Wrapper Generation with Patricia Trees

Sven Meyer zu Eißen Benno Stein
smze@upb.de stein@upb.de

Paderborn University
Department of Computer Science
D-33095 Paderborn, Germany

Abstract The automatic processing of search results that stem from Web-based search interfaces has come into focus, and it will remain important (as long as XML is not a universally applied technology). The reasons for this are twofold: (1) The need for value-added services such as filtering or graphical preparation of search results will increase. (2) The manual creation of tailored parsers for the information extraction from HTML pages cannot keep pace with the fast changing presentation of the search results in right these pages.

Automatic wrapper generation addresses this problem. It means the construction of a tailored parser for a certain type of HTML page with a minimum of manual intervention. This paper introduces the state of the art and presents an own development: A two-stage approach that combines highly efficient suffix matching based on a modified Patricia tree along with a knowledge-based analysis of candidate token sequences.

Key words: Information Extraction, Automatic Wrapper Generation, Wrapper Induction, Web Mining, Information Retrieval

1 Introduction

Web-based search interfaces are widely used as front-ends for information sources such as Web search engines, digital libraries, online shops, and other types of databases. Starting with a keyword search, they generate HTML result pages that contain a list of the found records. Such a semi-structured representation may be adequate for a human reader of this page; however, it is difficult to be further processed by applications that provide value-added services such as filtering, grouping, re-arranging, or graphical preparation: The generated record lists are not directly machine-readable and need to be "disrobed" of their wrapping code. Figure 1 shows an example.

Automatic wrapper generation deals with this problem; it aims at the automatic construction of a parser that extracts the interesting information by finding records and eliminating superfluous HTML code.[1] The following list mentions several challenges for automatic wrapper generation, and it also shows its importance for value-adding services.

[1] The term "automatic wrapper generation" may be misleading; perhaps a better description is "automatic parser generation for wrapping code".

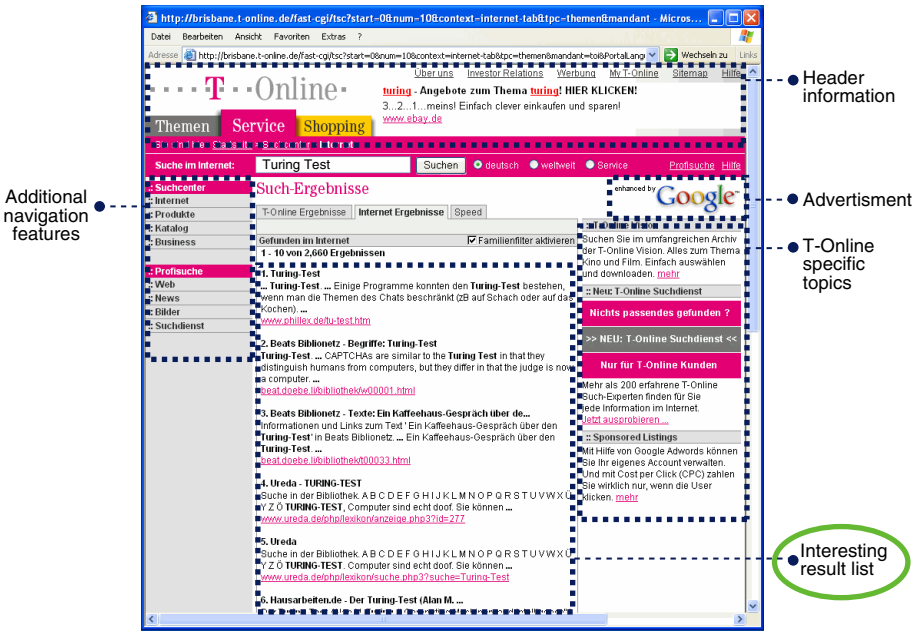

Figure 1. The snapshot shows a clipping of the T-Online search interface. For a subsequent processing of the search results a special block (bottom middle) has to be found as well as correctly parsed.

