Mining Templates from Search Result Records of Search Engines

Hongkun Zhao, Weiyi Meng
State University of New York at Binghamton
Binghamton, NY 13902, USA
{hkzhao, meng}@cs.binghamton.edu

Clement Yu
University of Illinois at Chicago
Chicago, IL 60607, USA
yu@cs.uic.edu

ABSTRACT

Metasearch engine, Comparison-shopping and Deep Web crawling applications need to extract search result records enwrapped in result pages returned from search engines in response to user queries. The search result records from a given search engine are usually formatted based on a template. Precisely identifying this template can greatly help extract and annotate the data units within each record correctly. In this paper, we propose a graph model to represent record template and develop a domain independent statistical method to automatically mine the record template for any search engine using sample search result records. Our approach can identify both template tags (HTML tags) and template texts (non-tag texts), and it also explicitly addresses the mismatches between the tag structures and the data structures of search result records. Our experimental results indicate that this approach is very effective.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services – Commercial Services, Web-based Services.

General Terms

Algorithms, Performance, Design, Experimentation.

Keywords

Information extraction, wrapper generation, search engine.

1. INTRODUCTION

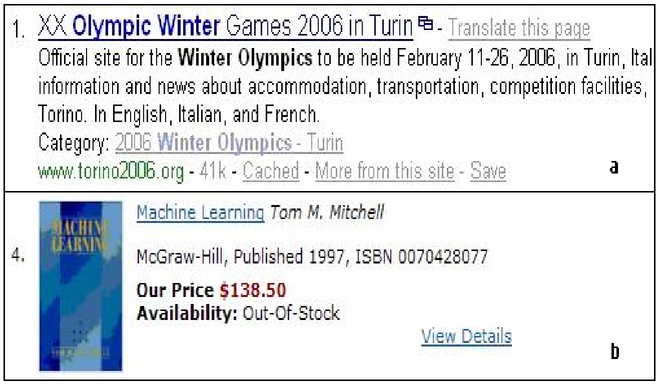
There are primarily two types of search engines, the first is text search engines that search web pages or other text documents and the second is Web databases that search structured data stored in database systems, including most e-commerce search engines. In this paper, we will uniformly call them as search engines.

When a search engine returns results in response to a user query, the results are presented as **s**earch **r**esult **r**ecords (SRRs). SRRs are usually enwrapped with HTML tags in dynamically generated web pages by script programs. Usually, each SRR contains information pertinent to a real world entity. For example, an SRR from a book search engine contains information about a book. Figure 1 shows two sample SRRs from two different search engines, the first from a text search engine and the second from a web database. To be user friendly, search engine designers make SRRs neatly arranged and clearly distinguishable from other content unrelated to the user query on result pages.

Although search engines as well as the result pages they produce are designed for human users to consume, more and more web applications require automatic extraction of SRRs from search engine result pages, such as deep web crawlers [20], shopping agents, large-scale metasearch engines [18, 23], etc.

An SRR usually consists of multiple data units with each having a particular meaning. For example, for the SRR in Figure 1a, its data units include the title of the web page, a short summary (snippet), the URL of the page, etc., and for the book SRR in Figure 1b, the data units include the book title, the author, the publisher, etc. In general, for SRRs returned from the same search engine, the data units in these SRRs are laid out and formatted similarly following a certain pattern. In other words, there exists an *SRR template* that is followed by all SRRs from the same search engine. Each SRR is an instance of the SRR template.

In this paper, we use *data units* to represent the *values* of real world entities but not formatting and template information. For example, in Figure 1b, "Our Price" is part of the template and "$138.50" is a data unit. Sometimes, we use "attributes" to represent the meaning of a data unit or a set of data units. For instance, "Our Price" is the attribute of "$138.50".



1. XX **Olympic Winter** Games 2006 in Turin - Translate this page
Official site for the **Winter Olympics** to be held February 11-26, 2006, in Turin, Ital
information and news about accommodation, transportation, competition facilities,
Torino. In English, Italian, and French.
Category: 2006 **Winter Olympics** - Turin
www.torino2006.org - 41k - Cached - More from this site - Save

4. Machine Learning *Tom M. Mitchell*
McGraw-Hill, Published 1997, ISBN 0070428077
Our Price $138.50
Availability: Out-Of-Stock
View Details

Figure 1. Non-tabular formatted SRRs

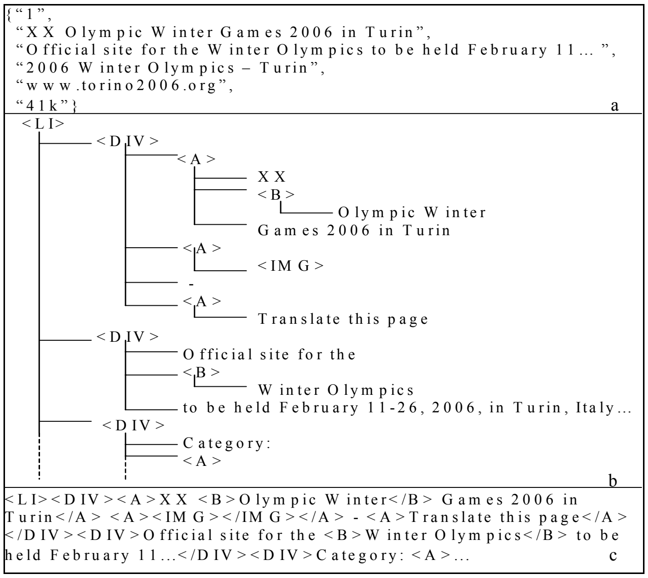
In this paper we study the problem of how to automatically identify the SRR template for any given search engine. Based on our observation, we believe the following three factors make automatic SRR template extraction a difficult problem:

1. *Data structure and tag structure mismatch*. As part of a result page encoded in HTML, each SRR is also formatted using HTML tags. The structure of the tags for an SRR will be called the *tag structure* of the SRR (see section 3 for more details) while the composition of the data units in the SRR will be called the SRR's *data structure*. Figure 2a shows the

KDD'07, August 12–15, 2007, San Jose, California, USA.

data structure of the SRR in Figure 1a, which basically contains the data units in the order they appear in the SRR; Figure 2b and 2c show two different views of the tag structure of the SRR, one in tag tree (or tag forest) and the other in tag string. In general, the tag structure and the data structure of an SRR may not match in two possible ways: one data unit is encoded in multiple tags and multiple data units are encoded in one tag (a matching pair of starting and ending tags is considered as one tag here). These mismatches make using the tag structures to identify data units difficult.

