. For example, one can argue that the token ⟨<TD align=center>⟩ has some similarity to all of the following tokens, but has less similarity to the tokens further down the list:

⟨<td align=center>⟩
⟨<font size=-1>⟩
⟨
⟩
⟨</p>⟩
⟨*"Telstar"*⟩

There is no single similarity measure for character strings or HTML tags, and it is not justifiable to create a fine-grained similarity measure on an *ad hoc* basis. We have therefore chosen to use a coarse-grained similarity measure with just five possible values. Two identical tokens have a similarity value of 1.0; two tokens of different types are considered completely dissimilar and have a measure of -1/3, as in the standard Smith-Waterman measure. For tokens that are of the same type but are not identical, we compute and compare the two features that we also use to represent generalised tokens — *level* and *open/close* for tag tokens, and *length* and *format* for text tokens. The similarity measure of the tokens depends on the number of features that match. Our token similarity function is therefore:

$$S(a,b) = \begin{cases} 1.0 & \text{if a and b are identical} \\ 0.5 & \text{if tokens are the same type and both features match} \\ 0.3 & \text{if tokens are the same type and just one feature matches} \\ 0.0 & \text{if tokens are the same type and neither feature matches} \\ -\frac{1}{3} & \text{if tokens are not the same type} \end{cases}$$

For tag tokens, the features are considered to match only if they are identical. For text tokens, the formats are considered to match if one is a subset of the other, and the lengths are considered to match if they are "close enough":

$$|(l_a - l_b)/(l_a + l_b + 1)| < 0.2$$

We selected the numeric values of the three intermediate similarity values on the basis of performance of the measure in a set of experiments.

4.2.3 Row Similarity Criteria

To determine whether two candidate rows are sufficiently similar to be the basis of an extraction pattern, the row similarity algorithm applies five different row similarity criteria measures. The first three criteria are independent of the correspondence between tokens in the two rows, so they can be computed before the Smith-Waterman algorithm is run. If the rows are insufficiently similar by any of these three criteria, the row detection algorithm will exit immediately, hence avoiding the expense of the Smith-Waterman algorithm. The other two criteria involve the best matching subsequences of the rows found by the Smith-Waterman algorithm.

The first three criteria require that the lengths of the two candidate rows (measured in different ways) differ by less than a threshold. To handle rows of varying lengths, the differences are scaled — if l_1 and l_2 are the lengths of the rows, then the criteria require that $\frac{l_1 - l_2}{l_1 + l_2 - 1} < \theta$. The three lengths are

- the number of tokens in the row,
- the number of text tokens in row,
- the total number of characters in the text tokens in the row.

These criteria focus particularly on the text tokens in the rows, since they represent the knowledge units to be extracted from the page.

The last two criteria, which must both be satisfied, use the best matching subsequences found by the Smith-Waterman algorithm. One criterion requires that the matching subsequences constitute the major part of the candidate rows. If *N* is the average of the number of tokens in the two candidate rows and *n* is the average of the number of tokens in the two matching subsequences, then the criterion requires that $\frac{N-n}{N+n-1} < \theta$. The final criterion requires that the similarity measure h for the two subsequences computed by the Smith-Waterman algorithm is close to the length, l, of the shorter of the rows (the maximum possible similarity measure): $\frac{l-h}{l+h-1} < \theta$.

4.3 The Pattern Induction Algorithm

Once the regular structure detection algorithm locates the regular data area and finds a set of similar segments from a web page, the pattern induction algorithm must generate a generalised pattern that will match all the segments.

The induction algorithm proceeds in three steps:

1. Collect the associated tokens from all segments, and save them into a set of associated token lists.
2. Generate a generic token for each associated token list. Given an associated token list $[t_1, t_2, ..., t_n]$,
   - Initialise $t_g = t_1$
   - For each $t_i, 2 \leq i \leq n$, replace t_g with $g(t_g, t_i)$, which is the next more general token of t_g and t_i.
   - Output t_g, which is the generic token.
3. The sequence of all generic tokens forms the generic pattern.

