Tree Pattern Inference and Matching for Wrapper Induction on the World Wide Web

Andrew Hogue*

December 7, 2003

Abstract

We develop a method for learning patterns from a set of positive examples to retrieve semantic content from tree-structured data. Specifically, we focus on HTML documents on the World Wide Web, which contain a wealth of semantic information and have an underlying tree structure. A user provides examples of relevant data they wish to extract from a web site through a simple user interface in a web browser. We use the notion of the *edit distance* between trees to combine these examples into a more general pattern. This pattern may then be used to retrieve other instances of the selected data from the same page or other similar pages.

1 Introduction

The World Wide Web is a rich source of relational data. Unfortunately, data on the web is generally formatted for human consumption, making it difficult to develop automated tools for information extraction which are both fast and effective. The technique of *wrapper induction* attempts to solve this problem by constructing templates for extracting semantic classes from a web site based on labeled examples.

Our approach to wrapper induction makes use of the inherently hierarchical nature of HTML documents. HTML is represented internally as a rooted tree, and any block of contiguous data displayed to the user is a single subtree of the full document. Because semantic classes also tend to span a contiguous section of the displayed page, a single semantic class may be represented by a subtree (or a set of sibling subtrees). Thus, when the user highlights an section of the page as a positive example of a semantic class, they are implicitly selecting a subtree of the document.

Once we have a set of positive examples, we attempt to form a general pattern that represents them. We do this using the notion of the *edit distance* between two trees. Edit distance is defined as the fewest number of insertions and deletions that are required to turn one tree into another. A byproduct of computing the edit distance between trees is a mapping between the trees, which may be used to find the greatest common subtree. It is this that we use to form our wrappers. These wrappers may subsequently be used to retrieve other instances of the same semantic classes from different pages.

*ahogue@lcs.mit.edu

The rest of this proposal is structured as follows. In Section 2 we survey related work, both on wrapper induction itself and on more general notions of patterns in tree structures. We define some relevant terms in Section 3, before we attempting to characterize the types of structures we expect to see on the web in Section 4. Section 5 describes possible representations for patterns. Finally, Section 6 describes a proposed user interface for users to specify examples and semantically label them.

2 Related Work

The task of *information extraction*, especially from web documents, has received a lot of attention in recent years. Wrapper induction was first presented by Kushmerick [13], which defined the HLRT class of wrappers. These wrappers were restricted to locating information which is delimited by four types of flags, the "head," "left," "right," and "tail." Because of this limitation, this class was found to successfully wrap only 48% of the surveyed relational data on the web. Similar approaches take advantage of the hierarchical structure of the semantic ontologies of web pages, but not of the hierarchical structure of the documents themselves [16].

Several other approaches to information extraction utilize probabilistic models. Hidden Markov Models offer the opportunity to learn not only the the probabilities but the state structure to represent information in various types of documents [8, 18]. Other models learn to classify data by assuming that "nearby" elements in a hierarchy should be classified similarly [19].

One example of an interactive system for learning patterns on various types of documents is LAPIS [15]. This system provides an interface where users may specify examples relevant to a pattern by highlighting them. Patterns are constructed using a language called *text constraints*, which includes operators such as *before*, *after*, *contains*, and *starts-with*. By using a pre-defined library of parsers which tokenize and label the document, users can create patterns of arbitrary complexity, or allow the system to infer them from examples.

Probabilistic Context Free Grammars, or PCFGs, are a statistical technique for semantically tagging semi-structured data [5]. When used to extract semantic content from English sentences, a probabilistic model is learned by parsing tagged training examples and noting the frequency of occurrences of certain context-free rules among the training data. This model may then be used to tag new sentences by finding the most likely set of rules which would have created that sentence in the model.

Tree pattern matching algorithms have been well studied in fields such as compiler optimization and molecular biology [4, 7, 10]. These approaches mostly focus on matching structures in unordered trees without wildcards. The problem of locating nodes in a tree by specifying the path from root to node is addressed by the XPath standard [3]. A related problem is that of *subtree isomorphism* [11, 14], which attempts to find similarly *shaped* structures in trees, without regard to labeling or sibling order.

The problem of pattern matching on strings also has a long history. The notion of edit distance between strings is a folk algorithm [6], and is used by the `agrep` program [22] to find approximate matches to a query string. One data structure which may have applications for pattern matching in trees is the *suffix tree* [9], which may be used for finding both exact and approximate matches and repeats [21].

3 Conventions and Definitions

An important notion for information extraction is that of an *ontology* for the data being retrieved. An ontology consists of semantic *classes*, representing types of objects which may exist in the world. Classes may have *properties*, which are specific to each instantiation of a class. In this paper, we will use SMALL CAPS to refer to semantic classes and properties. Classes will have the first letter capitalized.