2. *Optional/disjunctive component*. An SRR may contain optional components whose corresponding data units may be missing in some SRRs (e.g., some book SRR has a review link and some don't) or disjunctive components that may have different types of data units in different SRRs (e.g., a book SRR from a search engine has either a regular price or a discount price). Unfortunately, in general, unlike the XML output generated by web services, no information about the optional/disjunctive components can be found from the HTML source code of each SRR.
3. *Template tags and template text*. The SRR template may contain both template tags and template text. The former is part of the tag structure of the SRR and the latter often mingles with data units. Due to the intermingling of template text and non-template text (data units), separating them accurately and automatically is not easy. Furthermore, not all tags are part of the real template and these tags need to be identified and removed to facilitate the identification of real template tags.



```
{"1",
 "XX Olympic Winter Games 2006 in Turin",
 "Official site for the Winter Olympics to be held February 11... ",
 "2006 Winter Olympics – Turin",
 "www.torino2006.org",
 "41k"}                                                          a
```

```
<LI><DIV><A>XX <B>Olympic Winter</B> Games 2006 in
Turin</A> <A><IMG></IMG></A> - <A>Translate this page</A>
</DIV><DIV>Official site for the <B>Winter Olympics</B> to be
held February 11...</DIV><DIV>Category: <A>...              c
```

Figure 2. The data structure and tag structure (tag tree and tag string) of an SRR

While there are several reported works on data extraction from web pages, including both data record (i.e., SRR) level and data unit level extraction, the current techniques are inadequate in precise and automatic template extraction (please see detailed comparison with related work in section 2). In this paper, we propose a novel solution to the automatic SRR template extraction problem that explicitly addresses the above factors.

The main contribution of this paper is the development and evaluation of a new SRR template mining algorithm for any given search engine. Our algorithm has the following novel features. First, we introduce a directed acyclic graph (DAG) template model to help solve the optional/disjunctive component problem. Second, we consider the identification of both template tags and template texts. Specifically, we develop a solution to identify *decorative tags* that are the main non-template tags; we also develop a technique to identify template texts so non-template texts can be extracted as data units. These solutions are domain independent and search engine independent, meaning that they can be applied to the SRRs from any search engine in any domain. Third, we explicitly study the mismatch problem between the tag structures and data structures of SRRs. We show that identifying decorative tags help solve the mismatch that one data unit is encoded in multiple tags and identify template texts help deal with the mismatch that multiple data units are encoded in one tag. Fourth, we also introduce a clustering and voting based method to generate SRR templates and this method tends to generate more robust SRR templates. This means that even when the input SRRs may contain false SRRs which may lead to outliner tags and texts, our method can still obtain the correct template. Fifth, our method is fully automatic except a one-time, domain independent and search engine independent training of the components for identifying decorative tags and template texts. We also evaluate our solution experimentally and the results indicate that our solution is quite effective.

The rest of the paper is organized as follows. We review related works in section 2. In section 3, we present the template model and the overview of our method. Section 4 gives the major steps of our algorithm. Section 5 presents the experimental results. Finally section 6 concludes the paper.

2. RELATED WORKS

Information extraction from web pages is an active research area. Researchers have been developing various solutions from all kinds of perspectives. Readers may refer to [6, 13] for surveys about early and recent works in this area.

Many web information extraction systems [1, 3, 9, 10, 11, 12, 16, 19, 26] rely on human users to provide marked samples so that the data extraction rules could be learned. Because of the supervised-learning process, those semi-automatic systems usually have higher accuracy than fully automatic systems that have no human intervention. But semi-automatic methods are not suitable for large-scale web applications [18, 20, 23] that need to extract data from thousands of web sites. Also web sites tend to change their web page formats frequently, which will make the previous generated extraction rules invalid, further limiting the usability of semi-automatic methods. That's why many more recent works [2, 4, 5, 7, 8, 14, 15, 17, 21, 22, 25, 27, 28] focus on fully or nearly fully automatic solutions.

Web information extraction can be at the record level or data unit level. The former treat each data record as a single data unit while the latter go one step further to extract detailed data units within the data records. Record level extraction generally involves identifying data regions that contain all the records, and then partitioning the data regions into individual records. Since the records within a data region are usually highly homogeneous and the data regions are often constructed simply by a list of records, the record level extraction is easier than the data unit level extraction. Some recently proposed fully automatic extraction methods on record level extraction achieved satisfying performances [14, 21, 27]. On the other hand, the data unit level extraction is more complicated, and the performance of proposed fully automatic methods are not satisfactory.

Omini [4], IEPAD [5], MDR [14], ViNTs [27] and the method in [8] are record-level data extraction tools incapable of performing general data unit level extraction. RoadRunner [7], EXALG [2], DeLa [22], DEPTA [25], NET [15], ViPER[21] and the method in [28] and [17] are more relevant to this paper, because they all have the ability to extract data at data unit level. DeLa and the method in [28] and [17] also studied the automatic data unit annotation problem.

RoadRunner extracts data template by comparing web page pairs. One page is considered as initial template, and the other page is compared with the template, which is updated when there are mismatches. The algorithm tries to produce a template that fits all input web pages. EXALG extracts template by analyzing equivalence classes, which are sets of tokens that have the same frequency of occurrence on all input web pages. Templates are generated from large and frequently occurring equivalence classes (LFEQs). DEPTA uses an edit distance based tree matching technique to align tag trees for data extraction. Tags are considered as templates while texts are data to be extracted. NET extends DEPTA by supporting nested records extraction. Similar to IEPAD, DeLa builds suffix trees to detect patterns in web page string. DeLa's multi-level pattern extraction algorithm enables it to extract data with nested schema on the web page. ViPER improves MDR by providing a better sub-tree comparing method that allows consecutive data records with various lengths. ViPER also introduces a multiple sequence alignment algorithm that aligns maximal unique matches at different levels to extract the template of data records. Instead of aligning template, the method in [17] aligns data text between template tokens directly by comparing the features like content, presentation style, type, etc. The method in [28] combines the record extraction and attribute labeling to let them benefit from each other.

Many works [7, 15, 21, 25] ignore the problem of mismatches between the data structures and the tag structures of SRRs by assuming that HTML tags are template tokens and text tokens are data items to be extracted. The method in [28] simply assumes that the visual elements (with the exception of "noise") are data attributes to be labeled. The authors of EXALG noticed that HTML tags might appear in data while texts might contain template tokens. But their equivalence classes based method considers only tokens that have constant number of occurrences in different web pages as template tokens. Thus its ability to differentiate the different roles of tags and text tokens is limited.

Our work reported in this paper does not deal with record-level extraction even though we mention SRRs frequently. Instead, our work takes already extracted SRRs as input and tries to find the SRR template based on the SRRs returned from a search engine. We build our system to work on the SRRs extracted by ViNTs [27], which is a fully automatic record-level extraction tool. However, we recognize that ViNTs is not perfect, and as a result, incorrect SRRs may be extracted by it and these SRRs may be part of the input to our algorithm.

In this paper, we focus on mining the SRR template. Even though extracting data units is not our direct objective as we try to tackle the more fundamental template extraction problem, by identifying the SRR template for the SRRs of a search engine, data units in these SRRs can be extracted easily. Therefore, this paper is closely related to data unit level extraction. Our method differs from other solutions as we deal with the three factors (i.e., data structure and tag structure mismatch, optional/disjunctive component, and template tags and template text) more explicitly than existing works. Also the performance of our system is significantly better than that of contemporary works.

A fully automatic web information extraction tool should be robust to outlier inputs. Because it is not practical to simply assume that the inputs consist only of the desired data. Especially when we consider a working environment that is as complex as the World Wide Web. Our statistical method also has the advantage that is robust to outlier inputs which may be caused by the existence of false SRRs due to the use of an imperfect SRR extraction tool.

The authors of [24] showed that the problem of inferring unambiguous schemas (templates) for web data extraction is NP-complete. Thus it is understandable that all current efficient web information extraction tools, including ours, are based on heuristic solutions.