(1) A result list of records is embedded in header and footer information, which typically contains HTML code for navigation, logos, copyright notices, advertisements, etc.
(2) Each search interface brings along its own concept of wrapping its records, which includes particular font styles, enumeration styles, etc.
(3) The data structure within a record may vary with respect to presentation and data element constraints.
(4) Recurring navigational information like “search more of this” may be contained within a record.
(5) Structure and presentation of a generated list may change every now and then, when the provider modifies the design of the presentation.

The paper in hand is devoted to this problem; more specifically, it focuses on the extraction of records from Web search engines. This work is also related to our AIsearch project where search results from different information sources are grouped thematically within a categorization step [15, 18]. In this context, especially Point 5 is of a high importance, since we experience the generated HTML pages to change frequently. [2]

We present a two-stage approach that combines highly efficient suffix matching based on a modified Patricia tree with a knowledge-based analysis of candidate token

[2] This in turn means that human intervention becomes necessary to adapt in AIsearch the respective parser code for this information source.

sequences. The remainder of this paper is organized as follows. The next subsection gives an overview of existing approaches to automatic wrapper generation. Section 2 introduces our approach, and Section 3 presents some analysis results.

1.1 Classification of Existing Approaches

To hand-craft a tailored information extraction algorithm may be acceptable for a small number of search interfaces; however, in the long run it constitutes a considerable overhead: For each information source, a programmer must identify characteristic HTML patterns that wrap interesting data records, the so-called "extraction patterns", and use them to write a parser. This procedure is tedious and error-prone, and a small change in the design of a result page often renders hand-crafted parsers defective.

For this reason automatic information extraction algorithms have been developed in the last years. They can be divided into wrapper generation algorithms, wrapper verification algorithms, and wrapper re-induction algorithms [9]. Among these, wrapper generation algorithms are the best investigated ones. Their goal is to generate dedicated programs or program parameters like grammars or patterns, which can be used to identify record boundaries within result pages of a particular source.

Supervised wrapper generation algorithms[3] learn extraction patterns from a set of training documents wherein records or attribute boundaries have been labeled manually. The majority of these methods represent the patterns as a finite state machine, e. g. as a grammar, a regular expression, or in the form of a hidden Markov model [11, 1, 7, 3, 17, 5]. The underlying pattern identification algorithms include inductive and active learning strategies [10, 4].

Unsupervised techniques overcome the need to manually label training documents. The approach of Gao et al. uses a set of result pages from the same source and identifies a region that has a "tabular" structure [6]. Based on this table, candidate extraction patterns that will match the rows are inferred. Another system, called IEPAD, discovers repetitive patterns within a result page by means of a Patricia tree and proposes some of them as record candidates [2]. Liu et al. present an approach to extract data from HTML tables, which is based on the analysis of the parse tree of a Web site.

The task of wrapper verification algorithms is to check whether a generated wrapper still behaves as intended, or if design changes within its associated information source lead to a malfunctioning. Kushmerick et al. propose an algorithm that learns a probabilistic model of the extracted data during the training period, which captures data type characteristics like the fraction of numeric attributes within a record [8]. A significant change of the expected data type characteristics in a result page is interpreted as a design change, and intervention may become necessary. Given the case that a verification algorithm determines a malfunctioning of a wrapper, so-called wrapper re-induction algorithms come into play. Lerman and Minton propose a semi-automatic algorithm that also learns a probabilistic model of the data during a training period [12]. If the learned model does not fit the extracted data of a result page any longer, their algorithm maps expected attributes onto data fields within the records of the modified result page probabilistically.

[3] This class of algorithms is also known as semi-automatic wrapper generation algorithms.

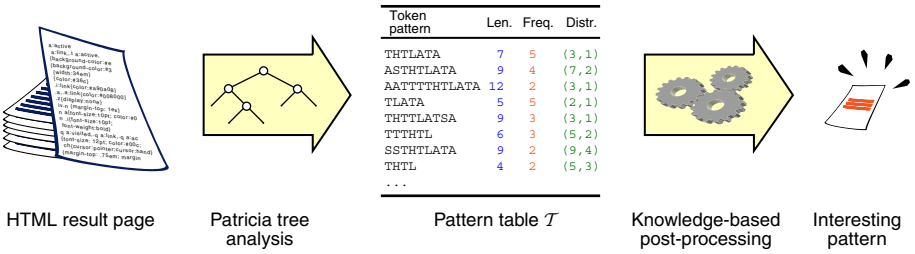

Figure 2. Two-stage approach to automatic wrapper generation: The identification of candidate patterns with a Patricia tree is coupled with a knowledge-based post-processing to find among the candidate patterns the most likely one(s).

