FastWrap: An Efficient Wrapper for Tabular Data Extraction from the Web*

Mohammad Shafkat Amin Hasan Jamil

Department of Computer Science, Wayne State University, USA

shafkat@wayne.edu, jamil@cs.wayne.edu

Abstract

In the last few years, several works in the literature have addressed the problem of data extraction from web pages. The importance of this problem derives from the fact that, once extracted, data can be handled in a way similar to instances of a traditional database, which in turn can facilitate application of web data integration and various other domain specific problems. In this paper, we propose a novel table extraction technique that works on web pages generated dynamically from a back-end database. The proposed system can automatically discover table structure by relevant pattern mining from web pages in an efficient way, and can generate regular expression for the extraction process. This approach requires no human intervention and experimental results have shown its accuracy to be promising. Moreover, the algorithm works in linear time to generate the wrapper.

1 Introduction

With the advent of numerous public domain databases and proliferation of scientific data, the necessity for a domain independent, query intrinsic knowledge extraction system has become apparent. Search engines like Google, Yahoo, Alta vista, although perform remarkably well when querying shallow web data sources, acquisition, integration and customer tailored query management of data from hidden web is yet to be perfected. One important essential tool systems require for autonomous information extraction is a wrapper. Wrappers can be generated both manually and automatically. The importance of automatic wrapper generation for web pages has been well recognized and addressed in recent research in the area of data integration. A large amount of information on the web is presented in regularly structured objects. A list of such objects in a web page often describes a list of similar items, e.g., a list of products

or services. They can be regarded as database records displayed in web pages using regular patterns. To facilitate the development of these information integration systems, an efficient and accurate tool, a wrapper, is needed.

In Figure 1, we see an example of an extraction scenario. The blue labeled rectangle falls under the region of interest and the red labeled rectangle, although contain structurally very similar data items, falls under non-important data category for the wrapper. From the example, a sample extracted data record may have the fields Title, Description, URL, link to similar pages etc. Thus it is only intuitive to address this extraction process by means of obtaining candidate patterns from the input page. We employ suffix tree based technique to obtain records which we term tabular data. In this paper, we present a novel automatic wrapper mediator system, which has been developed as a part of our Information Integration System, to streamline and expedite subsequent customizable and seamless query processing from heterogeneous data sources. The proposed tool does not require any prior knowledge of the target page and its content; neither does it require any domain specific assumption. Moreover, the wrapper generation process asymptotically takes linear time to progress.

Figure 1. Example extraction scenario

The paper is organized as follows. Section 2 describes previous approaches and state of the art methods of wrapper generation technologies. In section 3 we introduce the proposed algorithm. We then discuss top- k extension of the

*Research supported in part by National Science Foundation grants CNS 0521454 and IIS 0612203.

proposed method and its applicability to wrapper generation in section 4. Finally, we present the experimental results in section 5 and describe the future scope of research in section 6.

2 Related Work

Previous research on wrapper generation can be classified into three categories: 1. Wrapper programming languages, 2. Wrapper induction, and 3. Automatic extraction. The first approach provides some specialized pattern specification languages to help the user construct extraction programs. Systems that use this kind of approach include WICCUP [18], DEByE [16], etc. The second approach is wrapper induction, which uses supervised learning to learn data extraction rules from a set of manually labeled examples. Manual labeling of data is labor intensive and time consuming. Furthermore, for different sites or even pages in the same site, manual labeling process needs to be repeated because they may assume different templates. Example wrapper induction systems include SoftMealy [13], Stalker [19], WL2 [20] etc. The third approach is automatic extraction. Embley et al. [11] propose using a set of heuristics and domain ontologies to automatically identify data record boundaries. In [6], a method called IEPAD is proposed to find patterns from the HTML tag string of a page, and then use the pattern to extract data items. Another system called DeLA, reported in [23], generates a candidate wrapper based on a single page by finding repeated patterns in the HTML page also. It subsequently generalizes the wrapper by comparing multiple similar pages. In our work, we have concentrated on generating a wrapper that extracts tabular data pattern from an input HTML page in linear time, which can subsequently be used to extract information from other similar structured pages. The approach we have presented in this paper attempts to not only improve the wrapper generation time, but also to guarantee a fully-automated technique that results in a generalized wrapper to satisfy extraction scenario for pages with similar structure. Moreover, the proposed method does not require multiple input HTML pages to extract relevant records.