3. FUNDAMENTALS AND OVERVIEW

3.1 Basic Concepts

An HTML web page mainly consists of two types of elements: HTML tags and texts. We use the term tag and HTML tag interchangeably in this paper because we only work with HTML encoded web pages. A **tag** refers the name of any HTML tag that is encompassed by special characters '<' and '>' in the HTML source page, while the term **text** refers to characters that are not encompassed by '<' and '>'. When we consider a web page as a string of tokens (tag tokens and text tokens), an SRR is a sub-string of the token string of the entire web page that contains the SRR. The creation of the HTML code pieces of an SRR can be considered as using a template to enwrap the data units of an SRR. An SRR template consists of template strings and data slots that hold data units. Once we have the template, we can apply it to an SRR to identify the data units in the SRR.

The embedding nature of tags enables a well-formed HTML document to be converted to a tree structure, which contains two types of nodes: tag nodes and text nodes. We use the term tag tree to represent the tree structure of an HTML document in this paper. With the tag tree view of a web page, an SRR is part of a tag tree. In general, an SRR corresponds to a sub-forest located in a tag tree under a specific tag node.

Due to the loose HTML grammar, many web pages on the web are not well formed. However, a good browser can still "correctly" build the tag trees for a majority of ill-formed web pages. Note that by a pre-order traversal of the tag tree, we can always get a well-formed HTML document, which is equivalent to the tag tree. We use the term *tag string* to refer to the well-formed HTML code generated by tag tree traversal. Thus an SRR also corresponds to a tag string that is equivalent to its tag forest. We call an SRR's tag forest (or the equivalent tag string) the *tag structure* of the SRR, while the composition of the data units in the SRR the SRR's *data structure*.

In the tag string, HTML tags partition the texts in an SRR into small text fragments. Thus one may be tempted to extract data units based on this partition. Such a naïve method may work ok for search engines that arrange their SRRs in a tabular manner, e.g. each row represents an SRR and each column corresponds to a data unit. But based on our observation, most search engines do not arrange their SRRs in tabular format and in these cases the text fragments are often different from the data units.

Because HTML is a language for data presentation, although the tag structure of an SRR is generated according to the data structure of the SRR, the tag structure of an SRR is often considerably different from that of the SRR's data structure. Figure 2a and 2b&2c show such an example. There is no direct mapping between a node on the tag tree and a data unit: on the one hand, multiple nodes may match to one data unit, and on the other hand, one node may correspond to multiple data units.

3.2 Template Model

An SRR represents an instance of the SRR template on a web page. There are two types of strings in the HTML code of an SRR: the template strings and data strings. To generate an SRR, the search engine's script program enwraps data strings using template strings, which are token strings that are not content of any data units in an SRR. We introduce the concept of *data slots*, which are the carriers of the data units of SRRs. We define the SRR template based on data slots. An SRR template actually represents the relationships between data slots.

Definition 1 (Data Slot). A data slot is a triple <*lt*, *ds*, *rt*>, where *ds* represents the space that holds a data unit of an SRR (note that *ds* is not the data unit itself), *lt* (*rt*) is a template string that bounds *ds* on the left (right) side. We call *lt* (*rt*) as the data slot's left (right) bound. A data slot may have empty left or right bounds.

In Figure 2, the data slot that holds the data string "XX Olympic Winter Games 2006 in Turin" has a template string "<LI><DIV><A>" as left bound, and a template string "</A> <A><IMG></IMG></A> -…" as right bound.

When data slots DS_i = <lt_i, ds_i, rt_i> occurs immediately before data slot DS_j = <lt_j, ds_j, rt_j>, the right bound of DS_i is the same as the left bound of DS_j, i.e. $rt_i = lt_j$. In general, every SRR can be represented as a sequence of data slots and we use dss(SRR) to denote this sequence.

Definition 2 (SRR template). For a given search engine S, its SRR template, denoted SRRT(S), is a directed acyclic graph (DAG), called a **template graph**, denoted TG(S), where each node in the graph represents a data slot and an edge from data slot ds_1 to data slot ds_2 indicates that there exists a valid SRR produced by S in which ds_1 appears right before ds_2. In addition, for any valid SRR produced by S, there is a full directed path in TG(S) that is the same as dss(SRR), where a full directed path starts with an origin node (which has no incoming edges) and ends with a destination node (which has no outgoing edges). In general, we also require the template graph to be minimum (in terms of the number of nodes and edges) which means identical nodes and edges have been merged.

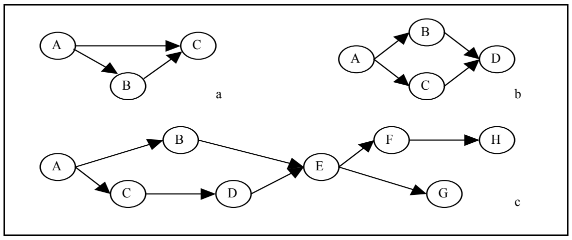


Figure 3. Sample template graphs

In general, TG(S) may have multiple origin nodes and multiple destination nodes. If the template graph of a search engine has just one full path, this means the SRRs of the search engine have a rigid layout format and every SRR has the same sequence of data slots. If there is an optional data slot that some SRRs have and some SRRs don't have, then the template graph will have a fork. Figure 3a shows an optional data slot B between data slot A and data slot C. Figure 3b illustrates two disjunctive data slots, B and C, between A and D, i.e., after A, either B or C, but not both, may follow, and both of them will be followed by D. Figure 3c shows a template graph with 4 full paths.

The method we present in this paper can automatically generate the graph representation of SRR template from a set of result pages that contain SRRs returned by a search engine.

3.3 Handling Mismatches

To correctly restore the SRR's data structure from its tag structure, we must be able to solve the problem of the mismatches between the data structure and the tag structure. There exist two major types of mismatches: a single data unit matches multiple nodes in the tag structure, and multiple data units match a single node in the tag structure. Figure 1a shows an example of a Type 1 mismatch. The text "XX Olympic Winter Games 2006 in Turin" is a single data unit, but it corresponds to four nodes in the SRR's tag structure: text "XX", text "Games 2006 in Turin", tag "B" and its child text "Olympic Winter" (see the tag structure in Figure 2b). Figure 1b shows an example of a Type 2 mismatch: the text string "McGraw-Hill, Published 1997, ISBN 0070428077" contains three data units corresponding to the name of a publisher, the year the book was published and the ISBN number of the book, respectively, but the entire text string is contained in a single text node in this SRR's tag structure.

3.3.1 Type 1 Mismatch

Let's examine the example in Figure 1a and Figure 2 more closely. It is the HTML tag "B" that splits the single text string into three pieces. Highlighting certain words in an SRR is actually a general phenomenon because Web page designers often use special HTML tags to embellish certain pieces of information to make them special. For example, they may want to highlight the query terms, use hyperlink to indicate further information is available, or, insert a small picture to make the web page more lively, etc. We call such kind of HTML tags **Decorative Tags**.

For the purpose of data extraction, we don't consider the tags that embellish a complete data unit as decorative tags, because these tags actually facilitate the data extraction. We are interested in those tags that embellish a portion of a data unit. By identifying and removing them, we would be able to restore the wholeness of the data units.