An example of a semantic class is a SEARCH RESULT returned by a web search engine. SEARCH RESULTS have properties like a LINK to the page in question, a DESCRIPTION of that page, and links to SIMILAR PAGES.

4 Characterizing Relevant Structures

To attack the problem of learning patterns for information extraction, we must first characterize the types of structures we expect to see when looking at semantic data on the web. These characterizations are the result of a survey of approximately 25 popular web sites containing structured, semantic data.

4.1 Repeated Instances

The first general characteristic of interesting data on the web is that it tends to "repeat" itself. This is not to say that the *content* is the same, but that the *structure* that displays this content is the same or similar.

The main reason for this repetitive characteristic is that many pages containing relational data on the web are generated by CGI [1] scripts. These scripts retrieve relational data from a database and dynamically publish it to the web using HTML templates. Each template has slots for a certain subset of the data, and the script simply fills the template from its database query before sending the response to the user.

Because CGI scripts utilize the same templates every time the same type of data is displayed on the same page, we see repeated patterns, both on the same page and across related pages. For example, when a user performs a search on Google (http://www.google.com/), they get back a page like that in Figure 1. Note that each SEARCH RESULT follows a similar structure, with a LINK to the result page, a SUMMARY of the result, the HOSTNAME and SIZE of the page, and a link to the CACHED page and to SIMILAR PAGES.

Another type of repeated structure appears *between* pages. When a site displays groups of relational data about one semantic class at a time, we get a structure of one template per type of object (or *class*). Our data structures for patterns will need to be able to carry state between pages, as the user will need to provide examples across multiple pages. A sample from the Internet Movie Database (http://imdb.com) is shown in Figure 2. Here we see two examples of the class MOVIE, served on two different pages, which show the same underlying structure. For instance, they both have a TITLE, a YEAR, a DIRECTOR and WRITERS. Other items such as the PLOT OUTLINE and RATING are also structurally similar.

Figure 1: A sample search on the Google search engine (http://google.com).

Figure 2: Two movie pages from the Internet Movie Database (http://imdb.com).

4.2 Internal and Leaf Nodes

Along with repeating structures, we also notice that there is a pattern to *which parts* of the subtrees under consideration actually change between examples. In almost all cases, the nodes that differ between examples are leaf nodes, or entire leaf-subtrees at the lowest level. The "upper" structure of the examples is uniformly consistent among a single class.

For example, in the Google SEARCH RESULTS depicted in Figure 1, only the text of each result changes. The higher, structural nodes, which affect layout and font selection, are identical between the insances. Another depiction of this phenomenon may be seen in Figure 3 (a), a schematic diagram of the mapping between SEARCH RESULT instances.

The idea that the *top* part of each subtree is the same, but the *bottom* part is not, is suggestive of a similar problem in string matching, where the *prefixes* of certain substrings match, but not the *suffixes*. The solution to matching on strings is to use *suffix trees*. We explore this connection more closely in Section 5.3.

4.3 Subtree Structure

In this section, we attempt to classify the distinct types of subtree structure we have observed in a survey of web pages that contain relational data. We have categorized these types of structure along two axes:

Spanning Elements The nodes that comprise a single example may either be grouped under a *single subtree*, or be spread across *multiple siblings*.

Shared Parents Examples of the same class or property may either *share* a parent node, or be located in *separate* parts of the tree (or even on separate pages).

We now proceed to investigate these axes in more depth, and give examples of their occurrence in practice.

4.3.1 Independent Subtrees

Independent subtrees are characterized by examples in a *single subtree*, grouped under *separate* parents. Example instances may be spread throughout the page, or even be located on separate pages.

Examples include the DIRECTOR, WRITER, or PLOT OUTLINE of an IMDB page, as seen in Figure 2, or each SEARCH RESULT on the Google page in Figure 1. A schematic diagram of this structure may be seen in Figure 3 (a).

4.3.2 Grouped Subtrees

A related structure is a set of *single subtrees*, each representing a separate semantic instance of the same class, grouped under a *shared* parent node.

One example of grouped subtrees are the ACTORS on an IMDB page. A schematic diagram of this structure may be seen in figure 3 (b).