The main task here is to generate the *next more general token* in step 2. If we use t_a and t_b to represent two associated tokens, then the following rules are used for defining the *next more general token* $g(t_a, t_b)$:

1. For any token, if $t_a = t_b$, then $g(t_a, t_b) = t_a$.
2. For the tag, text and missing tokens, let $t_a = h_a(x_a, y_a, z_a)$, $t_b = h_b(x_b, y_b, z_b)$, t_ϵ be a missing token,
   (a) If $h_a = h_b \in \{tag, text\}$, then $g(t_a, t_b) = h_a(x, y, z)$, where:
   $x = x_a$ if $x_a = x_b$; $x =$ "any" if $x_a \neq x_b$
   $y = y_a$ if $y_a = y_b$; $y =$ "any" if $y_a \neq y_b$
   $z = z_a$ if $z_a = z_b$; $z =$ "any" if $z_a \neq z_b$

   (b) If $h_a \neq h_b$ and $h_a \in \{tag, text\}, h_b \in \{tag, text\}$, then
   $g(t_a, t_b) = alt([h_a(any, any, any), h_b(any, any, any)])$

   (c) If there is one missing token, then
   $g(t_a, t_\epsilon) = opt(h_a(any, any, any))$
   $g(t_\epsilon, t_b) = opt(h_b(any, any, any))$

   (d) In the special case of two missing tokens, we define $g(t_\epsilon, t_\epsilon) = t_\epsilon$
3. For the generated optional and disjunctive tokens, let $t_a = h_a(any, any, any)$, $t_b = h_b(any, any, any)$, where $h_a \neq h_b$, we define
   $g(opt(t_a), t_a) = opt(t_a)$
   $g(opt(t_a), t_b) = alt([t_a, t_b, t_\epsilon])$
   $g(opt(t_a), t_\epsilon) = opt([t_a])$.
   $g(alt([t_a, t_b]), t_a) = alt([t_a, t_b])$
   $g(alt([t_a, t_b]), t_b) = alt([t_a, t_b])$
   $g(alt([t_a, t_b]), t_\epsilon) = alt([t_a, t_b, t_\epsilon])$
   $g(alt([t_a, t_b, t_\epsilon]), t_\epsilon) = alt([t_a, t_b, t_\epsilon])$

For the example given in Figure 6, the generic pattern learned is:

tag(line, o, any), tag(word, o, "td"), text(any, "cl", any), opt(tag(any,any,any)), tag(word, o, "td"), text(any, "cl", any), tag(word, c, "td"), tag(word, o, any), text(any, "$n", any), tag(word, c, "td"), tag(line, c, "tr").

Note that there is an optional token in this generic pattern. A larger scale example is shown in Figure 2, which has six optional tokens and one disjunctive token.

5 Experimental Results and Discussions

We have run three experiments to evaluate our algorithm. This section describes the experiments and their results, and compares our approach with other related work.

To evaluate our algorithm on a web site that contains a number of pages with a similar structure, we choose one page from the site at random, apply our algorithm to that page (without any labeling) to obtain a generalised extraction pattern, then apply the pattern to the other pages on the web site to extract all the knowledge units on each page. We consider the pattern successful if it achieves 100% precision and 100% recall for all the knowledge units on the other web pages in a web site. In all of our experiments, each web site consists of 8–10 pages.

5.1 Experiment 1: Learning Patterns from Ten Tabular Web sites

The first experiment was carried out on our basic corpus, which was built by down-loading Web pages from 10 tabular Web sites — 5 in the real estate advertisement domain and 5 in the car advertisement domain. The tabular pages contain missing tokens, mismatched tokens, HTML inconsistencies, and HTML mistakes. The experimental results are shown in Table 1. A pattern with 100% accuracy is marked with a "$\surd$"; a pattern with anything less than 100% accuracy is marked with a "$\times$".

As can be seen from Table 1, our system successfully learned patterns for eight out of the ten tabular web sites — 80% success. This suggests that the pattern learning algorithm is robust for tabular web sites with missing tokens, mismatched tokens, HTML inconsistencies, and HTML mistakes. The system was successful for most web sites in