Definition 3 (Decorative Tag). A decorative tag is an HTML tag that appears within a single data unit of an SRR and is designed to embellish a portion of the data unit. In other words, a decorative tag has the effect of dividing a single data unit into multiple text fragments with each corresponding to a different node in the tag structure of the SRR.

Because decorative tags only embellish a portion of a data unit, their occurrences would accompany the occurrences of the information that is being embellished. For example, the occurrences of query terms in the SRRs are embellished (Figure 1a.). In general, we observed that decorative tags tend to have random occurring patterns because the texts to be decorated may appear randomly in an SRR, while non-decorative tags tend to have more regular occurrence patterns. This property enables us to employ a machine learning approach to identify the decorative tags. We will introduce our neural network based decorative tag detector in section 4.1.

3.3.2 *Type 2 Mismatch*

To help users correctly understand the data in an SRR, there usually exist some special text tokens in the text that contains multiple data units to separate these data units. For example, the string ", Published" and ", ISBN" in Figure 1b are the text tokens that separate the text into three data units "McGraw-Hill", "1997" and "0070428077". Note that "Published" and "ISBN" actually correspond to semantic labels (or attribute names using database terminology) that give meanings to data units. We call text tokens that separate different data units in a string as **Template Texts**. Note that a template text can be as simple as a punctuation symbol like a comma or a semi-colon.

Definition 4 (Template Text). A template text is a non-tag token that is not (part of) any data unit (or attribute values) in the SRR that contains the token.

The problem of template text detection is to differentiate template texts from non-template texts. A significant characteristic of template texts is their high occurring frequency because they tend to appear in almost all (with the exception of optional data units) SRRs that are returned by the same search engine. But some non-template may also have high occurring frequencies, like punctuations, and words like "the" and "a". Further observation indicates that the template texts and non-template high frequency texts have different occurring patterns: the latter tend to have random occurring patterns while the former have more regular occurring patterns. Our solution will explore these special features. We introduce our neural network based template text detector in section 4.3.

3.4 Method Overview

Our SRR template mining algorithm includes two neural network classifiers: the decorative tag detector (DTD) and the template text detector (TTD). Their design and training details will be given in sections 4.1 and 4.3, respectively. Once we have the DTD and the TTD, we can plug them into the SRR template mining algorithm.

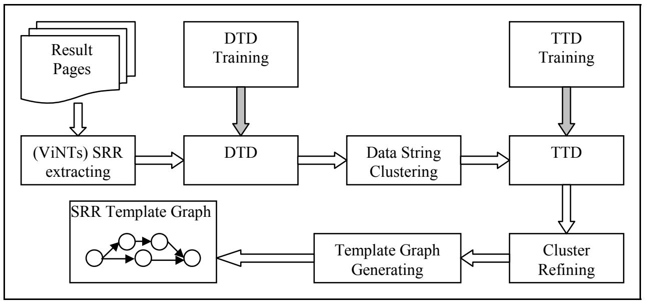


Figure 4. Method overview

Figure 4 shows the main steps of our algorithm. The decorative tag detector (DTD) and template text detector (TTD) will be trained in advance. The algorithm works as follows. The input is a set of sample result pages that contain SRRs returned by a search engine. First we apply ViNTs [27] on these sample pages to extract the SRR at the record level. Then we apply DTD on the tag forests of the SRRs to detect and remove decorative tags from the tag forests. We transfer the SRRs' tag forests into equivalent tag strings and extract data strings from them. The data strings are considered as candidate data units of the data slots in the SRR template and a candidate data unit may contain multiple real data units due to the existence of template texts in it. Next we apply a hierarchical clustering algorithm to cluster candidate data units extracted from all input SRRs. The next step is to apply TTD to each candidate data unit cluster to identify the template texts for this cluster. Then we use a refining step to separate the candidate data units in each cluster into data units using the newly identified template texts. This effectively creates multiple (including just one) data unit clusters out of each candidate data unit clusters. The last step is to build the SRR template graph. We map each data unit cluster to a vertex (a data slot) on the SRR template graph, and then use all SRRs to vote for the edges on the graph. We delete edges and vertices (data slots) with very small supports so that the final SRR template graph is robust and reliable.

4. TEMPLATE MINING ALGORITHM

As mentioned in section 3.4, we apply ViNTs [27] to the input result pages that are to be used to mine the SRR template. The output of ViNTs is a set of SRRs represented by their tag forests. ViNTs is a fully automatic but it does not guarantee that the SRRs extracted by this tool are 100% correct. The existence of incorrect SRRs brings some noise to the SRR template extraction process.

The following sub-sections presents all modules described in Figure 4 other than the SRR extractor.

4.1 Decorative Tag Detector (DTD)

The existence of decorative tags in the tag structure (Tag tree) causes the Type 1 mismatch problem. To solve the problem, we introduce a DTD, which is a neural network classifier, to identify the decorative tags, so that they could be removed from the tag structure of SRRs.

4.1.1 *Tags on tag forests*

DTD takes a tag, which is represented by its statistical features, in SRRs' tag forests as input. The output is a number indicating if the input tag is a decorative tag or not. An input tag is not merely a tag in the tag forest of a single SRR, it represents the same tag name that occurs in the similar locations of all SRRs returned by the same search engine. It is advisable to combine SRRs on different result pages of a search engine to extract a tag's features.

Every HTML tag has a tag name, for example, <TABLE>, <A>, <B>, etc. It is possible that one occurrence of a tag is a decorative tag while another occurrence of the same named tag is not a decorative tag. In other words, different occurrences of the same named tag may play different roles in SRRs. Because the tag forests of SRRs returned by the same search engine have similar structures, tags with the same name and appearing in the same specific locations of the tag structures of different SRRs are very likely to play the same role.

We use **Tag Path** [27] to specify the location of a tag on the tag forest. A tag path consists of a sequence of **path nodes**. Each path node *pn* consists of two components, the *tag name* (i.e., a tag node) and the *direction*, which indicates whether the next node following *pn* on the path is the next sibling of *pn* (indicated by "S", called S node) or the first child of *pn* (indicated by "C", called C node). Since we need to compare the tag locations on SRRs, we don't need the absolute tag paths that start from the tag tree root <HTML>. We work on relative tag paths that start from the root nodes of SRRs. As an example, the (relative) tag path of the first tag <B> of the tag structure in Figure 2 is: <LI>C<DIV>C<A>C<#TEXT>S.

The tag structures of different SRRs returned by the same search engine generally are not exactly the same. As a result, the tag paths of different occurrences of a node that plays the same role in different SRRs are likely to be different. We convert a tag's tag path into compact format following the method reported in [27], which is equivalent to the original tag path for locating a tag. By removing unimportant "noise" path nodes, a compact tag path can be more robust to represent a location on the tag forest of SRR than the original tag path. Thus a tag in the tag forest of an SRR is represented as <*n*, *p*>, while *n* is the tag name, *p* is the compact tag path of this tag on the tag forest of the SRR.

4.1.2 Decorative tag detector design

We discussed that one important property of decorative tags is that they have a random occurring patterns. A careful analysis reveals more: decorative tags are designed to embellish texts, which are leaf nodes on the tag structure. Thus the decorative tags tend to be located close to leaf nodes. Mostly they are the direct parents of the text nodes embellished by them. Also certain types of tags are likely to be used as decorative tags than others, for example, <B>, <I> are more likely than <TABLE>.

