

A Generalized Tree Matching Algorithm Considering Nested Lists for Web Data Extraction

Nitin Jindal and Bing Liu
 Department of Computer Science
 University of Illinois at Chicago
 851 S. Morgan St, Chicago, IL 60607
 nitin.jindal@gmail.com
 liub@cs.uic.edu

Abstract

This paper studies structured data extraction from Web pages. One of the effective methods is tree matching, which can detect template patterns from web pages used for extraction. However, one major limitation of existing tree matching algorithms is their inability to deal with embedded lists with repeated patterns. In the Web context, lists are everywhere, e.g., lists of products, jobs and publications. Due to the fact that lists in trees may have different lengths, the match score of the trees can be very low although they follow exactly the same template pattern. To make the matter worse, a list can have nested lists in it at any level. To solve this problem, existing research uses various heuristics to detect candidate lists first and then applies tree matching to generate data extraction patterns. This paper proposes a generalized tree matching algorithm by extending an existing tree matching algorithm with the ability to handle nested lists through a novel grammar generation algorithm. To the best of our knowledge, this is the first tree matching algorithm that is able to consider lists. In addition, it is well-known that there are two problem formulations for Web data extraction: (1) pattern generation based on multiple pages following the same template, and (2) pattern generation based on a single page containing lists of data instances following the same templates (each list may use a different template). These two problems are currently solved using different algorithms. The proposed (single) algorithm is able to solve both problems effectively. Extensive experiments show that the new algorithm outperforms the state-of-the-art existing systems for both problems considerably.

Keywords: We data extraction, Web mining

1. Introduction

Structured data extraction or wrapper generation on the Web is an important problem with a wide range of applications. Structured data are typically descriptions of objects retrieved from underlying databases and displayed in Web pages following some fixed templates. Examples of such objects are products, job listings, publications, etc. Extraction of such data enables one to integrate data/information from multiple Web sites to provide value-added services, e.g., comparative shopping, object search, and information integration. Figure 1 and Figure 2 show two examples of structured data objects. Figure 1 is a Web

page segment containing a list of two products. The description of each product is called a *data record*. Such a page is called a *list page*. Figure 2 shows a page segment



Figure 1. A list of data records



Figure 2. An example detail page

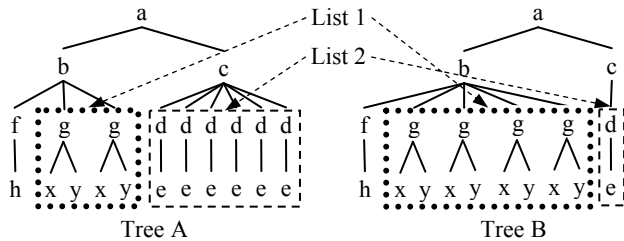


Figure 3: Two trees with lists

containing the detailed description of one product. Such a page is called a *detail page*. The objective of data extraction is to build wrappers (*data extraction programs*) to extract data from such types of pages.

There are two main approaches to wrapper generation, i.e., *wrapper induction* and *automated data extraction*. Wrapper induction uses supervised learning to learn data extraction rules from manually labeled training examples [13, 14, 19]. The disadvantages of wrapper induction are (1) the time-consuming manual labeling process and (2) the difficulty of wrapper maintenance [13, 14, 15, 19]. Due to the manual labeling effort, it is hard to extract data from a huge number of sites as each site has its own templates and requires separate manual labeling for wrapper learning. Wrapper maintenance is also a major issue because whenever a site changes the wrappers built for the site become obsolete. Due to these shortcomings, researchers have studied automated wrapper generation using unsupervised pattern mining [17]. Automated extraction is possible because most Web data objects follow fixed templates. Discovering such templates or patterns enables the system to perform extraction automatically.

There are two problem formulations [6, 17] for automated extraction. In the first formulation, the system is given multiple pages following the same template. This formulation is particularly suitable for data extraction from detail pages (Figure 2) because it is not possible to detect patterns from a single example. However, with multiple pages patterns can be discovered for extraction. In the second formulation, the system is given a single page with multiple data records (a list page) (Figure 1). The system detects the data records and extracts data items from them. Unsupervised methods are possible because of the repeated template patterns used by a list of data records. Our proposed algorithm is able to solve both problems.

Existing approaches to solving these problems are based on string matching or tree matching [3, 18, 29, 32]. However, both these matching techniques are not able to deal with nested lists. Current extraction algorithms either do not allow nested lists or have some heuristics to detect lists [3, 18, 29, 32]. Note that tree matching is used because HTML codes can be naturally represented as trees. [32] has shown that string matching techniques do not work well due to extensive use of only a few table related HTML tags in Web pages which can result in many wrong matches. Tree matching is more suitable because trees reflect the structures and layouts of the pages naturally and thus can eliminate most incorrect matches.

To see why lists cause problems, let us use an example. Suppose we have the two trees to be matched in Figure 3. Tree A has 23 nodes and tree B has 19 nodes. Each tree has two repeated lists (in dash-lined boxes). From a pattern discovery point of view, the two trees follow exactly the same template/pattern, and thus should be matched completely. However, current tree matching algorithms can only match 13 nodes. Two sub-trees starting from nodes g 's in tree B cannot be matched, and 5 sub-trees starting from nodes d 's in tree A cannot be matched.

This paper deals with this problem and makes two main research contributions:

1. It generalizes a tree matching algorithm so that it can consider lists. This is a challenging problem because list elements may not be exactly the same. Thus it is very difficult to detect the boundary of each list element to know that a list exists. To solve the problem, a special grammar form is identified for nested lists, and a novel grammar generation method is proposed to detect lists. This list handling ability is integrated into a tree matching algorithm (called Simple Tree Matching (STM)) [31]. This results in a *generalized tree matching* algorithm, called G-STM (Generalized Simple Tree Matching). To our knowledge, this is the first tree matching algorithm that is able to consider lists in its matching process.
2. For data extraction, this (single) algorithm is able to solve both extraction problems effectively, which is a major advantage of the proposed algorithm. Existing methods all use different algorithms to solve the two problems.

Extensive experiments show that G-STM outperforms the state-of-the-art existing systems for both types of problems. The system has been tested in a commercial setting and is in the processing being licensed to a commercial company.

The paper is organized as follows: Section 2 discusses the related work. Section 3 defines the data extraction problems that we are interested in solving using the proposed algorithm. Section 4 presents the proposed G-STM algorithm. Section 5 gives the evaluation results, and Section 6 concludes the paper.

2. Related Work

This work is related to wrapper induction and automated data extraction. Wrapper induction uses supervised learning to learn data extraction rules from a set of manually labeled examples [2, 4, 5, 11, 12, 18, 21, 22, 25, 35]. We have discussed issues with wrapper induction in the introduction section. Our technique requires no human labeling. It mines templates and extracts data automatically.

Finding a template from multiple input pages for data extraction was first studied in [6, 7], which presents the Roadrunner system. The algorithm is based on a heuristic tag-by-tag match method to infer a regular expression pattern. [1] presents the EXALG system, which is based on a frequency approach. Our proposed algorithm integrates an optimized tree matching method with grammar generation to solve the problem. We will show its superior performance compared to both Roadrunner and EXALG.

Regarding automated extraction based on a single list page, the following systems identify data records and extract data items from them: IEPAD [3], Dela [29], the system in [15], DEPTA [32] and NET [16]. Various heuristics are used to identify lists in a page. Their abilities to handle nested lists are limited. There are also several

systems that only segment lists into individual data records, e.g., MDR [18], [21], [35], Viper [26], MSE [34], ViNTs [33]. However, they do not identify or extract data items from these data records. Our proposed algorithm performs both functions.