No	Name of Web site	URL	Domain	K	Results
1	Australian Real Estate Online	http://www.property.com.au/	Reales	3	×
1	Yahoo Classifieds	http://realestate.classifieds.yahoo.com/	Reales	7	√
3	NM Apartments Online	http://www.nmapartment.com/tclarke-bin/aptsearch.pl	Reales	6	√
4	IFP Classifieds	http://www.french-property.com/cgi-bin/ifp/galois.pl?agent=CCC	Reales	4	√
5	Lex Allen	http://freespace.virgin.net/lex.allan/list.htm	Reales	5	√
6	Yahoo Classifieds	http://classifieds.yahoo.com/automobiles.html	Car	9	√
7	AutoWeb	http://www.autoweb.com/	Car	7	√
8	AutoConnect	http://209.143.231.6/	Car	6	√
9	AutoWeb Australia	http://www.autoweb.com.au/	Car	6	×
10	CarSales	http://www.carsales.com.au/	Car	4	√

Table 1. Result of Basic Testing Corpus. (K: No. of knowledge units in each row)

both the real estate advertisement domain and the car advertisement domain, which gives some support for our system being domain independent.

The failures on Web sites (1) and (9) are due to a special representation and content. In site (1), each row with data containing knowledge units is separated by a row containing a single image. In site (9), each single advertisement is represented by two rows of the table, where the first row contains one knowledge unit and the second row contains three knowledge units. In both cases, our system failed to find the similar rows, indicating a limitation of our system on complex tables.

5.2 Experiment 2: Learning Patterns from General Survey Web Sites

The second experiment further examines our learning algorithm's performance on tabular pages using a larger data set. The twenty tabular Web sites used in this experiment cover all the tabular Web sites obtained from a survey of car advertisement services using the index of an independent search engine *LookSmart*. The details of the twenty sites, including their name and URL, and the experimental results are shown in Table 2.

As shown in Table 2, our system successfully learned information extraction patterns in 18 out of the 20 sites — a success rate of 90%. All the patterns were generated within minutes. The experiment provides further evidence for the robustness of the pattern learning algorithm.

Again, the failure on the two Web sites is due to their special page layout. Site (5) presents data in a nested table, in which each cell of the table is another embedded table. The two levels of structure confused our system, so that it was unable to locate the table body. Site (12) presents data in a regular table with three columns, but the data in the third column varied greatly from row to row, sometimes containing an image instead of text. The variation was so great that our system considered the adjacent rows dissimilar, and therefore was not able to construct a pattern.

5.3 Experiment 3: Learning From 30 Sites Collected by Kushmerick

In the third experiment, we tested our learning system on the 30 Web sites collected by Kushmerick, so that we could compared our system with the six systems he developed — *LR*, *HLRT*, *OCLR*, *HOCLRT*, *N-LR* and *N-HLRT* [10]. We analysed each site to determine whether it had a tabular structure or not, since our system assumes that the pages have a tabular structure. Of Kushmerick's 30 sites, 16 were tabular and 14 were not. Note that we would not expect our algorithm to work very well on the non-tabular sites because they do not satisfy the assumptions of our algorithm. The 30 Web sites (from different domains) are shown in Table 3, annotated with their URL, the number of knowledge units to be extracted from each row (K), and whether the site has a tabular structure or not (T/NT).

The results of our system and the Kushmerick's six systems on the 30 web sites are compared and presented in Table 4. Note that we have reordered the sites to separate the tabular sites from the non-tabular sites.

According to Tables 3 and 4, our system successfully learned patterns for 14 web sites out of a total of 30 sites. While this was slightly worse than the systems LR (16/30), HLRT (17/30), OCLR (16/30), HOCLRT (17/30), and N-HLRT (15/30), it achieved much better result than the N-LR (4/30). This performance does not seem particularly good.

However, our system was designed only to learn patterns for semi-structured tabular web sites rather than non-tabular web sites, so that we expected it to fail on the 14 non-tabular sites (in fact, it succeeded on one of them). As can be seen from tables 3 and 4, our system successfully learned patterns for 13 out of the 16 tabular web sites, achieving an 81% success (which is quite similar to the results obtained in the previous two experiments). Kushmerick's algorithms were much less successful, even on the tabular web sites — one of his six systems achieved 56% success (N-HLRT), and the rest achieved a lower rate. This indicates that our system greatly outperforms the other six systems on tabular web sites.