We extract the 9 features (in Table 1) of an HTML tag that appears in the tag forest of SRRs. Features 1 and 2 capture the randomness of a tag's distances to its leaf descendants. Feature 2 is the estimated *standard deviation* of the distances of the tag (in occurrences on different SRRs) to its leaf.

Features 3, 4, 5, 6, 7 and 8 capture the randomness of a tag's occurring patterns under its direct parents. The feature 4, 6 and 8 are all standard deviations. All appearances of a tag directly under a common parent node together are considered as an occurring pattern. Feature 9 is the prior-probability of the tag name as decorative tags.

The DTD is a three layer backpropagation neural network classifier, with an input layer that consisting of 9 units, a hidden layer consisting of 4 units, and an output layer consisting of 1 unit. A tag is considered as decorative tag if the output has a value greater than 0.5.

Table 1. HTML tag features

Feature #	Description
1	Average distance to leaves
2	Deviation of distance to leaves
3	Average occurring number
4	Deviation of occurring number
5	Average first occurring position from left
6	Deviation of first occurring position from left
7	Average first occurring position from right
8	Deviation of first occurring position from right
9	Prior-probability of being a decorative tag

To collect sample tags for a search engine, we collect *N* result pages returned by the search engine. Then we use an automatic SRR extraction tool to extract the SRRs from the *N* result pages. Let's assume *n* SRRs are extracted.

We extract all HTML tags from the tag forest of each SRR. Note that a tag is represented as a tag name and tag path pair. By extracting the occurrence pattern of each tag in each SRR, we can get the statistics features (1-8).

At training stage, feature 9 is obtained by human users by manually marking the samples. Then we store the prior-probability values in a hash table, so that we can find the prior-probability of a tag quickly when the trained DTD is applied to detecting decorative tags.

We should point out that to train the DTD, we should collect sample tags (both positive and negative) from many search engines in different domains so that more varieties are taken into consideration. However, the training uses all the collected sample tags and is done only once. There is no separate training for each individual search engine. In other words, the training is domain-independent and search engine independent, and the trained DTD is also domain-independent and search engine independent. As a result, once the DTD is trained, it can be applied to any search engine to identify the decorative tags in its SRRs.

We should also note that when the trained DTD is applied to classifying the tags to extract the SRR template for a particular search engine, we need to collect sample tags from that search engine (see section 4.1.3 for more details about this).

4.1.3 Removing decorative tags

Once the DTD is trained, we are ready to use it in the SRR template mining algorithm. After we applied the SRR extraction tool on the sample pages returned by a search engine, we have a collection of SRRs. We check all SRRs to collect HTML tags and their occurring pattern statistics from the SRRs' tag forests. Thus we can calculate the features 1 – 8 for each tag (a <*n*, *p*> pair). We already have a prior-probability table for tag names obtained from DTD training. The feature 9 of a tag can be obtained by a simple table looking-up. A neutral 0.5 is assigned to tags that do not exist in the prior-probability table.

We feed the statistical features of each tag into DTD. An output greater than 0.5 indicates the input is a decorative tag. Once all decorative tags are identified. We traverse the tag forests of all SRRs again to remove the decorative tags from them. To remove a node from a tag forest, we simply take its immediate children to replace the node itself in the tag forest. Figure 5 shows an example. The node C in the tag tree in Figure 5a is a decorative tag. After it is removed, the tag tree becomes the one in Figure 5b.

4.2 Data String Clustering

After the decorative tags are removed from the tag forest of an SRR, we pre-order traverse the tag forest to generate the tag string (Figure 2c shows part of a tag string). We partition the tag string into segments by the alternatively occurring tags and texts, such that no segment contains mixture of tag and text, and, each segment is maximized. Since decorative tags have been removed, the segments that consist of tags are considered as template strings, while the segments that consist of texts are considered as initial data strings (data units). Let *d* represent a data string, *t* represent a template string, and $ts = \{t_1, d_2, t_3, \ldots, t_{i-1}, d_i, t_{i+1}, \ldots, t_n\}$ be a tag string.

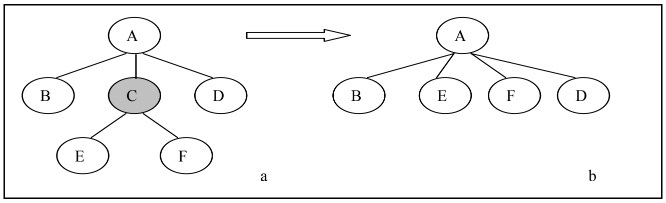


Figure 5. Removing Decorative Tag

Because of the existence of possible Type 2 mismatch, there might be template texts in d_i. Thus a general data string may be a combination of multiple data units plus template texts that bound them. Before we can detect the template texts, we need to group the data strings that are data units of the same semantic meaning (e.g., values of the attribute) in different SRRs, so that we can exploit the statistical properties of template texts.

Previous works [2, 5, 15, 21, 22, 25] developed sophisticated alignment techniques to group data strings. One problem of alignment-based techniques is that they are too sensitive to outliers. The existence of data that are extracted from false-SRRs (due to the imperfect nature of the SRR extraction tool used) will significantly degrade the quality of the outcome. Since we cannot guarantee that the SRRs we have are 100% correct when using a fully automatic tool, more robust method for grouping semantically related data strings is necessary. Our method uses a hierarchical clustering algorithm to group the data strings.

Generally, a data string is bounded by two template strings on a tag string. Thus a data string can be represented as $D = <t_L, d, t_R>$, While d represents the data string itself, t_L and t_R represent its left and right bounding template strings respectively. Note that t_L and t_R might be empty. Ideally, a data string with left and right bounds is an instance of a data slot in the SRR template. But at this point, we need further processing to get more reliable results.

A data string d is a text string, which might be a hyper link, a date value, a numeric value, etc. We define several text types of data strings to capture the different nature of text strings. Current prototype system defines the following 5 text types: NUMBER, DATE, HYPER LINK, PRICE and TEXT. The type "TEXT" refers to the text that can't be recognized as any of other types.

Let $length(d) = 1$ and $length(t) = 1$. Thus the length of a tag string ts is: $length(ts) = \sum length(d) + \sum length(t)$. The position of d_i in ts is defined as $pos(d_i) = i / length(ts)$.

Let $DT(t_i, t_j)$ represent the distance between template strings t_i and t_j, let $DTP(d_i, d_j)$ represent the type distance (a predefined distance matrix of types is used) between d_i and d_j, and let $DP(d_i, d_j) = |pos(d_i) - pos(d_j)|$. Let $DD(D_i, D_j)$ represent the distance between two data strings: $D_i = <t_{Li}, d_i, t_{Ri}>$ and $D_j = <t_{Lj}, d_j, t_{Rj}>$. To avoid one cluster contains two data strings from the same SRR, $DD(D_i, D_j) = \infty$ if D_i and D_j are from the same SRR, otherwise, the following formula is used:

$$DD(D_i, D_j) = \frac{K_1 \times DT(t_{Li}, t_{Lj}) + K_2 \times DTP(d_i, d_j) + K_3 \times DT(t_{Ri}, t_{Rj}) + K_4 \times DP(d_i, d_j)}{K_1 + K_2 + k_3 + k_4}$$

We apply a bottom-up hierarchical clustering algorithm on all data strings extracted from all SRRs of the same search engine. The distance between two clusters is computed by *single link*. A distance threshold is set up to terminate the clustering process.