In [23, 28], tree matching is used to extract the main content of news pages. However, the tree matching algorithm in [23] only finds lists for leaf nodes (relies on exact match when grouping sub-trees into lists). [28] does not consider lists. Using visual/rendering information (obtained from a Web browser) of Web pages to help extraction has also been studied by several researchers, e.g., [26, 33, 34, 35]. These techniques mainly consist of heuristic rules, which can exploit visual features in the process of identifying data records or lists.

In grammar generation, results from the learning of regular expressions [9, 21] show that the problem of finding a natural list is intractable in general. However, since we are interested in a particular form of grammar, we will show that it is achievable in polynomial time. In fact, our method is linear with an assumption. We have not seen any Web page that does not satisfy the assumption.

In our earlier work NET [16], a heuristic approach was proposed to find nested lists. The algorithm was modeled on MDR [18], which is based on tree traversal (not tree matching) and node comparison during traversal. However, NET works bottom-up or post-order, while MDR works top-down. Thus, MDR has difficulty in finding nested data records, while NET can find them. In [16], a grammar-based method was suggested to improve the algorithm in NET for detecting data records in each iteration of tree traversal. However, the method was not tested. The proposed method in this paper integrates the grammar based approach and tree matching to produce a brand new algorithm, which is a more principled approach. This not only produces a new data extraction algorithm, but also a generalized tree matching algorithm which can consider lists. No existing tree matching algorithm handles list. Our data extraction experimental results given in Section 5 also show that this generalized tree matching algorithm outperforms the state-of-the-art existing data extraction algorithms dramatically.

3. Problem Statement

Structured data can be modeled as *nested relations*, which are typed objects allowing nested sets and tuples. The types are defined as follows [6, 17]:

- There is a set of *basic types*, $B = \{B_1, B_2, \dots, B_k\}$. Each B_i is an atomic type, and its domain, denoted by $dom(B_i)$, is a set of constants;
- If T_1, T_2, \dots, T_n are basic or set types, then $[T_1, T_2, \dots, T_n]$ is a *tuple type* with the domain $dom([T_1, T_2, \dots, T_n]) = \{[v_1, v_2, \dots, v_n] \mid v_i \in dom(T_i)\}$;
- If T is a tuple type, then $\{T\}$ is a *set type* with the domain $dom(\{T\})$ being the power set of $dom(T)$.

A basic type B_i is analogous to the type of an attribute in relational databases, e.g., string and int. In the context of the Web, B_i is usually a text string, image-file, etc.

An instance of a tuple type is called a *data record*. An instance of a set type is called a *list*. A data record is simply the description of an object. For example, there are two data records (descriptions of the two cameras) in Figure 1. There is no nesting in Figure 1. An example nested record is shown in Figure 4. The first data record “Canning Jars by Ball” have two nested records, two different sizes (“8-oz” and “1-pt”) with different prices (\$4.95 and \$5.95).

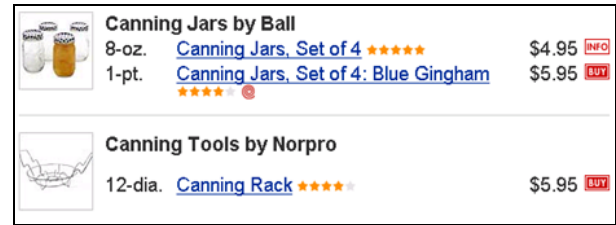


Figure 4. An example nested type

In a Web page, all the data are encoded using HTML tags. The two data extraction problem formulations are:

Problem 1: Extraction based on multiple pages

Input: A collection W of k HTML strings, which encodes k instances of the same type.

Output: A type σ , and a collection C of instances of type σ , such that there is a HTML encoding enc such that $enc: C \rightarrow W$ is a bijection.

Intuitively, the input is a set of HTML strings representing a set of given pages, which follow the same template (the same type) encoded in HTML. Our task is to mine the template for use in data extraction. In this work, the pattern/template is represented as a tree.

Problem 2: Extraction based on a single list page

Input: A single HTML string S , which contains k non-overlapping substrings s_1, s_2, \dots, s_k with each s_i encoding an instance of a set type. That is, each s_i contains a collection W_i of m_i (≥ 2) non-overlapping sub-substrings encoding m_i instances of a tuple type.

Output: k tuple types $\sigma_1, \sigma_2, \dots, \sigma_k$ and k collections C_1, C_2, \dots, C_k of instances of the tuple types such that for each collection C_i there is a HTML encoding function enc_i such that $enc_i: C_i \rightarrow W_i$ is a bijection.

Note that the k tuple types do not have to be all distinctly different as it is possible that different areas of a page may use the same template but contain different data.

Intuitively, the input string (a Web page) has k non-overlapping substrings s_1, s_2, \dots, s_k represent k lists in the page. Each list consists of m_i data records. Our task is to identify each list, mine the pattern/template from the list,

and extract data items from every data record of the list. Note that Figure 1 has only one list, but a general page can have multiple lists of data records.

4. The Proposed G-STM Algorithm

As we mentioned earlier, tree matching is an important method for finding data extraction patterns. Since existing tree matching algorithms cannot handle lists, it causes many complications. We now add the list handling capability to the popular tree matching algorithm, Simple Tree Matching (STM) [31], which has been shown to work very well for trees with no lists [32].

4.1 Simple Tree Matching

The general tree matching is defined as follows [27]: Let X be a tree and $X[i]$ be the i th node of X in a preorder walk of the tree. A *mapping* M between a tree A of size n_1 and a tree B of size n_2 is a set of ordered pairs (i, j) , one from each tree, satisfying the following conditions for all $(i_1, j_1), (i_2, j_2) \in M$:

- (1) $i_1 = i_2$ iff $j_1 = j_2$;
- (2) $A[i_1]$ is on the left of $A[i_2]$ iff $B[j_1]$ is on the left of $B[j_2]$;
- (3) $A[i_1]$ is an ancestor of $A[i_2]$ iff $B[j_1]$ is an ancestor of $B[j_2]$.

Intuitively, the definition requires that each node appears no more than once in a mapping and the order among siblings and the hierarchical relation among nodes are preserved. In this most general setting, mapping can cross levels, and replacements are allowed. Replacement means two different nodes can match with a cost incurred. STM is a restricted tree mapping algorithm, in which no node replacement or level crossing are allowed. The aim of STM is to find the maximum matching between two trees.

Let A and B be two trees, and $i \in A$ and $j \in B$ be two nodes in A and B respectively. A *matching* between two trees is defined to be a mapping M such that, for every pair $(i, j) \in M$ where i and j are non-root nodes, $(\text{parent}(i), \text{parent}(j)) \in M$. A *maximum matching* is a matching with the maximum number of pairs.

Let $A = R_A: \langle A_1, \dots, A_k \rangle$ and $B = R_B: \langle B_1, \dots, B_n \rangle$ be two trees, where R_A and R_B are the roots of A and B , and A_i and B_j are the i th and j th first-level sub-trees of A and B respectively. Let $W(A, B)$ be the number of pairs in the maximum matching of trees A and B . If R_A and R_B contain identical symbols, the maximum matching between A and B (i.e., $W(A, B)$) is $m(\langle A_1, \dots, A_k \rangle, \langle B_1, \dots, B_n \rangle) + 1$, where $m(\langle A_1, \dots, A_k \rangle, \langle B_1, \dots, B_n \rangle)$ is the number of pairs in the maximum matching of $\langle A_1, \dots, A_k \rangle$ and $\langle B_1, \dots, B_n \rangle$. If $R_A \neq R_B$, $W(A, B) = 0$.