Furthermore, compared with the other six systems, our

No	Name of the Web Site	URL	K	Result
1	AutoConnect	http://209.143.231.6/	6	√
2	AutoWeb	http://www.autoweb.com/	7	√
3	GetAuto	http://www.getauto.com/	5	√
4	AAAA Auto Exchange	http://www.cybernetsol.com/autoexch/	7	√
5	Auto Town Used Car Cinema	http://www.usedcarcinema.com/	5	×
6	AutoVantage	http://www.cuc.com/ctg/cgi-bin/AutoSavings/home?ref=	7	√
7	AutobyInternet	http://www.autobyinternet.com/	5	√
8	Barrett-Jackson Classic Car Auction	http://www.barrett-jackson.com/	6	√
9	Car.Prices.com	http://www.carprices.com/	7	√
10	CarSmart	http://www.carsmart.com/	7	√
11	Hertz Car Sales	http://www.hertzcarsales.com/index_home.html	5	√
12	Limos-On-Line	http://www.limos-on-line.com	3	×
13	Luxury Car Net	http://www.luxcarnet.com/	5	√
14	Megawheels.com	http://www.auto-search.com/	6	√
15	Preowned Vehicles	http://www.dealernet.com/	9	√
16	SIMI Motor Showroom	http://www.simi.ie/	6	√
17	Salvage Management of Syracuse	http://www.salvagemanagement.com/	4	√
18	San Antonio Ambulance Sales	http://saambulance.com	1	√
19	Stoneage	http://www.stoneage.com/	6	√
20	VehicleNet: The Internet Auto Mall	http://www.vehiclenet.com	7	√

Table 2. Result of survey Testing Corpus. (K: No. of knowledge units)

No.	Name of Web Site	URL	K	T/NT
1	Computer ESP	http://www.computeresp.com	4	T
2	CNN/Time AllPolitics Search	http://allpolitics.com/	4	T
3	Film.com Search	http://www.fi lm.com/admin/search.htm	6	NT
4	Yahoo People Search: Telephone/Address	http://www.yahoo.com/search/people/	4	T
5	Cinemachine: The Movie Review Search Engine	http://www.cinemachine.com	2	NT
6	Pharm Web's World Wide List of Pharmacy Schools	http://www.pharmweb.net/	13	T
7	TravelData's Bed and Breakfast Search	http://www.ultranet.com/biz/inns/search-form.html	4	T
8	NEWS.COM	http://www.news.com/	3	T
9	Internet Travel Network	http://www.itn.net/	13	T
10	Time World Wide	http://pathfi nder.com/time/	4	NT
11	Internet Address Finder	http://iaf.net/	6	T
12	Expedia World guide	http://www.expedia.com/pub/genfts.dll	2	NT
13	thrive@pathfi nder	http://pathfi nder.com/thrive/index.html	4	NT
14	Monster Job Newsgroups	http://www.monster.com/	3	T
15	NewJour: Electronic Journals & Newsletters	http://gort.ucsd.edu/newjour/	2	T
16	Zipper	http://www.voxpop.org/zipper/	11	NT
17	Coolware Classifi eds Electronic Job Guide	http://www.jobsjobsjobs.com	2	NT
18	Ultimate Band List	http://ubl.com	2	T
19	Shops.Net	http://shops.net/	5	T
20	Democratic Party Online	http://www.democrats.org/	6	NT
21	Complete Works of William Shakespeare	http://the-tech.mit.edu/Shakespeare/works.html	5	NT
22	Bible (Revised Standard Version)	http://etext.virginia.edu/rsv.browse.html	3	NT
23	Virtual Gardon	http://pathfi nder.com/vg/	3	T
24	Foreign Languages for Travelers Site Search	http://www.travlang.com/	4	T
25	U.S. Tax Code On-Line	http://www.fourmilab.ch/ustax/ustax.html	2	T
26	CD Club Web Server	http://www.cd-clubs.com/	5	NT
27	Expedia current Converter	http://www.expedia.com/pub/curcnvt.ddll	6	NT
28	Cyberider Cycling WWW Site	http://blueridge.infomkt.ibm.com/bikes/	3	NT
29	Security APL Quote Server	http://qs.secapl.com/	18	T
30	Congressional Quarterly's On the Job	http://voter96.cqalert.com/cq_job.htm	8	NT

Table 3. The 30 Web sites collected by Kushmerick for Experiment 3. (K: No. of knowledge units)

system has two important properties. One is that our approach does not require any human labelling of the example pages, whereas all six of Kushmerick's methods need hand labelled training pages. The second property is that