4.3 Template Text Detector (TTD)

TTD is designed to identify template texts from data string clusters. For each cluster, we first identify high frequency texts. Intuitively, a template text tends to have a high occurring frequency, but not all texts that occur frequently are template texts. Some common tokens, like "the", "of" and punctuations, also tend to have high frequencies in most text strings.

The TTD is a neural network classifier that classifies high frequency text strings into two categories: template texts and non-template texts. Before introducing the TTD design, we would like to describe how to extract high frequency texts from a cluster first.

4.3.1 Extracting high-frequency texts

Suppose we have a data string cluster c, which contains n data strings. The high-frequency texts are sub-strings of data strings in c that occur in many other data strings in c. Because a template text tends to occur in a relatively fixed position in data strings, we do not use the simple term frequency. Instead, we introduce a position-weighted voting algorithm to find out the high-frequency texts. The basic idea is that the same token occurring at the same position in different data strings is more likely to be a template text. Figure 6 shows the algorithm for extracting high-frequency texts from a data string cluster.

A data string d of length m consists of m tokens $d = \{tk_i \mid 1 \leq i \leq m\}$. Each token tk_i has a relative position, which is defined as $Ptk_i = i/m$. Thus the position distance between two tokens can be computed as $DPT(tk_i, tk_j) = | Ptk_i - Ptk_j |$.

We pick up k data strings by randomly sampling the data strings in C as seeds. For each seed data string, we take its tokens as the seed tokens. All seed tokens have an initial weight of 0. Then tokens in every other data strings in C are used to vote for seed tokens. E.g., if a token tk in d is the same as a seed token tk_s, the weight of tk_s is updated as $W'(tk_s) = W(tk_s) + (1- DPT(tk_i, tk_j))$.

After the voting, for each seed data string, we group maximum consecutive tokens that have a weight higher than a threshold as the high-frequency texts. We take all high-frequency texts extracted from the k seed data strings as the samples for *TTD*.

```
Algorithm $High\_Freq\_Text(C)$
  $H \leftarrow \Phi;$
  $S \leftarrow$ sampling data strings in $C;$
  for $s \in S$ do
    for $tk_s \in s$ do
      $W(tk_s) \leftarrow 0;$
    for $d \in C$ and $d \neq$ s do
      for $tk_s \in s$, $tk_d \in d$ do
        if $tk_s = tk_d$
          $W(tk_s) \leftarrow W(tk_s) + (1-DPT(tk_s, tk_d));$
      end for;
    end for;
    identify $\forall$ max sub-string $ss \subset$ s and $W(tk) > T$ for $\forall$ $tk \in ss;$
    put $\forall$ $ss$ into $H;$
  end for;
  return $H$;
```

Figure 6. Extracting High-frequency Texts

4.3.2 TTD design

Similar to DTD, TTD is also a neural network classifier, which can identify template texts from high-frequency texts extracted from a data string cluster. The inputs of TTD are the statistical features of high-frequency texts. Table 2 shows the 7 features used in current TTD.

Table 2. High-frequency text features

Feature #	Description
1	Average occurring number
2	Deviation of occurring number
3	Average first occurring position from left
4	Deviation of first occurring position from left
5	Average first occurring position from right
6	Deviation of first occurring position from right
7	Prior-probability of being a template text

Features in Table 2 have the similar meanings as their counterparts in Table 1. Features 1-6 are used to capture the randomness of the occurring patterns of a high-frequency text in data strings. Feature 7 is the highest value of the priorprobabilities of the tokens in the high-frequency text to be a template text.

The TTD is a three-layer backpropagation neural network classifier, with an input layer consisting of 7 units, a hidden layer with 3 units and an output layer with 1 unit. We use the training samples extracted from the same result page set that is used to train DTD to train TTD. The sample high-frequency texts are collected as follow: after SRRs are extracted from result pages, apply DTD to detect and remove decorative tags, and then cluster data strings and collect high-frequency texts and their features from the clusters as the samples for TTD.

Similar to DTD, TTD also needs to be trained only once and the trained TTD is domain-independent and search engine independent so it can be applied to any search engines.

4.3.3 Identifying template texts and refining data string clusters

We perform the data string clustering to group data units of the semantic type together. To reduce the effect of outliers (false-SRRs), small clusters with their numbers of members smaller than a threshold are discarded. We extract high-frequency text strings from every cluster using the method in section 4.3.1. Then the occurrences of each high-frequency text in every data string in the cluster are checked to collect the statistical features (1-6). The prior-probability of the high-frequency text is obtained by a table look-up. The highest prior-probability of all tokens in the high-frequency text is used as the prior-probability of the text. A neutral 0.5 is assigned if no token in the text exists in the prior-probability table.

Each high-frequency text is fed into TTD. An output larger than 0.5 indicates the input is a template text. All identified template texts are used to further partition the data strings. The template texts on the left and right boundaries are merged into their template string neighbors.

We cluster the newly generated data strings again using the method described in section 4.2. After removing the small clusters, all kept data strings are used to generate template graph.

4.4 Generating Template Graph

A data string cluster of size N contains N data strings from N different SRRs. An SRR, on the other hand, contains M data strings that belong to M different clusters. Ideally, each data string cluster represents a data slot in the SRR template. Note that the order of data strings in an SRR represents the relationship of the data slots corresponding to the clusters that contain the data strings in the SRR.

Consider an SRR r consisting of data strings $<d_1, d_2, \ldots, d_i, d_{i+1}, \ldots, d_m>$, while each data string belongs to cluster $c_1, c_2, \ldots, c_i, c_{i+1}, \ldots, c_m$ respectively. Let's further assume the ideal case that each cluster c_i, $1 \leq i \leq m$, represents a data slot ds_i, $1 \leq i \leq m$, in the SRR's template graph. Thus any adjacent data string pair $<d_i, d_{i+1}>$ in r implies that there exists an edge between ds_i and ds_{i+1} in the SRR's template graph. The SRR r implies that there is a path $ds_1, ds_2, \ldots, ds_i, ds_{+1}, \ldots, ds_m$ in the template graph.

Due to the optional and disjunctive data units, a single SRR can only provide a partial view of the template graph. However, the combination of multiple SRRs with all possible unit variations can provide the complete view of the SRR template graph.

Because of the existence of false SRRs (outliers), errors in the DTD, TTD and the data string clustering algorithm, not all SRRs provide the correct information about the template. We design an algorithm to let SRRs vote for the template graph. The idea is that as long as a majority of SRRs provide correct information, the algorithm will generate the correct template graph. Figure 7 shows an example of the voting process.