$W(A, B)$ is defined as follows:

$$W(A, B) = \begin{cases} 0 & \text{if } R_A \neq R_B \\ m(\langle A_1, \dots, A_k \rangle, \langle B_1, \dots, B_n \rangle) + 1 & \text{otherwise} \end{cases}$$

$$m(\langle \rangle, \langle \rangle) = 0$$

Algorithm: G-STM(A, B)

```

1  if the roots of trees  $A$  and  $B$  contain different symbols
   then
2    return  $(0, A.nodes, B.nodes)$ 
3  else
4     $k \leftarrow$  the number of first-level sub-trees of  $A$ ;
5     $n \leftarrow$  the number of first-level sub-trees of  $B$ ;
6    Initialization:  $m[i, 0] \leftarrow 0$  for  $i = 0, \dots, k$ ;
7                    $m[0, j] \leftarrow 0$  for  $j = 0, \dots, n$ ;
8  for  $i = 1$  to  $k$  do
9    for  $j = 1$  to  $n$  do
10    $W[i, j] \leftarrow$  G-STM( $A_i, B_j$ )
11   $(W, nodes_A, nodes_B) \leftarrow$  Detect-Lists( $W, A, B$ );
12  for  $i = 1$  to  $k$  do
13   for  $j = 1$  to  $n$  do
14    $m[i, j] \leftarrow$  MAX( $m[i, j-1], m[i-1, j],$ 
                         $m[i-1, j-1] + W[i, j].score$ )
15  return  $(m[k, n] + 1, nodes_A, nodes_B)$ 

```

Figure 5: The proposed G-STM algorithm

$$m(s, \langle \rangle) = m(\langle \rangle, s) = 0$$

$$m(\langle A_1, \dots, A_k \rangle, \langle B_1, \dots, B_n \rangle) =$$

$$\max(m(\langle A_1, \dots, A_{k-1} \rangle, \langle B_1, \dots, B_{n-1} \rangle) + W(A_k, B_n),$$

$$m(\langle A_1, \dots, A_k \rangle, \langle B_1, \dots, B_{n-1} \rangle),$$

$$m(\langle A_1, \dots, A_{k-1} \rangle, \langle B_1, \dots, B_n \rangle)).$$

In the context of the Web, A and B are the DOM trees to be matched. R_A and R_B are the root nodes of the two trees represented by their respective html tags. Clearly, this is a dynamic programming formulation. If there are lists, their elements are treated just like any other nodes, which is problematic. Below, we present the proposed new method which includes lists handling in the algorithm.

4.2 The Proposed G-STM Algorithm

The proposed algorithm G-STM which handles nested lists is given in Figure 5, where A and B are trees to be matched. It follows the formulation of STM above, but detects lists at each step. In order to detect lists, for every pair of nodes A and B we output a tuple of $(score, nodes_A, nodes_B)$, where $score$ is their match score and $nodes_A$ and $nodes_B$ are the number of nodes (or sizes) of A and B respectively. These tree sizes are needed in detecting lists, which needs to use normalized match scores. Due to the use of matrix notations, i and j represents the i th and j th children sub-trees of A and B respectively.

Lines 1-2 in Figure 5 compare the labels (tag names) of the root nodes of the two trees A and B and return 0 if they are different. Lines 3-7 initialize variables. Lines 8-10 run G-STM on each pair of the first level sub-trees, and stores the scores and the number of nodes tuple in the score matrix W . Each cell $W[i, j]$ of the W matrix contains three values, the score, the tree size of i th child sub-tree of A (A_i) and the tree size of j th child sub-tree of B (B_j). As we will see for a node which contains lists, the tree size will be

Function: Detect-Lists(W, A, B)

```

1  $k \leftarrow$  the number of rows of  $W$ ;
2  $n \leftarrow$  the number of columns of  $W$ ;
3 Define a temporary matrix  $W_{norm}$ , of  $k$  rows and  $n$ 
  columns where  $W_{norm}[i, j]$  is normalized score of  $i$ th
  sub-tree of  $A$  (denoted by  $S_{A_i}$ ) and  $j$ th sub-tree of  $B$ 
  (denoted by  $S_{B_j}$ );
4  $W_{norm}[i, j] \leftarrow$ 
   $W[i, j].score / \text{MAX}(W[i, j].nodes_1, W[i, j].nodes_2)$ ,
  for  $i = 1, \dots, k$  and  $j = 1 \dots, n$ ;
5 for  $i = 1$  to  $k$  do
6   for  $j = 1$  to  $n$  do
7      $score_{norm} \leftarrow W_{norm}[i, j]$ ;
8      $max_{A_i} \leftarrow \text{MAX}(W_{norm}[i, 1 \dots n])$ ;
9      $max_{B_j} \leftarrow \text{MAX}(W_{norm}[1 \dots k, j])$ ;
    // $max_{A_i} (max_{B_j})$  is the maximum normalized score
    sub-tree  $S_{A_i}$  ( $S_{B_j}$ ) has with any child sub-tree of
    tree  $B$  ( $A$ )
10    if  $score_{norm} > \tau_1$  AND  $score_{norm}/max_{A_i} > \tau_2$ 
      AND  $score_{norm}/max_{B_j} > \tau_2$  then
11       $S_{A_i}$  and  $S_{B_j}$  match
12      Assign a distinctive symbol to each pair of
      matched sub-trees.
13 Each unmatched sub-tree is assigned a unique symbol;
14 Let  $String_1$  and  $String_2$  be the strings generated for tree
   $A$  and tree  $B$  after assigning the symbols.
15  $g_A \leftarrow \text{Grammar-Generation}(String_1)$ ;
16  $g_B \leftarrow \text{Grammar-Generation}(String_2)$ ;
17 if  $g_A$  matches  $g_B$  then
  //this is carried out only based on the list parts
  (represented by “+” in regular expressions), i.e.,
  with other parts removed.
18 return ( $W, nodes_A, nodes_B$ )  $\leftarrow$  UpdateW( $W, g_A, g_B$ )
19 else return ( $W, A.nodes, B.nodes$ )

```

Figure 6: Algorithm for detecting lists

adjusted by only keeping one instance/element of the list. Function Detect-Lists in Line 11 detects lists by generating regular expression grammars for the child nodes of both root nodes. The grammars generated help identify lists in the two trees. It also updates the score matrix W based on the detected lists. Lines 12-15 compute the matching matrix M based on dynamic programming.

4.3 The Detect-Lists Function

The *Detect-Lists* function is given in Figure 6. The basic idea is the following: Since W matrix contains the pair-wise match scores of all the child sub-trees of tree A and tree B , they can obviously be used to detect lists because a list is a set of records which match (or are similar to) each other. From the W matrix, we know which child sub-tree from A matches which child sub-tree from B . Those matched sub-trees are replaced with the same unique

symbol. The not-matched sub-trees are given different symbols. This results in two strings of symbols, one for the sub-trees of A and one for the sub-trees of B . These strings are then used to generate grammars, which help find lists.

Lines 1-14 produce a string from the child sub-trees of each tree and lines 15-16 call the Grammar-Generation function to generate grammars. Lines 12-13 assign each group of matched sub-trees a unique symbol, and each unmatched sub-tree a different symbol. What is considered a match is discussed below and controlled by two empirical thresholds. Appending the symbols together following the order of the child sub-trees gives us a string, which is used in grammar generation. We use normalized scores (line 4) to find the matched sub-trees as normalized scores reduce discrepancies resulted from different tree sizes in matching. Matched sub-trees are found using W_{norm} based on the following criteria (line 10):

1. The match $score_{norm}$ of two sub-trees is larger than some threshold value τ_1 (set to 50%). Note that to match primitive list nodes like $\langle li \rangle$, $\langle dd \rangle$, $\langle dt \rangle$, etc the normalized score is 50%. This sets the upper bound on τ_1 . Since upper bound is so tight, τ_1 is fixed to 50%. Unfortunately, this low threshold also causes many non-lists to become lists. To prevent that from happening, we use the second condition below.
2. The ratio of $score_{norm}$ and the maximum normalized score of the two sub-trees (max_{A_i} and max_{B_j}) has to be larger than a threshold value τ_2 (set to 70%). This criterion is based on the observation that if a sub-tree forms a list with other sub-trees, then its matched score should be within some range of its matched score with the sub-tree with which it has the maximum match. It helps offset the low threshold set for the first criterion.