In the literature on the subject the term "automatic" refers to the degree of automation in a wrapper generation algorithm for a given source at a given time. As pointed out above, the challenge in a meta-search situation is to construct a parser that is robust against changes of result pages in time. This is what we call adaptive.

2 Adaptive Wrapper Generation for Search Engines

Result pages of search engines contain several regular structures; one of these structures is the list with the snippets that characterize the matching documents for the query and which we would like to identify. A regular structure contains sequences that are tagged in a uniform way. For example at Lycos,[4] a snippet is wrapped in the following code:

```
<li><a>TEXT</a><font color="#808080"></font><br
/>TEXT<span>TEXT</span><a>TEXT</a></li>
```

If one considers the n suffix strings that can be formed from a given HTML page of length n, several among these suffixes start with the same prefix.[5] When inserting the suffixes in a Trie,[6] multiple occurrences of the prefixes can be efficiently detected. Since in an HTML result page several hundreds of such recurring patterns can be found, additional knowledge must be employed to detect those few patterns that actually wrap the interesting snippets. This observation suggests a two-stage approach for pattern identification (cf. Figure 2):

(1) Creation of a table $\mathcal{T}$ with candidate patterns.
(2) Knowledge-based post-processing of $\mathcal{T}$ to identify the interesting pattern(s).

However, a necessary prerequisite is the tokenization of an HTML page, which provides us with a string S of tokens. There is the question of how fine-grained the text

[4] `http://www.lycos.de`
[5] A suffix of a string S is a substring of S that starts at some position $i \leq |S|$ and ends at position $|S|$.
[6] The term "Trie" is derived from the terms "tree", "information", and "retrieval" and designates an index structure for efficient text search [19].

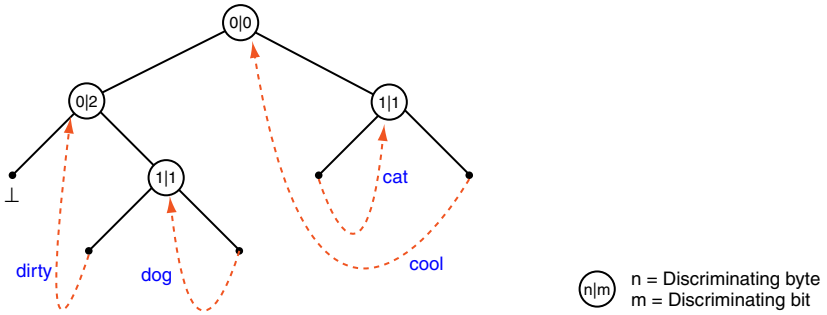

Figure 3. A Patricia tree that contains the words “cool”, “cat”, “dirty”, and “dog”. For performance reasons the discriminating position is encoded as a pair of byte and bit position.

elements in the HTML page shall be distinguished. Chang and Lui distinguish merely two tokens, namely “HTML tag” and “plain text” [2]; to leave more flexibility for the knowledge-based post-processing step we currently support about 130 different tokens.

2.1 Table Creation

Let S be a string (of tokens) of length n. As mentioned above, a Trie provides an efficient means to set up a table of candidate patterns. The theoretically optimum algorithm for constructing an index of all suffixes for S is the suffix tree [14]; its runtime is linear in the length of S. A naive algorithm would generate the n suffix strings of S and insert them in a standard Trie, which results in a runtime of $O(n^2)$.