3 Proposed Technique

The proposed algorithm has four main steps. The core technique related to pattern extraction is implemented using the suffix tree. In the proposed algorithm, we extract commonly occurring patterns of highest length and super-maximal repeats, as discussed in subsequent sections, as candidates. After the pattern has been extracted, it is refined using KMP-prefix (Knuth-Morris-Pratt) algorithm and then it is converted into a regular expression which is subsequently used to extract the record level data items. KMP-

prefix algorithm is applied to prevent extraction of patterns that may have part of its prefix appended in the end. The overall algorithm is given below and the individual components are discussed in subsequent sections.

Algorithm 1 FastWrap

```

1: str  $\leftarrow$  convert HTML to symbol list
2: st  $\leftarrow$  build suffix tree for str
3: lrp  $\leftarrow$  get the longest repeated/ super-maximal pattern
4: r[]  $\leftarrow$  get modified repeated patterns after applying KMP
5: for each  $p$  in pattern do
6:   apply circular modified alignment to  $p$ 
7: end for

```

3.1 Purging and Symbolizing the HTML Text

An HTML page contains tags as well as texts. Among all the tags that an input HTML page may contain, some of them are not of any interest for the whole wrapper generation process. A non-exhaustive list of tags that are unimportant for the extraction process is given below.

, <nobr>, <wbr>, <script>, <style>, , <meta>, <noscript>, <input> etc.

Moreover, comment tags and are also considered unnecessary for extraction purposes. So these tags and any information enclosed within these tags (except < b > and
, in which case only the tags are stripped from the input page as the text enclosed by these tags may contain relevant information) are discarded from the input HTML page. Tags like <script>, <style>, and <meta> are required only for the processing purposes, and thus are also considered unimportant for extraction¹. Subsequently, the input HTML page is converted into a list of symbols, where every tag and the ending tag (if exists) associated with it is given a unique symbol representation. Moreover, all the text items are symbolized by the same symbol throughout the whole document. Thus, as part of our preprocessing, we do not differentiate among distinct text items. We use hash tables to map between unique HTML tag items and their corresponding symbols. Let H be an input HTML page consisting of alphabets $X = \zeta \cup \Delta$, where ζ denotes all HTML tags and Δ all non tag texts present in the input page. Then in the symbol generation phase, we define a function $\eta : X \rightarrow \lambda$, where λ is the set of symbols representing HTML tags and text. So, all the HTML tags are symbolized by tag specific unique symbols and all text items are symbolized by a special symbol, where no distinction is made among structurally different text information.

¹Many other unimportant tags have been identified in the literature from empirical experiments which can be included in this set to further improve the extraction process.

3.2 Suffix Tree Generation

A suffix tree is a data structure that represents the suffixes of a given string in a way that allows for particularly fast implementation of many important string operations.

Definition 3.1 (Suffix Tree) A suffix tree T for an m -character string S is a rooted directed tree with exactly m leaves numbered 1 to m . Each internal node, other than the root, has at least two children and each edge is labeled with a non-empty substring of S . No two edges out of a node can have edge labels beginning with the same character. The key feature of a suffix tree is that for any leaf i , the concatenation of the edge labels on the path from root to leaf i exactly spells out the suffix of S that starts at position i . That is, it spells out $S[i \dots m]$ [12].