These thresholds are needed because although the lists may follow the same template, there are usually many optional items in which prevent complete match. We also note that the matches here are transitive, which can result in some errors when they are not. However, the errors are rare as our strong experimental results show.

We also note that in lines 8-9 we do not repeatedly search the maximum W_{norm} every time. After strings are produced, lines 15-16 call *Grammar-Generation* to generate grammars from the strings. If the two grammars from the two trees match, a list is found. Here by “match” we mean that the two grammars are the same after removing those non-list parts. The list is represented with $(D)^+$ in regular expressions (see section 4.4), where D represents a data record pattern. We will explain this further in the next sub-section, where we define the grammar that we need.

Lines 17-18 updates W . Since some lists are found, we do not want to pass the number of list nodes or the raw match scores up due to different list lengths which can make the upper level match impossible. Then, we want to compress/collapse each list to only one element/record, which means we will update the value in the W matrix. This will be discussed in Section 4.5.

Function: Grammar-Generation(*String*)

```

1 Initialize a data structure for NFA  $N = (Q, \Sigma, \delta, q_0, F)$ ,
  where  $Q$  is the set of states,  $\Sigma$  is the symbol set containing
  all symbols appeared in String,  $\delta$  is the transition relation
  that is a partial function from  $Q \times (\Sigma \cup \{\varepsilon\})$  to  $Q$ , and  $F$ 
  is the set of accept states,
   $Q \leftarrow \{q_0\}$  ( $q_0$  is the start state),  $\delta \leftarrow \emptyset$  and  $F \leftarrow \emptyset$ ;
2  $q_c \leftarrow q_0$ ; //  $q_c$  is the current state
3 for each symbol  $s$  in String in sequence do
4   if  $\exists$  a transition  $\delta(q_c, s) = q_n$  then
5      $q_c \leftarrow q_n$  // transit to the next state;
6   else if  $\exists \delta(q_i, s) = q_j$ , where  $q_i, q_j \in Q$  then
7     if  $\exists \delta(q_f, \varepsilon) = q_i$ , where  $\delta(q_i, s) = q_j$  and  $f \geq c$ 
8       then
9         TransitTo( $q_c, q_f$ )
10      else TransitTo( $q_c, q_i$ )
11      $q_c \leftarrow q_j$ 
12   else create a new state  $q_{c+1}$  and a transition
13      $\delta(q_c, s) = q_{c+1}$ , i.e.,  $\delta \leftarrow \delta \cup \{(q_c, s), q_{c+1}\}$ 
14      $Q \leftarrow Q \cup \{q_{c+1}\}$ ;
15      $q_c \leftarrow q_{c+1}$ 
16   if  $s$  is the last symbol in String then
17     Assign the state with the largest subscript the
18     accept state  $q_r$ ,  $F = \{q_r\}$ ;
19   TransitTo( $q_c, q_r$ );
20 generate a regular expression based on the NFA  $N$ ;

```

Function TransitTo(q_c, q_s)

```

1 while  $q_c \neq q_s$  do
2   if  $\exists \delta(q_c, \varepsilon) = q_k$  and  $k > c$  then
3      $q_c \leftarrow q_k$ 
4   else create a transition  $\delta(q_c, \varepsilon) = q_{c+1}$ , i.e.,
5      $\delta \leftarrow \delta \cup \{(q_c, \varepsilon), q_{c+1}\}$ ;
6      $q_c \leftarrow q_{c+1}$ 

```

Figure 7: Grammar generation

4.4 The Grammar-Generation Function

This function takes a string of symbols to generate a grammar. A solution for regular grammar inference may be used to solve our problem. However, this is not feasible. The theoretical notion of inference “in the limit” with positive examples alone is undecidable [9]. Hence, known applications of regular grammar inference use heuristics specific to both problem formulations and solutions. We found most such heuristics are inapplicable to our problem. An example of such an application is XTRACT [8] which uses MDL [24], an information theory technique, to infer schemas from a collection of XML documents. However, the technique is not suitable for our domain because it uses heuristics “inspired by DTDs” that enumerates a small set of potential regular expressions and then uses the MDL principle to pick the “best”. Also, its MDL based approach relies on availability of more than one example strings to generate a regular expression which does not work for a single list page where a regular expression has to be identified from just one string of symbols.

We now define the grammar that we are looking for.

Regular expressions for nested relations: Since a list

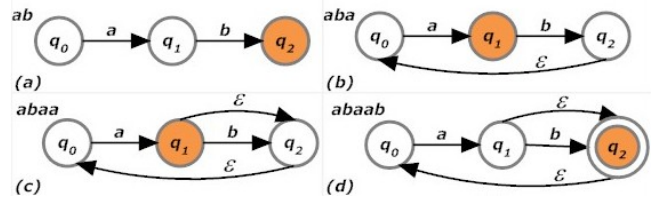


Figure 8. Various stages in the formation of NFA from the string *abaab*. Symbols on left is the processed string. The shaded state is the current state.

is a sequence of data records that follow the same template D , thus its regular expression is $(D)^+$. The regular expression of a nested relation is defined as follows:

- There is a set of *atomic symbols*, $S = \{S_1, S_2, \dots, S_n\}$;
- If $D_i \in \{D_1, D_2, \dots, D_k\}$ is an atomic symbol or a list regular expression, then $D_1 D_2 \dots D_k$ is a *record regular expression*;
- If D is a *record regular expression*, $(D)^+$ is a *list regular expression*.

Based on this form of grammar, we present the grammar generation algorithm in Figure 7. This algorithm makes the following assumption:

Assumption: All symbols made up the first record in a list are present.

Note that here each symbol represents a sub-tree, not a data item to be extracted, which is a leaf. Thus, the assumption says that all sub-trees making up the first record should be present. It does not say that there is no missing item in the record. In fact, in each sub-tree, there can be missing data items. Hence, this assumption is rather weak. Of course, if it is not satisfied by some pages, it causes extraction errors. Our strong extraction results show that the assumption is satisfied by general Web pages.

We now explain the algorithm. Line 1 initializes a NFA (non-deterministic finite automaton). Lines 2-16 traverses *String* from left to right to construct the NFA. Line 17 produces a regular expression from the NFA. We use an example to illustrate (Figure 8). Consider the following string, “a b a a b”, which should produce the regular expression of $(ab)^+$ (? for optional). We start with start state q_0 as the current state. For an input symbol s (which is a), we check if there exists a transition from some state to another using that symbol a (lines 4-6). If not, we create a new state and make a transition from the current state to the new state using symbol s and make the new state the current state (lines 11-13). In our string, the first two symbols a and b will form 2 new states (Figure 8(a)). Next we see another symbol a and the current state q_2 . Now, there exists a transition from state q_0 to q_1 using symbol a (line 6). We make an ε transition from state q_2 to q_0 and then another transition from q_0 using symbol a (lines 7-10) to q_1 (Figure 8(b)). The next symbol is also a and the current state is q_1 . Since there exist a transition from q_0 to q_1 using a and an empty transition from q_2 ($> q_1$) to q_0 , (line 7), so we make an ε transition from q_1 to q_2 and make