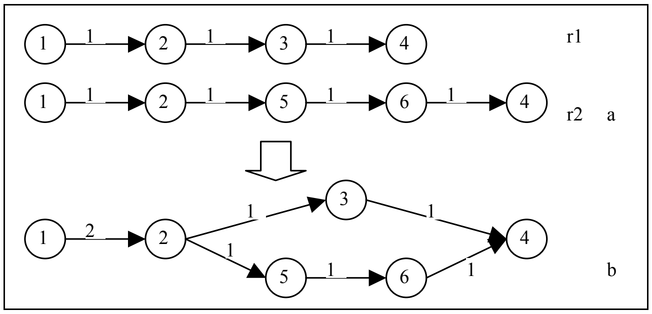


Figure 7. Two SRRs vote for template graph

Figure 8 shows the template graph voting algorithm. A vertex in a template graph represents a data slot in the SRR template. We consider each data string cluster as a data slots initially, and then let all SRRs vote for the relationships between the data slots. In essence, an SRR template graph represents the data slots and their relationship in SRRs.

```
Algorithm $Build\_Temp\_Graph(S_{SRR})$
  $G(V, E) \leftarrow \Phi;$
  for $r \in S_{SRR}$ do
    for $<d_i, d_{i+1}> \subset r$ do
      $c_i \leftarrow$ the cluster $d_i$ belongs to;
      $c_{i+1} \leftarrow$ the cluster $d_{i+1}$ belongs to;
      $v_i \leftarrow$ the template node corresponding to $c_i$;
      If $v_i$ = null
        Create $v_i$ and add $v_i$ to $V$ ;
      $v_{i+1} \leftarrow$ the cluster $v_{i+1}$ belongs to $c_{i+1}$;
      if $v_{i+1}$ = null
        Create $v_{i+1}$ and add $v_{i+1}$ to $V$;
      $e \leftarrow$ the edge from $v_i$ to $v_{i+1}$;
      if $e$ = null
        Create $e$ and add $e$ to $E$;
      $W(e) \leftarrow W(e) + 1$;
    end for;
  end for;
  for $e \in E$ do
    if $W(e) < T_e$
      remove $e$ from $E$;
  end for;
  for $v \in V$ do
    if $W(v) < T_v$
      remove $v$ from $V$;
  end for;
  break cycles in $G$ by removing edges with lowest weight in
    cycles;
  return $G$;
```

Figure 8. Template Graph Voting Algorithm

At first we have an empty template graph. Then for each SRR, we check its consecutive data string pairs d_i and d_{i+1}. The clustering algorithm in section 4.2 ensures that two data string pairs belong to two different clusters. We get the data string clusters c_i and c_{i+1} that d_i and d_{i+1} belong to. The next step is to ensure that there are vertexes v_i and v_{i+1} in the template graph that correspond to the two data clusters, and the weight of the edge from v_i to v_{i+1} is increased by 1.

After all SRRs voted for the template graph, we check the graph to remove low weight edges and vertexes. They are considered as false because of their low supports. In some cases there might be cycles in the generated template graph, which is contradictory to the assumption that a template graph is a DAG. We break the detected cycles by removing the edges with the lowest weights in them. The outcome is the SRR template graph.

5. EXPERIMENTS

We implemented a prototype SRR template mining system based on the proposed method. The system contains a DTD trainer, a TTD trainer and an SRR template builder. The DTD trainer and TTD trainer are used to train the neural network based Decorative Tag Detector and the Template Text Detector. As introduced in section 3.3, The SRR template builder consists of an SRR extraction module, a DTD, a data string clustering module, a TTD and a template generating module. With pre-trained DTD and TTD, the SRR template builder is a fully automatic template building system. It takes a set of result pages that contain SRRs as input; the output is a template graph that can be used to extract the detailed data units from the SRRs. The prototype system shows that the proposed method is both effective and efficient. On a laptop with a Pentium M 1.3G processor, it can build the SRR template from a set of 10 result pages within 5 to 30 seconds.

We present the experimental results in two parts. Section 5.1 presents the performance of DTD and TTD as neural network classifiers; section 5.2 presents the performance of the SRR template builder with the DTD and TTD embedded. Accordingly, the testbed consists of two parts: part 1 consists of sample result pages downloaded from 57 search engines, while part 2 consists of sample pages downloaded from 50 search engines. These two sets of search engines are disjoint. For each search engine, we collect 5 to 10 sample result pages by manually submitting probe queries. Since a typical search engine result page contains 10 or more SRRs, we usually have more than 50 SRRs to collect statistic features for tags and high-frequency texts for DTD and TTD. We use the first 57 search engines to train DTD and TTD.

5.1 Testing of DTD and TTD

We implement the DTD and TTD based on Joone engine 1.2.1 [29]. Both DTD and TTD consist of three sigmoid layers. Training samples for DTD are HTML tags extracted from the tag forests of SRRs. We use the method introduced in section 4.1.2 to collect the sample tags and their features. A total of 923 sample tags are collected from the 57 search engines in testbed part 1. The training samples for TTD are high-frequency texts. They are collected by the method introduced in section 4.3.1. From the result pages of the 57 search engines, a total of 943 high-frequency texts are collected as the training samples for TTD.

We use 5-fold cross validation to measure the performance of DTD and TTD. The results Table 3 show that the proposed features and the design of the neural network classifiers can effectively identify the decorative tags as well as template texts.

Table 3. Performances of DTD and TTD

Fold	1	2	3	4	5	Average
DTD	0.913	0.983	0.978	0.951	0.962	0.957
TTD	0.936	0.958	0.973	0.941	0.936	0.949

5.2 Testing of SRR Template Builder

Before we test the SRR template builder on testbed part 2, we train the DTD and TTD on all samples collected from testbed part 1. The testing of SRR template builder on each search engine is fully automatic, without any human involvement.

The 50 search engines in testbed part 2 consist of a wide variety of search engines: 20 e-commerce search engines, which are usually Web databases that search structured data, 15 news search engines, 10 medical search engines and 5 others. Among the 50 search engines, only three arrange their results in tabular format.

We use a method similar to EXALG [2] to measure the performance of the SRR template builder. We first manually check the SRRs of each search engine to identify the data units in the SRRs. Then the data slots in the generated template graph are checked against the manually identified data units. We count the numbers of data slots that match and mismatch the data units. Finally the recall and precision of data units are used to measure the performance of the SRR template builder.

The manual identification of data units from semi-structured SRR is a somewhat subjective process. We follow the following basic guidelines in doing the identification. First we do not extract data that is presented in the attributes of HTML tags, such as the HREF attribute of a hyper link. Thus the functional links like “Cached” and “Similar pages” are considered as templates instead of data. Second, a date is considered as a single data unit instead of a composition of month, day and year. Third, a complete hyper link is considered as a single data unit, etc. Another thing worth noting is that we don’t differentiate the optional and disjunctive data units from other data units when computing the statistics.

Table 4. Performance of SRR Template Builder

Category	Actual	Extracted	Correct	Recall	Precision
E-com.	133	134	120	0.902	0.896
News	61	69	59	0.967	0.855
Medical	43	43	39	0.907	0.907
Misc.	24	27	22	0.917	0.815
Total	261	273	240	0.920	0.889

Table 4 shows the summary of the test of the SRR template builder. The column with header Category lists the search engine categories. The column with header Actual, Extracted and Correct list the actually number of data units in the SRRs for each search engine, the number of data slots in the generated SRR template for each search engine, and the number of data slots in SRR template that match the corresponding data units for each search engine, respectively. The last two columns list the recall and precision. Each row presents the performance of the SRR template builder on a search engine category, except for the last row, which presents the performance over all 50 search engines.