Suffix tree construction algorithms have been extensively discussed in the literature [8, 14]. We use Ukkonen's [22] algorithm for suffix tree generation that boasts a linear time complexity and improves upon Weiner's [12] algorithm in terms of space requirement. Our goal in this phase is to convert the HTML input page into a string of symbols using which we construct a suffix tree for pattern extraction.

3.3 Extracting Longest Repeated Substring from the Suffix Tree

Given a string T with $|T| = n$, where $n > 0$, the longest repeated substring problem is to identify and locate the longest substring x , occurring at two or more distinct, possibly overlapping, positions in T . For finding the longest normal repeat in a string T , Karp et al first proposed a $O(|T| \log |T|)$ time algorithm [15]. However, it is an easy application of the suffix tree to find it in $O(|T|)$ time, where $|T|$ represents the length of T , i.e., constructing and extracting longest repeated substring in suffix trees is linear time solution [21]. The internal nodes of a suffix tree represent common prefixes of the suffixes of the input string. It is thus evident that the longest substring represented by an internal node in the tree is the longest repeated substring of the input. The algorithm works as follows: after constructing the suffix tree, the longest repeated substring is indicated by the deepest fork node in the tree [4]. In this case, the depth of the node is measured by the number of characters traversed from the root. Let ω be a string over the alphabet λ , where λ represents the symbolized version of the HTML page and δ be a function $\delta(\omega) = \mu$, where μ represents a suffix tree over ω . We define a function σ such that $\sigma(\mu) = \tau$. Here τ represents the longest repeated substring acquired from the suffix tree in linear time using algorithm stated in [21].

3.4 Refining the Extracted Pattern

The longest repeated substring extracted from the suffix tree may contain overlapped data which, from the context of web wrapper generation, may pose to be spurious. For example, for the string *abcdabcdabc*, the longest repeated pattern returned by the suffix tree would be *abcdabc*, which clearly is not our pattern of interest. This is due to the fact that the pattern *abcdabc* has the suffix *abc* which is also a prefix of the pattern. This may denote that the suffix *abc* is a part of the subsequent record of the input. Thus the second *abc* beginning at position 5 in the input string is basically a part of the second "*abcd*" beginning at position 5. To refine this pattern we employ KMP prefix computation function [9]. The prefix computation is formalized as follows: given a pattern $P[1 \dots m]$, the prefix function for the pattern P is the function $\pi : \{1, 2, 3, \dots, m\} \rightarrow \{0, 1, \dots, m-1\}$ such that $\pi(q) = \max\{k : k \leq q \text{ and } P_k \sqsupseteq P_q\}$. That is $\pi(q)$ is the length of the longest prefix of P that is a proper suffix of P_q and P_q is the prefix of P of length q . Thus we calculate $\pi(q)$ for all q and extract $\pi(m)$, where m is the length of the pattern. If L is the length function, then $L(P) = m$. So our pattern of interest would be, $P_{L(P)-\pi(L(P))}$ or $P_{m-\pi(m)}$. Thus by running the algorithm for the above mentioned example, we get *abcd* as our pattern of interest. The worst case running time of KMP prefix computation function is $O(m)$. Let ω be the symbolized HTML page, then the whole wrapper generation algorithm can be formulated as $\sigma(\delta(\omega))_{L(\omega)-\pi(L(\omega))}$. As none of the constituent part of the algorithm asymptotically requires more than linear time, the algorithm overall thus runs in linear time.

3.5 Regular Expression Generation

The pattern extracted from the previous step is converted into a regular expression in this step. Whenever a text item is encountered in the pattern, it is replaced by the regular expression $[\text{<^>}]^*$, where $\wedge$ in this context defines that anything but the characters $<$ and $>$ can appear in the matching text. Thus any non-tag item is considered as text. All the tags are concatenated with the string $[\text{>}]^*$ to account for the attributes and the parameters associated with a tag. Thus the regular expression for a Table tag would be $< \text{Table}[\text{>}]^* >$ which will ensure the extraction of table tags that has attributes like "border" associated with it. For example, the regular expression generated for the expression pattern $<\text{li}><\text{i}><\text{b}>\text{TEXT}</\text{b}></\text{i}></\text{li}>$, may appear like the following

```
"<li[>]*>[\s]*<i[>]*>[\s]*[<^>]*
<b[>]*>[\s]*[<^>]*</b[>]*>[\s]*
</i[>]*>[\s]*</li[>]*>[\s]*"
```