Function: UpdateW(W, g_A, g_B)

```

1  nodesA = 1, nodesB = 1; //Initialize to 1, for root nodes
2  for each +-pattern in grammar gA or in grammar gB do
3    Let SA and SB be the list of child sub-trees in A and B
      respectively, which are assigned the same symbol
      in the instances of that pattern
4    Let, SA = {SAi}, i = 1...x;
5       SB = {SBj}, j = 1...y;
6    repeat steps 7-19 for all such pair of lists
7      avgScore = 0, avgNodes = 0, count = 0;
8      for each subtree SAi in SA and SBj in SB do
9        avgScore += W[i,j].score
10       avgNodes += W[i,j].nodes1 + W[i,j].nodes2
11       count++
12     avgScore = avgScore / count
13     avgNodes = avgNodes / (2*count)
14     W[i,j].score = avgScore,      i = j = 1
15     W[i,*].score = 0, for i > 1
16     W[* ,j].score = 0, for j > 1
17     nodesA += avgNodes;
18     nodesB += avgNodes
19 for each child-tree of A not inside a pattern do
20   nodesA += number of nodes in child-tree;
21 for each child-tree of B not inside a pattern do
22   nodesB += number of nodes in child-tree;
23 return (W, nodesA, nodesB)

```

Figure 9: Update the score matrix W

q_1 the current state (Figure 8(c)). The following symbol is b and current state is q_1 . There exists a transition from q_1 to q_2 using b (Figure 8(d)). State q_2 is the accepting state (line 15). A regular expression can be easily generated. Note that the regular expressions produced by this algorithm do not have disjunctions (i.e. $a|b$) except $(a|\epsilon)$, which means a is optional (denoted by $?$). Such regular expressions are called *union-free regular expressions* [6, 16]. We also note that due to the fixed grammar form and the assumption, the algorithm in Figure 7 is a deterministic algorithm. Below we give a bigger example and the regular expression generated by the algorithm.

“a b c b c a b a b c a b” $\rightarrow (a(bc^?)^+)$

Incidentally, one of the main problems in data extraction is to identify data record boundaries in a list. The proposed algorithm identifies them automatically, which is shown in the final grammar by $+$. We will also compare the record boundary identification result of our algorithm with that of a state-of-the-art system in Section 5.2.3.

4.5 The UpdateW Function

The grammars generated help identify repeats (which are data records forming lists) and optional items in the two trees. Next the score matrix W and the number of nodes in the two trees have to be adjusted to reflect the presence of lists. In this work, we handle repeats by keeping only one of its instances (which are data records). Then the corresponding rows and columns in the score matrix W have to be revised. As a result, the number of nodes will also be updated. For every list, we calculate the average

score of matching one instance in the first tree with all instances in the second tree. The entries in the score matrix W corresponding to the first instance in both trees are updated with the score. The rest of the rows and columns corresponding to the other instances are set to 0.

In the algorithm (Figure 9), line 1 initializes the number of nodes in the two trees. Lines 2-18 compute the average score (line 12) and the average number of nodes (line 13) of a record instance in a list. Lines 14-16 set the score of the first pair of instances to the average score and the other entries to zero. Lines 17-18 add the average number of nodes in one instance to the total number of nodes in the trees. Lines 19-22 add the number of nodes in the remaining child sub-trees to the total nodes under the trees.

4.6 Multiple Tree Alignment

G-STM only matches two trees. If we need to build a wrapper using multiple trees (pages), multiple alignments is needed. We use the partial tree alignment (PTA) method in [32]. PTA produces a single template tree from multiple trees. G-STM is still used in tree matching. PTA is not included in our algorithm. PTA is also used to align multiple records in a list. The aligned records are replaced with the single list template tree for further alignments. For example, there is a list with some records containing nested lists. Each lower level list is aligned first to produce a template tree before being used in higher level alignments.

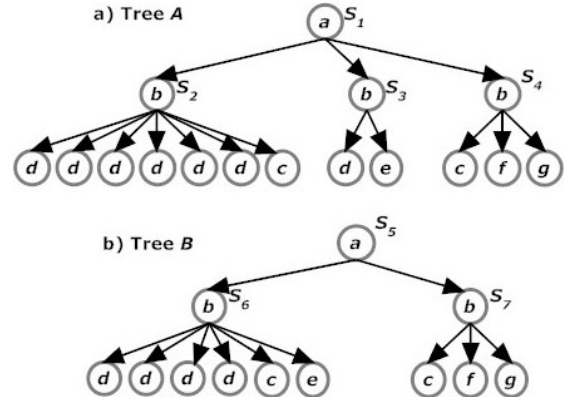


Figure 10. Example trees for G-STM

Table 1. The output of G-STM on child sub-trees S_1 and S_5 . (sc, n_1, n_2) is returned by G-STM after matching the two trees. sc is the score, n_1 and n_2 are the number of nodes in first and second sub-trees respectively. sc_N is the normalized score $score_{norm}$.

S_1-S_5	S_6	S_7
	(sc, n_1, n_2)	(sc, n_1, n_2)
	sc_N	sc_N
S_2	(3, 3, 4)	(2, 8, 4)
	75%	25%
S_3	(3, 3, 4)	(1, 3, 4)
	75%	25%
S_4	(2, 4, 7)	(4, 4, 4)
	28%	100%

4.7 A Complete Example of G-STM

Consider two trees in Figure 10. The template of the two trees has a list inside a list. Sub-trees S_2 and S_3 of tree A are repeating with sub-tree S_6 of tree B . Also, the node d is repeating in these sub-trees (in a more complex scenario node d can be a sub-tree too). Table 1 shows the output of G-STM on different child sub-trees of nodes S_7 and S_5 . We explain the entry corresponding to S_2 and S_6 in the table as an example. When we run $G\text{-STM}(S_2, S_6)$, we get the grammars d^+c and d^+ce for the two sub-trees. The repeating pattern is d^+ . Sub-tree S_2 has 6 instances of d and subtree S_6 has 4 instances of d . The average score $avgScore$ (respectively, the average number of nodes $avgNodes$) of comparing an instance of d^+ in S_2 to an instance of d^+ in S_6 is 1 (1). So, the score (number of nodes) contributed by d^+ to either sub-tree is 1. Also, the two sub-trees S_2 and S_6 have nodes b and c in common. So, the total score is 3. The number of nodes in S_2 is 3 (nodes b and c plus 1 instance of d^+). The number of nodes in S_6 is 4 (nodes b , c , and e plus 1 instance of d^+). Thus, the output of $G\text{-STM}(S_2, S_6)$ is (3, 3, 4). The normalized score $score_{norm}$ is 75%. Now, we see that the $score_{norm}$'s of $G\text{-STM}(S_2, S_6)$ and $G\text{-STM}(S_3, S_6)$ are the same even though they have different scores under the original STM algorithm. So, based on Table 1, sub-trees S_2 and S_3 will be matched with S_6 , and sub-tree S_4 will be matched with S_7 . Now, we compute the output of $G\text{-STM}(S_7, S_5)$. The average score $avgScore$ of $G\text{-STM}(S_2, S_6)$ and $G\text{-STM}(S_3, S_6)$ is $(3+3)/2 = 3$. The score of $G\text{-STM}(S_4, S_7)$ is 4. So, the total score is 8 ($3+4+1$). The number of nodes outputted by $G\text{-STM}(S_2, S_6)$ and $G\text{-STM}(S_3, S_6)$ are (3, 4) and (3, 4). The final average of the number of nodes in these three matching sub-trees is $(3+4+3+4)/4 = 3.5$. The number of nodes in sub-trees S_4 and S_7 is 4. So, the total number of nodes under S_7 (the same as in S_5) is $3.5+4+1 = 8.5$. Thus, the output of $G\text{-STM}(S_7, S_5)$ is (8, 8.5, 8.5).