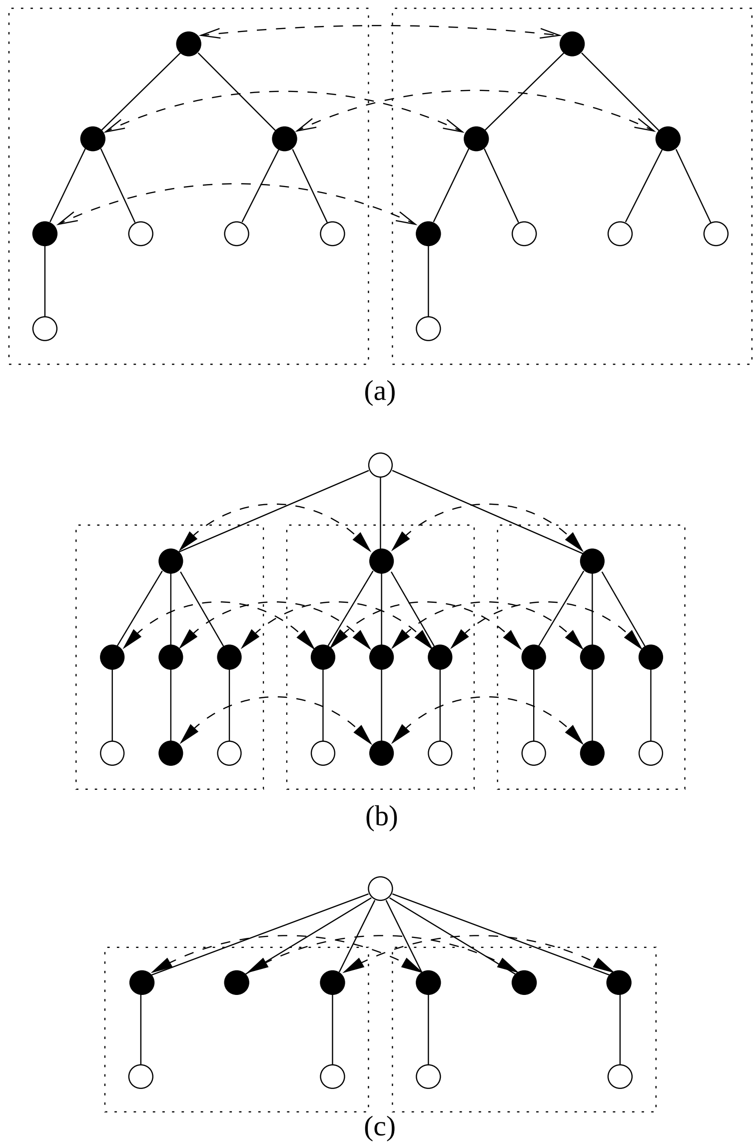

Figure 3: (a) Independent subtrees. (b) Internal subtrees. (c) Internal siblings. In these figures, shaded nodes are included in the best matching, as shown by the dashed arrows. Unshaded nodes are not mapped. The dashed boxes represent separate semantic elements.

4.3.3 Grouped Siblings

A more difficult structure occurs when a single instance of a semantic class is spread across *sibling* children of a single *shared* parent node. These structures are more difficult because there is not necessarily a clear distinction between classes, as there is when semantic classes are grouped into separate subtrees.

An example of this structure may be seen with the STORY class on the New York Times website (http://nytimes.com). A schematic diagram of this structure may be seen in figure 3 (c).

4.3.4 Independent Siblings

The final and most difficult type of structure is that of *siblings* which occur independently throughout the page with *separate* parent nodes. However, semantic data with this structure seems to occur rarely in practice, and we did not encounter any examples in our survey.

5 Data Structures for Patterns

Given the various types of expected structures described above, we need to find an appropriate representation which is flexible enough to represent them, but at the same time specific enough to exclude parts of a page which are irrelevant. The pattern structure should be easily constructed from the subtrees which represent positive examples given by the user. It should also support fast matching against new documents to find other instances of the pattern.

5.1 Best Mapping

For semantic information that exists in independent subtrees in the page, one approach is to compute the *best mapping*, between positive examples. The best mapping is a byproduct of computing the *tree edit distance* between the examples, first defined by Tai [20] and improved by Zhang [23]. By removing any elements which are deleted or changed in the best mapping between the examples, we are left with a "skeleton" tree which contains all of the common structural elements of the examples, but eliminates those parts which are specific to a single example. We call this skeleton tree the *greatest common subtree*, or GCS.

For instance, in each part of Figure 3, the common elements are represented by shaded circles, while specific elements are unshaded.

In addition to being useful for representing the independent subtrees of Section 4.3.1, the GCS may also be modified to easily represent grouped subtrees like those in Section 4.3.2. By allowing nodes of the GCS to match more than once against the target tree, a single GCS can match all of the grouped subtrees under a single parent, as in Figure 3 (b).