Our approach is oriented at the naive algorithm: However, the n suffixes of S are not generated explicitly but “read off” by moving an index from 1 to n over S. Every suffix is identified by its starting position, and a Patricia tree (explained below) is used to identify all suffixes that start with the same token sequence. Though the theoretical runtime still is $O(n^2)$ this approach will behave even more efficient than a suffix tree implementation, except for a few pathological cases.[7]

A Patricia tree[8] is a particular digital search tree that has two salient properties: (1) Each inner node in the tree is used for differentiation purposes, say, each inner node has two successors. (2) The keys (strings) are not stored into leafs but into inner nodes, which saves half of the nodes. A Patricia tree has the characteristic of digital search trees in that its structure is independent of the insertion sequence. A digital search tree considers keys as bit sequences; an inner node defines the position of the key that shall be investigated for discrimination purposes. Figure 3 gives an example.

From a Patricia tree all repeating sequences in the token string S can be collected in $O(n)$ runtime and put into a table $\mathcal{T}$.

[7] Rationale for this behavior is that the length of the longest common subsequence in a tokenized HTML page can be assessed by a constant.

[8] The term “Patricia” is an acronym for “practical algorithm to retrieve information coded in alphanumeric”. The data structure was proposed by Morrison [16].

2.2 Table Post-Processing

Typically the table $\mathcal{T}$ of candidate patterns contains more than hundred entries. For reliable identification of the interesting element, all patterns (token sequences) are characterized by several features. The most important ones are: pattern length, pattern frequency, pattern distribution, average pattern distance.

This information is used within heuristic rules that assign positive and negative evidence values to the patterns—example:

```
IF length(p)*frequency(p)/n > 0.2 THEN raise_evidence(p, 2)
```

Here $p \in \mathcal{T}$ designates a pattern; the rule assesses the portion by which p covers the entire token string S. In our current implementation, which focuses on HTML result pages of search engines, the evidence values can be estimated; nevertheless, it is planned to acquire them by a machine learning approach soon.

3 Quantitative Analysis

Adaptive wrappers are generated on the fly, in extreme cases each time a search result is delivered from a search engine. I. e., performance analyses are not only interesting with respect to extraction quality, but also with respect to wrapper generation time. We did some analyses in this connection, based on a test corpus with about 100 generated result pages for several popular search engines. Our wrapper implementation is done in Java, and the experiments were conducted on a Pentium IV 1.2 GHz system running RedHat Linux.

Amount of sample pages	Tokenization	Patricia tree generation	Pattern extraction	Pattern analysis	Total
28KB	121 ms	23 ms	32ms	19ms	195ms

Table 1. Average runtime of different steps in the course of adaptive wrapper generation.

Table 1 shows the averaged runtime for wrapper generation for one result page. The tokenization of the result pages took over 60% of the total runtime, since we used a generic HTML parser that was not optimized for our tokenization step. The heuristic ranking of relevant patterns within the created table $\mathcal{T}$ was always very good, and thus explains the low pattern analysis time. Due to the fact that a wrapper verification algorithm also has to parse and analyze a result page, we do not expect substantial runtime differences between both approaches. Note that it is conceivable to generate an adaptive wrapper on the client machine of a user, in a stand-alone meta search application.

AltaVista	Lycos	Netscape
$\approx 80\%$	$\approx 90\%$	$\approx 90\%$

Table 2. Portion of correctly identified result lists. Basis were about 100 different result pages for each of the mentioned search engines.

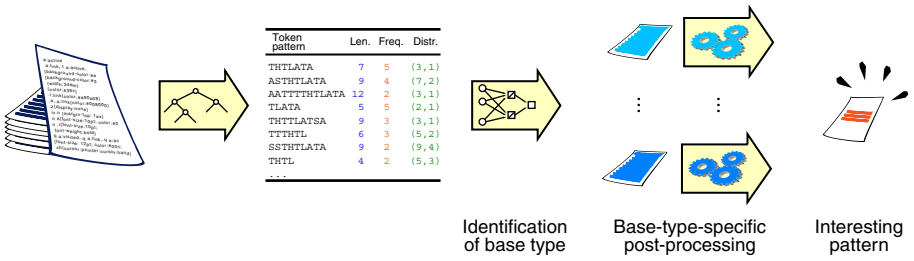

Figure 4. If the process of automatic wrapper generation is organized as a three-stage approach, the knowledge-based post-processing step gains twice: It becomes more effective and easier to be implemented.

Table 2 contains some classification results. The post-processing was able to identify most of the records. However, we employed the knowledge that a record at least contains a URL and a headline.