In the generated regular expression, the generic suffix $[\text{>}]^*$ added to each of the tags satisfies any tag structure

that may have attributes associated with it. Since the existence of these attributes are not significant for the extraction purpose, we have chosen to generalize its probable appearance in the input HTML text.

3.6 Circular Alignment Modification

The regular expression that is generated by the previous stages may still suffer from misalignment, which in turn may result into overlapped record extraction from the web data sources. In order to deal with this, a circular alignment modification has to be performed. This module takes as input the regular expression generated thus far and repeatedly checks to see whether the regular expression has a valid HTML pattern. So, the exact order and the nesting of the tags has to be properly maintained. For example for the string $S = abcdaxbcdyabd$, we may get the longest repeated pattern $bcd a$. But in reality, the original row in the HTML page may be represented by $abcd$. This can happen due to missing attributes and multi-valued attributes. When we try to map back $bcd a$ it does not yield a valid HTML syntax. But we can convert this invalid pattern into a valid pattern by performing a circular alignment modification² on the extracted patterns, i.e., if needed, invalid patterns can be transformed into a valid pattern by performing a circular alignment modification. The algorithm 2 summarizes this process. It should be evident for this algorithm that the method is non-linear in the size of the pattern. However, in practical instances the pattern is so short that the cost of the circular alignment modification is negligible.

Algorithm 2 Circular Alignment Modification

```

1: for  $i = 1$  to  $PatternLength$  do
2:    $Head = 1$ 
3:    $Tail = PatternLength$ 
4:   if  $CircularValidation(Head, Tail)$  then
5:     return  $Pattern$ 
6:   else
7:      $Pattern = RotatePattern()$ 
8:   end if
9: end for

```

4 Top- k Pattern Acquisition

In a pragmatic web data driven extraction scenario, siphoning a single pattern that matches a user expectation may often prove to be counter intuitive unless a domain dependent use case is considered. So the importance of actually extracting top- k patterns for the web data sources and ranking them also needs to be addressed. To address this issue we introduce the following two concepts.

²Here, circular alignment means a circular shift of the obtained pattern to get the valid pattern.

Definition 4.1 (Maximal Repeat) A maximal repeat σ is a substring of S that occurs in a maximal pair in S . That is σ is a maximal repeat in S if there is a triple $(P_1, P_2, |\alpha|) \in R(S)$ [12].

In the definition above P_1 and P_2 are the starting positions of σ and $R(S)$ stands for the set containing all the triples describing maximal pairs in S . Clearly, maximal pairs are identical substrings α and β in S that cannot be extended on either side without violating the equality of the two substrings. Extracting maximal repeats as a candidate pattern in the web data extraction applications has a caveat though. For example, in $S = aabxayaab$, substring a is a maximal repeat but so is aab . Thus the definition of super-maximal repeat below comes in handy.

Definition 4.2 (Super-Maximal Repeat) A super-maximal repeat is a maximal repeat that never occurs as a substring of any other maximal repeat.