4.8 Complexity Analysis of G-STM

The dynamic programming algorithm STM has the complexity of $O(n_1n_2)$ as it has no level-crossing or node replacements, where n_1 and n_2 are the numbers of nodes in trees A and B respectively [31]. G-STM inherits the dynamic programming of STM. Grammar generation is linear in the length of the string. Preparing the strings for grammar generation is $O(n_1n_2)$. UpdateW is again linear. Thus, G-STM has the same complexity as STM.

4.9 Extraction Based on Multiple Pages and a Single List Page

We now discuss how to use G-STM for data extraction. The first step is to construct a DOM tree (or tag tree) from each input Web page. This process is fairly straightforward and standard due to the nested structure of HTML tags. See [32, 17] for detailed discussions on tree building. The tree is then used by G-STM.

As mentioned earlier, G-STM can be used to generate data extraction patterns based on both problem

formulations discussed in Section 3. Clearly, Problem 1 (based on multiple pages) can be solved by directly applying the G-STM algorithm. Each page is simply a tree. Problem 2 (based on a single list page) can be solved using the same algorithm too. We only need to make a copy of the input page and give both pages to G-STM (the original and the copy). This one algorithm solving both problems is a major advantage of the proposed approach.

5. Experimental Evaluation

This section evaluates the proposed technique. It consists of two parts for the two problem formulations.

- 1. Pattern generation based on multi-pages.** Here, we compare G-STM with RoadRunner [6] and EXALG [1]. Since EXALG is not publicly available, we can only compare it based on its results of 9 samples at its site.
- 2. Pattern generation based on a single list page.** We compare G-STM with the DEPTA system [32]. We will not compare with NET as it performed similarly to DEPTA initially. However, the current version of DEPTA has been improved twice after its original publication [32]. Although there are several other data extraction systems, MDR [18], VIPER [26], ViNTS [33] and MSE [34], they only segment a list of data records but do not extract individual data items. Since G-STM also can segment data records, we will compare it with the state-of-the-art MSE system.

5.1 Experimental Web Pages

Two public domain data sets and one data set of our own are used to thoroughly test the proposed algorithm.

Two public domain benchmark data sets:

1. TBDW Ver. 1.02 [30], which has pages from 51 sites, 5 pages per site.
2. ViNTs data set 2 [33], which has pages from 101 sites, 11 pages per site.

They are search result pages of the deep Web. Deep Web refers to databases hidden behind Web query interfaces. When the user issues a query (e.g., find houses for sell in a particular area), the system retrieves the relevant data records from the underlying database and displays them in Web pages (e.g., a list of houses). These sets have been used in VIPER, MSE and ViNTS, (although these systems only segment data records, and do not extract individual data items). Note that the host servers of these data sets are down sometimes; interested readers can send authors emails to obtain the data sets. DEPTA does not have associated test sets. It used live pages for testing when the paper was written. Most of the pages no longer exist.

All the pages in these two data sets are list pages, and they are used to test G-STM for both problem formulations. All pages were used for multi-page extraction (Problem 1). For single page extraction (Problem 2), 1 page per template/site is used, i.e., 51 pages from TBDW and 101 pages from ViNTs. These pages were selected by the

Table 2. Data Sets Characteristics

Data Set	TBDW	ViNTs-2	Our Data Set
# Sites (templates)	51	101	32
# Pages per site (template)	5	11	5
Avg. # records per page	20	15	10 (for 22 sites)
Avg. # items per record	3.5	3.5	6 (for 22 sites)
Avg. # non-list items per page	6	6	34

authors of VIPER to test their record segmentation algorithm. Note that DEPTA only takes a single page and extracts data from its data records.

Our own data set: To thoroughly test the system, we also collected data from 32 Web sites which are not search engine results. The sites were randomly selected using Yahoo directory. Our set has sites such as news, books, movies, weather, sports, company sites, government sites, brochures, forums, etc. From each site, we randomly picked 5 pages which follow the same template. We did not use more pages from each site because the other pages are very similar. Out of these 32 Web sites, pages from 10 Web sites do not have lists, and the rest (22 sites) all contain lists. For multiple page extraction (Problem 1), pages from all 32 sites were employed. For single page extraction (Problem 2), we only used all pages from the 22 sites as they are list pages.

Table 2 summarizes the statistics of the three data sets. We can see that our data set has a larger average number of items per record and also has more non-list items per page. Search engine result records from TBDW and ViNTs-2 are relatively simple.

Additional data set. EXALG [1] authors provided the results of their model on pages from 9 Web sites:

amazon cars, amazon pop music, MLB, RPM packages, UEFA teams, UEFA players, eBay, Netflix and US Open.

An average of 25 pages per site was provided. This data set is also used to compare our system with EXALG. EXALG itself is not publicly available for testing.

5.2 Experimental Results

We now present the experimental results. The results of Problem 1 (based on multiple pages) are given first, and the results for Problem 2 (based a single list page) second.

Table 3. Experiments for Multiple Page Extraction

Data Set	TBDW		ViNTs-2		Our Data Set	
	G-STM	Road Runner	G-STM	Road Runner	G-STM	Road Runner
Precision	97.6%	74.8%	98.1%	85.4%	91.6%	88.2%
Recall	96.9%	77.6%	96.3%	82.3%	96.9%	60.7%
F-score	0.97	0.76	0.98	0.84	0.94	0.72

5.2.1 Extraction Based on Multiple Pages

Table 3 shows the experimental results of G-STM and Roadrunner on the three data sets. RoadRunner was tested on only 17 sites from TBDW (out of 51) and 54 sites from ViNTs-2 (out of 101). For the rest of the sites, Roadrunner did not work because all the pages of each site contain exactly the same number of records. This happens if the pages are first n search result pages for a query with m records per page. RoadRunner only detects list when the number of records in two pages are different [6].

For G-STM, 2 randomly selected pages were used to generate patterns, which were used to extract data from the remaining pages. For Roadrunner, all the pages from a site were given to the system for extraction as it does not have an option for separate testing. Table 3 gives the aggregated precisions, recalls and F-scores of the two systems. Note that those pages on which Roadrunner did not work were not used in computing the results for Roadrunner, but were used to compute the results of G-STM.

Table 4 (rows 1 to 32) shows the individual result for each site for our data set. It also shows separate results for non-list and list data elements. TP is the average number of true positive items per page extracted from the pages of a site, FP (or FN) is the number of false positive (or false negative) items extracted per page of a site. Navigational links are not counted as they are not useful. Rows 3, 12, and 15 are empty for Roadrunner because it crashed on the pages of the sites and thus did not generate any result. These sites were not used in computing the precision, recall and F-score for Roadrunner.

From Tables 3 and 4, we observe that G-STM outperforms Roadrunner by a large margin in F-score, about 20% in all results. Although G-STM is better in both precisions and recalls, the major lose of Roadrunner is recall (around 30% on our data set), which show that Roadrunner is unable to match a large number of items. We believe that its heuristic matching algorithm is inferior to tree matching based on optimization. Thus, Roadrunner only works well on simple pages with clear structures. We also observe from Table 4 that G-STM outperforms RoadRunner considerably on commercial sites like movies, weather, finance and products, etc (e.g. rows 4, 5, 7, 11, 13, 14, 16, 18, and 19). Such sites contain complex lists with multiple sections. G-STM performs equally well on both list and non-list data records. We noticed that for rows 4 and 19, there are a large number of data items that Roadrunner could not extract. If we do not consider these two sites, the precision, recall and F-score of Roadrunner are 91.5%, 70% and 0.79, which are still much poorer than those of G-STM, 93.9%, 95.4% and 0.94 without considering the two sites.