4 Current Work

The two-stage approach to wrapper generation presented in this paper provides a high degree of flexibility. Nevertheless, the knowledge-based post-processing step becomes more and more intricate with the number of different information sources that shall be handled.

It would be useful in this connection, if a certain “result page base type” is recognized in advance, such as “Shop” or “Link List”, and a dedicated set of rules is chosen and applied in the knowledge-based post-processing step (see Figure 4). In our current work we investigate how the necessary recognition step can be realized by learning a fingerprint from the pattern table.

Moreover, we are developing measures of robustness and flexibility for a generated wrapper in order to prognose both (1) its reliability when parsing HTML pages from information sources the parser was not designed for, and (2) the expected malfunctioning rate depending of extent of modifications of the HTML page.

References

[1] Naveen Ashish and Craig Knoblock. Wrapper Generation for Semi-Structured Internet sources. *SIGMOD Rec.*, 26(4):8–15, 1997. ISSN 0163-5808.

[2] Chia-Hui Chang and Shao-Chen Lui. IEPAD: Information Extraction Based on Pattern Discovery. In *Proceedings of the Tenth International Conference on World Wide Web*, pages 681–688. ACM Press, 2001. ISBN 1-58113-348-0.

[3] B. Chidlovskii, J. Ragetli, and M. de Rijke. Automatic Wrapper Generation for Web Search Engines. In *Proceedings WAIM’00*, LNCS. Springer, 2000.

[4] A. Finn and N. Kushmerick. Active Learning Selection Strategies for Information Extraction. In *ECML-2003 Workshop on Adaptive Text Extraction & Mining*, 2003.

[5] Dayne Freitag and Nicholas Kushmerick. Boosted Wrapper Induction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 577–583. AAAI Press / The MIT Press, 2000. ISBN 0-262-51112-6.

[6] X. Gao, M. Zhang, and P. Andreae. Learning Information Extraction Patterns from Tabular Web Pages without Manual Labeling. Technical report, Victoria University of Wellington, 2003.

[7] C. N. Hsu and C. C. Chang. Finite-State Transducers for Semi-Structured Text Mining. In *Proceedings of IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications*. Pergamon Press, 1999.

[8] N. Kushmerick. Wrapper Verification. *World Wide Web Journal*, 3(2):79–94, 2000.

[9] N. Kushmerick and B. Thomas. Adaptive Information Extraction: Core Technologies for Information Agents. In *Intelligent Information Agents R&D in Europe: An AgentLink perspective*, 2002.

[10] N. Kushmerick, D. Weld, and B. Doorenbos. Wrapper Induction for Information Extraction. In *Proceedings of IJCAI-97*, 1997.

[11] Nicholas Kushmerick and Daniel S. Weld. *Wrapper Induction for Information Extraction*. PhD thesis, Department of Computer Science & Engineering, University of Washington, 1997.

[12] Kristina Lerman and Steven Minton. Learning the Common Structure of Data. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 609–614. AAAI Press / The MIT Press, 2000. ISBN 0-262-51112-6.

[13] Bing Liu, Robert Grossman, and Yanhong Zhai. Mining Data Records in Web Pages. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 601–606. ACM Press, 2003. ISBN 1-58113-737-0.

[14] E. McCreight. A Space-Economical Suffix Tree Construction Algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[15] Sven Meyer zu Eißen and Benno Stein. The AIsearch Meta Search Engine Prototype. In Amit Basu and Soumitra Dutta, editors, *Proceedings of the 12th Workshop on Information Technology and Systems (WITS 02), Barcelona Spain*. Technical University of Barcelona, December 2002.

[16] D. R. Morrison. PATRICIA—Practical Algorithm to Retrieve Information Coded in Alphanumeric. *Journal of the ACM*, 15(4):514–534, October 1968.

[17] Stephen Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning*, 34(1-3):233–272, 1999. ISSN 0885-6125.

[18] Benno Stein and Sven Meyer zu Eißen. AIsearch Homepage. `http://www.aisearch.de`, 2003-2004.

[19] G. Stephen. *String Searching Algorithms*. World Scientific, 1994.