Thus, in order to extract more than one candidate pattern reflecting user expectation, the choice of either maximal repeat or super-maximal repeat seems logical. This myriad of patterns, once extracted can be ranked according to an objective function that reflects syntactic and semantic comprehension of the domain in question. For the simplest use case, a function of pattern size and number of repeats can be used as parameters for the objective function. The extraction of all maximal repeats and super maximal repeats can be executed in linear time from a constructed suffix tree [12]. If σ is a maximal repeat then there must be at least two copies of σ in S where the character to the right of the first copy differs from the character to the right of the second copy, so σ will be a path-label of a node v in the suffix tree. A node v of the tree is called left diverse if at least 2 leaves in v 's sub-tree have different left characters and string σ labeling the path to an internal node v of T is a maximal repeat if and only if v is left diverse. For super-maximal repeats, a left diverse node v represents a super-maximal repeat if, and only if, all of v 's children are leaves, and each has a distinct left character. We can find the left diverse nodes by the following procedure:

- Leaf node:
 - Label leaves with their left character.
- Internal node v :
 - If any child is left diverse, so is v .
 - If two children have different left character labels, v is left diverse.
 - Otherwise, take on left character value of children.

The patterns thus obtained must undergo the same purging and filtering process before they are ranked to be used for practical purposes. In order to facilitate extraction of

nested fields and missing attributes, the top- k patterns extracted are then compared using edit distance with the highest ranked pattern to test whether the inclusion of this pattern would improve overall performance. Let P_0 be the highest ranked pattern and P_i be any other Pattern. Let $ED = EditDistance(P_0, P_i)$. For all patterns with $ED \leq T_h$, where T_h is a user given threshold, participating data items are extracted and all the data items are subsequently aligned using standard multiple sequence alignment technique[17].

5 Experimental Results and Performance

The implementation of the wrapper has been done as a part of an on going data integration project in our lab. It has been implemented using Java. Java API *suffix trees for natural language* [3] has been used for the implementation and processing of suffix trees, and JTidy [1] has been used in order to ensure the conformation of the input HTML file to the current standard. Moreover, in order to facilitate the subsequent processing of the extracted records, the system offers support for MySQL, DB2, Oracle and MonetDB [2], which needs to be chosen beforehand by the user, as the back end database. The algorithm has been tested thoroughly using the numerous web sites in various domains such as google.com, yahoo.com, altavista.com, citeseer.ist.psu.edu, csbooks.com, ncbi.nlm.nih.gov, droidb.org, david.abcc.ncifcrf.gov, www.genome.ucsc.edu. Initially we tested the wrapper on a large number of web pages (39 real and several hundreds synthetic pages) and in each case we used 20 search terms from each page to extract. We report our results below using 9 such pages because they are representative of the overall result. As is shown, in most of the cases, we were able to extract all the relevant tuples and exclude non-interesting tuples. After running several experiments in these different sites using various search keywords, the average accuracy of the tool reached 97.08% as shown in the graph of Figure 2.

Figure 2. Wrapper accuracy for different sites

Table 1. Running Time comparison

Wrapper Name	Running time
-Road Runner[10]	Exponential with input(without pruning)
-DELA[6]	$O(n \log n)$
-Grammar Induction [7]	$O(n^2)$
-Extracting structured data from web page [5]	Polynomial
-FastWrap	Linear

In graph of Figure 2, the x axis represents different searches performed in different sites and the y axis represents the percentage of data records extracted accurately. It can be seen that the wrapper works reasonably well for all these sites using different search terms. Several searches were performed using different keywords in the same site as well as other sites. The performance measurement to observe here are not only the relevant data extraction but also pruning of the irrelevant data. Precision and Recall have been calculated using the experiment result in order to project their performance. In order to acquire a single performance measure, F Measure, which is the weighted harmonic mean of Precision and Recall, has been calculated and plotted in Figure 3.

Figure 3. F Measure for different search pages

It is evident from the plots and Table 1 that the wrapper outperforms most of the current wrapper mediator system in terms of speed and is comparable with the state-of-the-art in terms of accuracy. The technique proposed here requires a single input HTML page for subsequent processing and it is capable of extracting spatially sparsely located records in the input page. The algorithm works in linear time. This is because all the constituent components of the algorithm beginning from suffix tree generation, longest repeated substring extraction to refining the pattern using KMP prefix algorithm works in linear time. Circular alignment modifica-

tion, if applied, requires larger that linear time complexity. But this is applied to the extracted pattern and for all practical purposes the length of the extracted pattern is much smaller than the input text. Experimental results show that the wrapper generated from the largest repeated pattern, refined by the KMP prefix algorithm, is sufficient in most of the cases. A tabular comparison of running time of the proposed technique with other existing techniques is given in Table 1. Figure 4 depicts the average extraction time for the representative web sites.