We can see that the proposed method is generally effective, with a recall of 92% and precision of 88.9%. We compare our performance with EXALG [2], which is currently the best automatic web data template extraction tool in the literature. EXALG correctly extracted 80% of data units from a collection of 45 web sites. Our recall is significantly higher although different datasets are used in these experiments. EXALG didn’t report its precision.

The fairly even performance among the search engines in the four categories shows the proposed method works almost equally well on a wide varieties of web databases and search engines.

We missed 8% of data units during the test. The major reason is that some data units have almost exactly the same values in all SRRs on result pages of some search engines (for example, there are two sets of data units in the SRRs of officedepot.com, one set is for "units" and the other is for "availability"; on all result pages we collected, the value of "units" is always "each", and the value of "availability" is always "In Stock"). Thus those data units were incorrectly identified as templates by the system. Another reason is the failure of TTD, which sometimes incorrectly put two or more data units into a single data slot. Some false data slots are also generated, which prevented a higher precision. This is mostly caused by the failure of DTD, which may sometimes incorrectly split a data slot into more data slots. This seems to imply that the 57 search engines that were used to train DTD cannot represent the 50 search engines in part 2 very well. In fact, the majority of search engines in testbed part 1 are Google like document search engines, while the search engines in part 2 have a wider variety. We expect a larger training set would improve the performance.

6. CONCLUSION AND FUTURE WORK

Knowing the precise template of SRRs can greatly help applications that need to interact with search engines. For example, the template can help extract data units from the SRRs, which is a critical step in SRR annotation. In this paper, we proposed a new technique to extract the precise SRR template for any search engine automatically. As mentioned in the introduction section, this solution has quite a few novel features. For the first time, we systematically identified the factors that make the extraction of SRR template difficult and proposed novel solutions to address them explicitly and specifically. Even though our solution involves training in order to build the decorative tag detector and the template text detector, the training only needs to be carried out once and the trained detectors can be applied to any search engines, including those that are not used in the training. Our experimental results indicate that our solution is significantly more accurate than an existing state-of-the-art solution.

Our experiment revealed that our statistics-based solution does have an inherent weakness in dealing with attributes that have the same or nearly the same values (data units) in all SRRs. These data units will be mistakenly recognized as template texts. We plan to investigate how to overcome this weakness. One idea is to supplement our solution with a different solution. For example, one phenomenon we observed about these data units is that they often have their annotation labels next to them (usually in front of them) in the SRRs. In this case, these labels in all SRRs are also identical, and if the data units are not completely identical, these labels (they are template texts) can be recognized as the labels for the data units following them by the common prefix annotator proposed in [17, 22] and consequently the data units are also recognized correctly. But this still does not solve the problem when all the data units are identical so new solutions are still needed.

7. ACKNOWLEDGMENTS

This work is supported in part by the following NSF grants: IIS-0414981, IIS-0414939 and CNS-0454298.

8. REFERENCES

[1] B. Adelberg. NoDoSE – A Tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents. ACM SIGMOD Conference, 1998.

[2] A. Arasu, H. Garcia-Molina. Extracting Structured Data from Web Pages. ACM SIGMOD Conference, 2003.

[3] R. Baumgartner, S. Flesca and G. Gottlob. Visual Web Information Extraction with Lixto. VLDB Conference, 2001.

[4] D. Buttler, L. Liu, C. Pu. A Fully Automated Object Extraction System for the World Wide Web. ICDCS 2001.

[5] C. Chang, S. Lui. IEPAD: Information Extraction based on Pattern Discovery. WWW Conference, 2001.

[6] C. Chang, M. Kayed, M. R. Girgis and K. F. Shaalan. A Survey of Web Information Extraction Systems. IEEE TKDE, Vol 18, No. 10, Oct. 2006.

[7] V. Crescenzi, G. Mecca, P. Merialdo. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. VLDB Conference, 2001.

[8] D. Embley, Y. Jiang, Y. Ng. Record-Boundary Discovery in Web Documents. ACM SIGMOD Conference, 1999.

[9] A. Hogue and D. Karger. Thresher: Automating the Unwrapping of Semantic Content from the World Wide Web. WWW Conference, 2005.

[10] C. Hsu and M. Dung. Generating Finite-State Transducers for Semi-structured Data Extraction from the Web. Information Systems. 23(8): 521-538, 1998.

[11] U. Irmak, and T. Suel. Interactive Wrapper Generation with Minimal User Effort. WWW Conference, 2006.

[12] N. Kushmerick, D. Weld, R. Doorenbos. Wrapper Induction for Information Extraction. Int'l Joint Conf. on AI, 1997.

[13] A. Laender, B. Ribeiro-Neto, A. da Silva, J. Teixeira. A Brief Survey of Web Data Extraction Tools. ACM SIGMOD Record, 31(2), 2002.

[14] B. Liu, R. Grossman and Y. Zhai. Mining Data Records in Web Pages. ACM SIGKDD Conference, 2003.

[15] B. Liu and Y. Zhai. NET – A System for Extracting Web Data from Flat and Nested Data Records. WISE Conference, 2005.

[16] L. Liu, C. Pu and W. Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. IEEE ICDE, 2000.

[17] Y. Lu, H. He, H. Zhao, W. Meng, C. Yu. Annotating Structured Data of the Deep Web. IEEE ICDE, 2007.

[18] W. Meng, C. Yu, K. Liu. Building Efficient and Effective Metasearch Engines. ACM Compu. Surv., 34(1), 2002.

[19] I. Muslea, S. Minton, C. Knoblock. A Hierarchical Approach to Wrapper Induction. Int'l Conf. on Auton. Agents, 1999.

[20] S. Raghavan, H. Garcia-Molina. Crawling the Hidden Web. VLDB Conference, Italy, 2001.

[21] K. Simon, and G. Lausen. ViPER: Augmenting Automatic Information Extraction with Visual Perceptions. ACM CIKM 2005.

[22] J. Wang, F. Lochovsky. Data Extraction and Label Assignment for Web Databases. WWW Conference, 2003.

[23] Z. Wu, V. Raghavan et al. Towards Automatic Incorporation of Search Engines into a Large-Scale Metasearch Engine. IEEE/WIC WI-2003 Conference, 2003.

[24] G. Yang, I. V. Ramakrishnan and M. Kifer. On the Complexity of Schema Inference from Web Pages in the Presence of Nullable Data Attributes. ACM CIKM, 2003.

[25] Y. Zhai, B. Liu. Web Data Extraction Based on Partial Tree Alignment. WWW Conference, 2005.

[26] Y. Zhai, B. Liu. Extracting Web Data Using Instance-Based Learning. WISE Conference, 2005.

[27] H. Zhao, W. Meng, Z. Wu, V. Raghavan, C. Yu. Fully Automatic Wrapper Generation for Search Engines. WWW Conference, 2005.

[28] J. Zhu, Z. Nie, J. Wen, B. Zhang, W. Ma. Simultaneous Record Detection and Attribute Labeling in Web Data Extraction. ACM SIGKDD Conference, 2006.

[29] http://www.jooneworld.com/.