Site No.	T/NT	Our System	LR	HLRT	OCLR	HOCLRT	N-LR	N-HLRT
1	T	√	√	√	√	√	×	×
2	T	√	×	×	×	×	×	×
4	T	√	√	√	√	√	√	√
6	T	×	×	×	×	×	×	×
7	T	√	×	×	×	×	×	√
8	T	√	√	√	√	√	×	√
9	T	×	×	×	×	×	×	×
11	T	√	×	×	×	×	×	×
14	T	√	×	√	×	√	×	√
15	T	√	√	√	√	√	×	√
18	T	√	×	×	×	×	×	√
19	T	√	√	√	√	√	√	√
23	T	√	√	√	√	√	×	√
24	T	√	×	×	×	×	×	×
25	T	√	√	√	√	√	×	√
29	T	×	×	×	×	×	×	×
3	NT	×	√	√	√	√	×	×
5	NT	×	√	√	√	√	×	√
10	NT	×	√	√	√	√	×	×
12	NT	×	×	√	×	√	×	√
13	NT	×	√	√	√	√	×	×
16	NT	×	×	×	×	×	×	×
17	NT	×	×	×	×	×	×	√
20	NT	×	√	√	√	√	√	√
21	NT	×	×	×	×	×	×	×
22	NT	√	√	√	√	√	×	√
26	NT	×	×	×	×	×	×	×
27	NT	×	√	√	√	√	×	√
28	NT	×	√	×	√	×	√	×
30	NT	×	√	√	√	√	×	×

Table 4. Results of our system and other six systems developed by Kushmerick.

our method needs fewer training pages than the other six methods. Kushmerick's six systems need at least 2 example pages for learning a pattern/wrapper for a site. The two Web sites that he ranked as the most difficult, (14 and 1), need an average of 4.8 and 4.5 example pages respectively. Our system only needs a single randomly chosen page as the training example on the same sites. That our system automatically and reliably learns patterns from a single, randomly chosen source file with no human interaction required demonstrates the strength of our approach.

As in the previous experiments, analysis of the tabular sites on which all the systems failed (6, 9, and 29) showed that they had unusual format or structure that broke the assumptions underlying our system. Site 6 has a landscape table in which the columns rather than rows represented the items. On Site 9, some pages have multiple tables; our system successfully learned a pattern that matched one table, but was not able to create a pattern that matched all tables. On some pages of site 29, items were spread over two or more lines of the table, which confused our system. Note also that all of Kushmerick's algorithms also failed on these three sites, suggesting that they were indeed difficult sites.

5.4 Related Work

Our approach differs from most online information extraction systems [8, 11, 12] in that it does not require manual labelled training examples. Other related work which learns patterns without manual labelling is briefly discussed as follows.

An intelligent information agent ShopBot [3] can learn wrappers from un-tagged examples. ShopBot needs a small amount of domain knowledge such as the names of example products and the attributes of products. It learns patterns by querying with familiar products and inducing from multiple response pages such as "failure" pages and "normal" pages. ShopBot works on highly structured data (all data are presented in a uniform format) in a specific product domain. Compared with Shopbot, our system is domain independent and can learn generic patterns from a single "normal" page for semi-structured tabular web sites.

There also have been some systems that can learn patterns from unlabelled Web pages with semi-structured data. A semi-automatic system [1] learns site specific patterns based on heuristics for identifying headings and hierarchies, such as the font size and levels of indentation. This system works on documents with hierarchical organization, but does not handle tabular pages. The wrapper developed by Cohen and Fan [2] learns domain independent patterns from a set of working wrapper programs (automatic or manually coded). This system can work on highly structured data such as lists and hotlists. In contrast with these systems, our approach can learn from tabular web sites with irreg-

ularities such as missing and mismatched tokens, HTML inconsistencies, and mistakes, and only requires the source file of a single web page for the learning process.

6 Conclusions

The paper has described a domain independent approach to automatically constructing information extraction patterns for semi-structured, tabular web pages. The system developed has a pattern language for representing the patterns, a regular structure detection algorithm to identify the repeated structures on a page, and a pattern induction algorithm to generate information extraction patterns. The system is able to infer a site-specific extraction pattern for a web site from a single, unlabeled page selected from the site. The experiments and results show that the pattern language and the pattern learning algorithms are effective and robust for web sites containing pages with a tabular structure. The system has several advantages over similar systems, in that it does not require manual labelling, only requires one web page for learning, and can handle missing tokens, mismatch tokens, HTML inconsistencies and mistakes.