Figure 4. Extraction Time for Different Sites

6 Conclusion and Future Research

In this paper, we have presented a novel technique to generate web wrapper for tabular data extraction in linear time. Our system is fully automatic and it generates a reliable and accurate wrapper for web data integration purpose. In this case, no prior knowledge of the input HTML page nor any prior training is required. We conducted experiments on multiple web sites to evaluate our system and the results prove the approach to be promising. We are currently working on extending the functionality of the wrapper by enhancing its applicability to plain text search results as well. Moreover, we are also currently working on establishing a wrapper storage system that will facilitate filtering of the extracted results for integration purposes.

References

- [1] Jtidy html parser and pretty printer in java. <http://jtidy.sourceforge.net/>.
- [2] Monetdb, query processing at light speed:. <http://monetdb.cwi.nl/>.
- [3] Suffix trees for natural language:. <http://stnl.sourceforge.net/>.
- [4] L. Allison. Suffix tree, <http://www.csse.monash.edu.au/~lloyd/tildelgds/tree/suffix/>.
- [5] A. Arasu, H. Garcia-Molina, and S. University. Extracting structured data from web pages. In *ACM SIGMOD*, pages 337–348, New York, NY, USA, 2003.
- [6] C.-H. Chang and S.-C. Lui. Iepad: information extraction based on pattern discovery. In *WWW*, pages 681–688, 2001.
- [7] B. Chidlovskii, J. Ragetli, and M. de Rijke. Wrapper generation via grammar induction. In *ECML*, volume 1810, pages 96–108. Springer, Berlin, 2000.
- [8] W. Cohen, M. Hurst, and L. Jensen. A flexible learning system for wrapping tables and lists in html documents, 2002.
- [9] T. Cormen, Leiserson, Rivest, and Stein. *Introduction to algorithms*. Prentice Hall, 2004.
- [10] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *27th VLDB*, pages 109–118, 2001.
- [11] D. W. Embley, Y. Jiang, and Y.-K. Ng. Record-boundary discovery in Web documents. pages 467–478, 1999.
- [12] D. Gusfield. *Algorithms on Strings, Trees and Sequences, Computer science and computational biology*. Cambridge University Press, 1997.
- [13] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.
- [14] T. Jiang, L. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. In *Proc of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 75–86, London, UK, 1994. Springer-Verlag.
- [15] R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *STOC*, pages 125–136, 1972.
- [16] A. H. F. Laender, B. Ribeiro-Neto, and A. S. da Silva. Debye - date extraction by example. *Data Knowl. Eng.*, 40(2):121–154, 2002.
- [17] M. A. Larkin, G. Blackshields, N. P. Brown, R. Chenna, P. A. Mcgettigan, H. Mcwilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez, J. D. Thompson, T. J. Gibson, and D. G. Higgins. Clustal w and clustal x version 2.0. *Bioinformatics*, 23(21):2947–2948, November 2007.
- [18] Z. Li and W. K. Ng. Wiccap: from semi-structured data to structured data. *Proc. of Intl. Conf. on Engineering of Computer-Based Systems*, pages 86–93, 2004.
- [19] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *Third Annual Conference on Autonomous Agents*, pages 190–197, 1999.
- [20] D. Pinto, A. McCallum, X. Lee, and W. Croft. Table extraction using conditional random fields. In *Proc. of the 26th ACM SIGIR*, 2003.
- [21] G. A. Stephen. *String Searching algorithms*. Lecture Notes series on Computing- vol. 3, 1994.
- [22] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
- [23] J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In *WWW*, pages 187–196, 2003.