Errors in G-STM: There are three main sources of errors in the results of G-STM: (1) Non-lists are taken as lists. This is a hard problem because in some cases, understanding of the words is needed to make decisions. (2) Sometimes nodes are not aligned properly. This is mainly caused by some irregular tags. (3) Some list nodes

Table4. Experiments for Multi-Page Extraction (Our Data)

URL	G-STM						ROADRUNNER					
	All Items		Non-List Items		List Items		All Items		Non-List Items		List Items	
	TP	FP/FN	TP	FP/FN	TP	FP/FN	TP	FP/FN	TP	FP/FN	TP	FP/FN
amazon.com	83	1/2	7	1/2	76	0/0	78	1/7	2	0/7	76	1/0
att.com	43	1/0	6	1/0	37	0/0	43	1/0	6	1/0	37	0/0
cnet.com	86	1/6	4	1/1	82	0/5	-	-/-	-	-/-	-	-/-
fortune.com	170	4/5	28	4/5	142	0/0	7	0/168	7	0/26	0	0/142
weather.com	48	0/0	34	0/0	14	0/0	2	7/46	2	7/32	0	0/14
yahoo.com	40	0/2	2	0/2	38	0/0	22	0/20	4	0/0	18	0/20
imdb.com	85	1/0	45	1/0	40	0/0	45	40/45	45	40/5	0	0/40
newegg.com	94	0/10	38	0/6	56	0/4	94	3/10	34	0/10	60	3/0
citigroup.com	40	0/0	40	0/0	0	0/0	38	0/2	38	0/2	0	0/0
usa.gov	56	8/0	56	8/0	0	0/0	56	4/0	56	0/0	0	4/0
worldportsource.com	50	4/4	33	4/0	17	0/4	6	0/48	6	0/27	0	0/21
bankofamerica.com	14	2/3	14	2/3	0	0/0	-	-/-	-	-/-	-	-/-
addons.mozilla.org	34	0/0	11	0/0	23	0/0	6	0/28	6	0/5	0	0/23
movies.go.com	60	14/0	60	14/0	0	0/0	0	0/60	0	0/60	0	0/0
overstock.com	136	16/18	31	3/5	105	13/13	-	-/-	-	-/-	-	-/-
rxlist.com	40	0/0	28	0/0	12	0/0	29	9/11	19	0/9	10	9/2
epicurious.com	23	1/0	23	1/0	0	0/0	23	0/0	23	0/0	0	0/0
leisure.travelocity...	54	1/4	6	1/0	48	0/4	6	0/52	6	0/0	0	0/52
gamespot.com	110	70/4	22	1/0	88	69/4	7	4/107	7	0/15	0	4/92
mathworld.wolfram...	17	0/0	17	0/0	0	0/0	5	12/12	5	0/12	0	12/0
rediff.com	19	5/0	19	5/0	0	0/0	19	0/0	19	0/0	0	0/0
mtv.com	33	2/6	15	1/5	18	1/1	8	31/31	8	0/12	0	31/19
un.org	24	9/2	24	9/2	0	0/0	26	0/0	26	0/0	0	0/0
Yellowpages.com	80	0/2	24	0/0	56	0/2	66	0/16	8	0/16	58	0/0
Globalsecurity.org	44	8/3	44	8/3	0	0/0	30	16/17	30	0/17	0	16/0
people.com	75	7/0	45	7/0	30	0/0	48	24/27	42	24/3	6	0/24
microsoft.com	58	4/9	58	4/9	0	0/0	67	0/0	67	0/0	0	0/0
nytimes.com	38	7/9	24	1/2	14	6/7	7	0/40	7	0/19	0	0/21
egov.cityofchicago...	133	23/1	6	2/1	127	0/0	134	0/0	7	0/0	127	0/0
pricegrabber.com	110	0/5	14	0/0	96	0/5	113	2/2	12	0/2	101	2/0
worldaffairsboard...	100	0/0	5	0/0	95	0/0	100	0/0	5	0/0	95	0/0
howardforums.com	75	0/0	15	0/0	60	0/0	75	0/0	15	0/0	60	0/0
Total	2072	189/95	798	79/46	1274	89/49	1160	154/749	512	72/279	648	82/470
Precision	91.6%		90.9%		93.4%		88.2%		87.6%		88.7%	
Recall	96.9%		94.4%		96.3%		60.7%		64.7%		57.9%	
F-score	0.941		0.926		0.948		0.719		0.744		0.70	

were not identified. The reason for this problem is that some list nodes are too different from the other list nodes. This problem may be solvable based on visual (or rendering) information [26, 32, 34]. In this work, our focus has been on enhancing the tree matching with list handling to solve our problem. We have not used the visual information to engineer a better system. Our future work will exploit that. However, our current results are already considerably better than existing systems.

G-STM vs. EXALG: We now compare G-STM with EXALG and RoadRunner based on pages from 9 sites provided by EXALG (see Section 5.1). For each site, G-STM used 2 random pages for wrapper generation and the rest of the pages for extraction. EXALG and RoadRunner

take all pages as input and extract the data. Table 5 shows the overall results of the three algorithms. Since EXALG is not available for evaluation, we use the results given at the site. G-STM had 100% recall and 100% precision, whereas EXALG had 90% recall and 100% precision. EXALG does slightly better than RoadRunner. Most pages from these 9 websites had fairly simple structures. Only three websites of *ebay*, *netflix* and *US open* have more complex structures and both EXALG and RoadRunner faltered on these sites, but G-STM can extract them perfectly.

5.2.2 Extraction Based on a Single List Page

Table 6 shows the overall results of G-STM and DEPTA on the three data sets. DEPTA runs on only 20

pages from TBDW (out of 51) and 32 pages from ViNTs-2 (out of 101) and 12 pages from our data set (out of 22, which contain lists) due to crash or no results produced. For DEPTA’s results in Table 6, we only used those pages that DEPTA ran successfully. For the results of G-STM, all pages were used in computing its results.

We observe from the table that G-STM outperforms DEPTA on all three data sets. The difference is the highest for our data set as it contains more complex data records. G-STM has near perfect precision and recall, except in a few cases where the pages have little internal structure, just text elements using simple style tags. Notice that for this single page based extraction, we are only concerned with lists in a page, not the other data items, because only lists allow patterns to be discovered. Table 7 shows the detail results on our data set. For 16 out of 22 sites, G-STM gave perfect precision and recall. The reasons for loss in precision and recall for the remaining sites were the same as for multi-page extraction. For example, the loss of precision was mainly due to some primate non-list data elements in the records being considered as lists (e.g. rows 2, 4, 7 and 19 in Table 7).

5.2.3 Segmenting List of Data Records

Since some existing systems can perform data record segmentation (identifying their boundaries), we compare SMT-L on this task with the state-of-the-art system MSE. Since MSE only finds the important list of data records in a page. In comparison, we only consider those lists that MSE is able to identify. MSE (provided by the authors) takes up to 5 training pages to learn a wrapper. So, for MSE randomly selected 5 pages were used for training and the remaining as test pages (for TBDW and our data set the maximum number of pages is 5, so all the pages were given for unsupervised training and testing). For G-STM, only one page was given at a time as an input in segmentation. Recall is the number of records extracted fully and precision is measured based on the records that do not belong to the list identified by the systems. Both G-STM and MSE perform well on TBDW and ViNTs-2 datasets. G-STM has precision and recall of 100% and 96.5%, and MSE has precision and recall of 100% and 98%. These pages are simple as they are regular lists of search results in the centers of the pages. However, on our dataset (pages with lists from 22 websites), MSE does poorly in finding and segmenting lists. For 9 out of 22 websites, MSE could not find any list of records in the page (it returned only the list of navigational links at the top, left or bottom of the page). So we could not compare the performance of G-STM with MSE on these 9 sites. On the remaining 13 sites, MSE has precision and recall of 97.3% and 65.7% on the lists which it identified, while G-STM has precision and recall of 100% and 95% on the same lists of records.