6.1 Limitations and Future Work

Although our approach is domain independent, it does make assumptions about the structure of the pages, and is unlikely to be effective on web sites that do not satisfy those assumptions. Some of these assumptions are fundamental to the approach — that the pages contain descriptions of multiple entities, that the entities are described by a list of attributes, and that the information on the page is structured in the form of successive "rows", each describing one entity, and each presented in a similar format. Relaxing these assumptions would change the nature of the task such that a completely different approach would be required. We believe that there are many web sites that would meet these assumptions, and therefore these assumptions are not a significant limitation.

However, the current implementation of our system makes additional assumptions that simplify the algorithms, but also limit the application of the system. In future work, we intend to remove these additional assumptions and extend the range of web sites to which the system could be applied.

- Our current pattern language is limited in the kinds of generalisations that it can represent, and the pattern induction algorithm is therefore not able to find an effective pattern when the format of the data varies too much across adjacent rows. For example, the generalisation of tags completely ignores all the attributes of the tag, and the generalised text tokens cannot represent strings of a range of lengths. We will investigate a richer generalisation space for the pattern language.

- Our current regular structure detection algorithm relies on heuristics to identify candidate rows before trying to match them. These heuristics are overly specific, and unnecessarily restrict the applicability of the system. We will investigate extending the Smith-Waterman algorithm to identify all the repeating substructure from the whole page, without constructing candidate rows first. We believe that an analysis of the repeating structure will be able to identify the rows in a much wider class of web pages. This will extend the range of applicable web sites.

- The regular structure detection algorithm is also confused by two separate tabular regions on a single page. Currently, the system will only consider the region with the greatest similarity (usually, the largest table). This limitation could be resolved by setting a threshold level, so that the system could identify several regions on a single page and build separate extraction patterns for each one. Extending the Smith-Waterman algorithm, as above, may also enable the system to identify multiple tabular regions on a single page.

- Our pattern language does not handle links and images effectively. Both of these are often part of the desired knowledge units, but the assumptions of the tokeniser causes both links and images to be treated as part of the markup rather than the content. We may need to recognise the special nature of links and images in HTML documents to be able to distinguish them from other kinds of markup tags.

References

[1] N. Ashish and C. A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *Proceedings of Cooperative Information Systems*, 1997.

[2] W. W. Cohen and W. Fan. Learning page-independent heuristics for extracting data from web pages. In *WWW8*, Toronto, 1999.

[3] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent. In *Agent 97*, 1997.

[4] D. Freitag. Information extration from html: Application of a general machine learning approach. In *AAAI-98*, 1998.

[5] Xiaoying Gao. *Knowledge-based Information Extraction from the World Wide Web*. PhD thesis, University of Melbourne, Victoria, Australia, June 2000.

[6] R. Grishman. Information extraction: Techniques and challenges. In Maria Teresa Pazienza, editor, *Information Extraction, a Multidisciplinary approach to an Emerging Information Technology (Lecturer Notes in Artificial Intelligence 1299)*, pages 10–27. Springer, 1997.

[7] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the web. In *Workshop on management of semistructured data*, 1997.

[8] C.-N. Hsu. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In *AAAI'98 Workshop on AI and Information Integration*. http://www.isi.edu/ariadne/aiii98-wkshp/proceedings.html, 1998.

[9] J.-T. Kim and D. I. Moldovan. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):713–724, 1995.

[10] N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, Department of Computer Science and Engineering, University of Washington, 1997.

[11] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Journal of Artificial Intelligence*, 118:15–68, 2000.

[12] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *The 3rd conference on Autonomous Agents(Agent'99)*, 1999.

[13] E. Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1044–1049, 1996.

[14] E. Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 474–479. AAAI Press/ MIT Press, 1999.

[15] D. Smith and M. Lopez. Information extraction for semi-structured documents. In *Workshop on Management of Semi-Structured Data, in conjunction with PODS/ SIGMOD*, Tucson Arizona, 1997.

[16] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. of Mol. Biol.*, 147:195–197, 1981.

[17] S. Soderland. Learning to extract text-based information from the world wide web. In *Proceedings of Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, 1997.