In summary, our extensive experiments enable us to conclude that G-STM performs dramatically better than the current state-of-the-art research systems.

Table 5. Experimental Results on the EXALG Data Set

System	Precision	Recall	F-score
G-STM	100%	100%	1.00
RoadRunner	91%	92%	0.91
EXALG	100%	90%	0.94

Table 6. Experiments for Single Page Extraction

Data Set	TBDW		ViNTs-2		Our Data Set	
	G-STM	DEPTA	G-STM	DEPTA	G-STM	DEPTA
Precision	99.8%	99.5%	98.5%	95.1%	98.4%	88.8%
Recall	96.6%	85.3%	96.7%	83.9%	99%	86.3%
F-score	0.98	0.92	0.98	0.89	0.99	0.88

Table 7. Page by Page Single Page List Extraction Results on Our Data

	URL	G-STM		DEPTA	
		TP	FP/FN	TP	FP/FN
1	amazon.com	76	0/0	76	0/0
2	att.com	37	5/0	-	-/-
3	cnet.com	87	0/0	75	3/12
4	fortune.com	142	10/0	52	6/90
5	weather.com	14	0/0	-	-/-
6	yahoo.com	38	0/0	-	-/-
7	imdb.com	40	4/0	40	13/0
8	newegg.com	60	0/0	60	0/0
9	worldportsource.com	21	0/0	21	6/0
10	addons.mozilla.org	23	0/0	-	-/-
11	overstock.com	118	0/0	112	0/4
12	rxlist.com	12	0/0	12	0/0
13	leisure.travelocity...	49	0/3	-	-/-
14	gamespot.com	83	0/9	92	23/0
15	mtv.com	19	0/0	-	-/-
16	yellowpages.com	58	0/0	-	-/-
17	people.com	30	0/0	-	-/-
18	nytimes.com	21	0/0	21	0/0
19	egov.cityofchicago...	127	6/0	123	28/4
20	pricegrabber.com	101	0/0	76	10/25
21	worldaffairsboard	95	0/0	95	18/0
22	howardforums...	60	0/0	-	-/-
	Total	1311	20/12	855	107/135
	Precision	98.4%		88.8%	
	Recall	99%		86.3%	
	F-score	0.99		0.88	

6. Conclusions

This paper studied automated data extraction. The contribution of this paper is two-fold. First, it integrated list handling into a tree matching algorithm to produce a

generalized tree matching algorithm. Detecting lists is based on a novel grammar generation method. To our knowledge, no current tree matching algorithm is able to consider lists. Yet, handling lists is essential for Web data extraction. Second, it is shown through extensive experiments based on existing benchmark data sets and our own new data set that the proposed G-STM algorithm outperforms the state-of-the-art existing methods dramatically. What is also important is that the single G-STM algorithm can solve both problems of Web data extraction. These two problems are currently solved using different algorithms. The system has been tested in a commercial setting and is in the processing being licensed to a commercial company.

7. References

- [1] Arasu, A. and Garcia-Molina, H. Extracting structured data from web pages. *SIGMOD'03*.
- [2] Chakrabarti, D., Kumar, R., Punera, K. Page-level Template Detection via Isotonic Smoothing. *WWW'07*.
- [3] Chang, C. and Lui, S. IEPAD: Information extraction based on pattern discovery. *WWW'2001*.
- [4] Cohen, W. W., Hurst, M., and Jensen, L. A flexible learning system for wrapping tables and lists in html documents. *WWW'2002*.
- [5] Cohen, W. W. and Fan, W. Learning Page-Independent Heuristics for Extracting Data from Web Pages. *Computer Networks 1999*.
- [6] Crescenzi, V., Mecca, G. Automatic information extraction from large websites. *J. ACM*, 51(5), 2004.
- [7] Crescenzi, V., Mecca, G., Merialdo, P. Roadrunner: Towards automatic data extraction from large web sites. *VLDB '01*.
- [8] Garofalokis, M., Gionis, A., Rastogi R., Seshadr, S., and Shim, K. XTRACT: A system for extracting document type descriptors from XML documents. *SIGMOD, 2000*.
- [9] Gold, E.M. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
- [10] Hsu, C.-N. and Dung, M.-T. Generating finite-state transducers for semi-structured data extraction from the web. *Inf. Syst.*, 23(9):521–538.
- [11] Irmak, U., Suel, T. Interactive wrapper generation with minimal user effort. In *WWW'06*.
- [12] Kosala, R., Blockeel, H., Bruynooghe, M. and Bussche, J. V. Information extraction from structured documents using k-testable tree automaton interface. *Data Knowl. Eng.* 58(2): 129-158, 2006.
- [13] Kushmerick, N. Wrapper induction: efficiency and expressiveness. *Artificial Intelligence*, 118, 2000.
- [14] Kushmerick, N., Weld, D. and Doorenbos, R. Wrapper induction for information extraction. *IJCAI-1997*.
- [15] Lerman, K., Getoor, L., Minton, S. and Knoblock, C. Using the structure of web sites for automatic segmentation of tables. In *SIGMOD-2004*.
- [16] Liu, B., Zhai, Y. NET - A System for Extracting Web Data from Flat and Nested Data Records. In *WISE-2005*.
- [17] Liu, B. Web Data Mining: Exploring Hyperlinks, Contents and Usage Data, Springer, 2007.
- [18] Liu, B., Grossman, R., Zhai, Y. Mining Data Records in Web Pages. In *KDD-2003*.
- [19] Muslea, I., Minton, S. and Knoblock, C. A hierarchical approach to wrapper induction. In *AGENTS'99*.
- [20] Nie, Z. Wu, F., Wen, J-R, and Ma, W-Y. Extracting Objects from the Web. In *ICDE 2006*.
- [21] Oncina, P., and Garca, J. *Inferring regular languages in polynomial update time*. Pattern Recognition and Image Analysis, pages 49–61.
- [22] Pinto, D., McCallum, A., Wei, X. and Croft, B. Table extraction using conditional random fields. *SIGIR '03*.
- [23] Reis, D.-C., Golgher, P.-B., Silva, A.-S. and Laender, A.-F. Automatic web news extraction using tree edit distance. In *WWW'04*.
- [24] Rissanen, J. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.
- [25] Sakamoto, H., Murakami, Y., Arimura, H. and Arikawa, S. Extracting Partial Structures from HTML Documents. *FLAIRS 2001*.
- [26] Simon, K. and Lausen, G. ViPER: Augmenting Automatic Information Extraction with Visual Perceptions. *CIKM '05*.
- [27] Tai, K.-C. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.
- [28] Vieira, K., Silva, A., Pinto, N., Moura, E., Cavalcanti, J. and Freire, J. A fast and robust method for web page template detection and removal. *CIKM'06*.
- [29] Wang, J., and Lochovsky, F. H. Data extraction and label assignment for web databases. In *WWW '03*.
- [30] Yamada, Y., Craswell, N., Nakatoh, T. Hirokawa, S. Testbed for information extraction from deep web. *WWW'04*.
- [31] Yang, W. Identifying syntactic differences between two programs. *Softw. Pract. Exper.*, 21(7), 1991.
- [32] Zhai, Y., and Liu, B. Web data extraction based on partial tree alignment. *WWW '05*.
- [33] Zhao, H., Meng, W., Wu, Z., Raghavan, C. and Yu, C. Fully automatic wrapper generation for search engines. *WWW'05*.
- [34] Zhao, H., Meng, W, Yu, C. Automatic extraction of dynamic record sections from search engine result pages. *VLDB'06*.
- [35] Zhu, J., Nie, Z., Wen, J-R., Zhang, B. and Ma. W.-Y. 2D Conditional Random Fields for Web Information Extraction. *ICML-05*.