5.2 Finite Automatons

While the GCS of the positive examples is useful for creating patterns when semantic classes are easily separated into separate subtrees, it runs into problems when the classes are stretched across several siblings of the same parent. These issues are a result of the

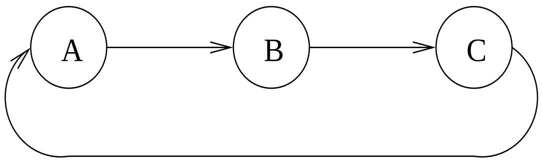

Figure 4: An example finite automaton, built for the set of children $\{A, B, C, A, B, C, A, B, C\}$.

inability for the tree edit distance algorithm to independently map examples to each other, because they share sibling nodes.

One approach to this problem which we will consider is representing child nodes in the pattern as *finite automata*. This gives us greater control over the way that pattern nodes match against the target tree. The algorithm will first identify an "alphabet" of subtrees which make up the siblings, and then use this alphabet to create a finite automaton which will be used to model the pattern.

An example automaton, built for the set of children $\{A, B, C, A, B, C, A, B, C\}$, is shown in Figure 4. The repeating pattern $\{A, B, C\}$ is captured by the state transitions in the automaton. This ability to capture repeating patterns among siblings is vital to being able to effectively pattern match structures where semantic classes are not easily grouped into separate subtrees.

5.3 Suffix Trees

As mentioned in Section 4.2, we expect most patterns to be defined by similarities in internal nodes, and differences in leaf nodes. This type of structure in trees is an analog of the structure that suffix trees take advantage of in strings.

Kosaraju [12] utilizes the notion of a "suffix tree of a tree" in his tree pattern matching algorithm. The suffix tree is constructed using all paths in the pattern and the target tree from leaf to root, and then clusters of nodes are computed using the suffix tree.

We plan to further explore the applications of suffix trees to ordered, labeled subtrees, both for pattern matching and for finding repeated instances of the same or similar subtrees within the same tree.

6 User Interface

The classic alternative to wrapper induction has always been to write a script which parses the source of a given page and retrieves the relevant information. Unfortunately, this tactic is difficult at best, and all but impossible for non-technical users. In order for wrapper induction to be useful to these users, an intuitive, easy-to-use interface must be developed.

The requirements for this interface are as follows:

- Users must be able to highlight or indicate relevant content on a page.
- Users must be able to create and/or assign both semantic classes and properties to the selected content.

- Users must be able to act on the patterns that are created, or request that actions be taken on their behalf.

We plan to accomplish these goals by integrating the wrapper induction algorithm into the Haystack [17] platform. Haystack offers a rich environment for managing semantically labeled data using the RDF [2] standard.

Several tools which do not currently exist in Haystack must be developed. First, while there is a web browser, it does not have the functionality to map highlighted sections of the page into subtrees in the HTML source. These subtrees must then be run through the wrapper induction algorithm and stored as RDF.

Users must also have the ability to select and label an instance of an entire semantic class in a page. They should then be able to label certain properties of that class within the same example. These labelings will further help the wrapper induction algorithm to find the best mapping between examples.

In addition, an interface must be developed to enable users to work with ontologies. As stated above, when users select an example, they must be able to label it with semantic tags. To accomplish this, users will need to browse existing ontologies, as well as to create new classes and properties.

As an example, a user might wish to label a Google search result page with semantic information. They would first highlight and select an entire search result, and label it with the class SEARCH RESULT. The user would then proceed to label the parts of the instance with properties such as the LINK, SUMMARY, or link to the CACHED page. Once one example was labeled, the user would then proceed to label another example, and the two could be combined into a full pattern for retrieving search results from a Google page.

7 Conclusion

The main goal of this proposed thesis is to develop an effective, intuitive method for creating wrappers for semantic data on the World Wide Web. To this end, we plan to develop data structures for storing hierarchical wrappers, an algorithm for inferring them from positive examples, and a user interface to enable their creation. The effectiveness of our solution to these problems will be seen in the number and variety of semantic data that is able to be captured by these wrappers, as well as the usefulness that the system presents to common users.

References

[1] The CGI specification, Version 1.1. http://hoohoo.ncsa.uiuc.edu/cgi/interface.html.

[2] Resource Description Framework (RDF) specification. http://www.w3.org/RDF, 1999.

[3] XML Path language (XPath) specification. http://www.w3.org/TR/xpath, 1999.

[4] Cole, Hariharan, and Indyk. Tree pattern matching and subset matching in deterministic $O(n \log^3 n)$-time. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1999.

[5] Michael Collins and Scott Miller. Semantic tagging using a probabilistic context free grammar. In *Proceedings of 6th Workshop on Very Large Corpora*, Montreal, Canada, 1997.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[7] Moshe Dubiner, Zvi Galil, and Edith Magen. Faster tree pattern matching. *J. Association of Computing Machinery*, 41(2):205–213, March 1994.

[8] Dayne Freitag and Andrew McCallum. Information extraction with HMM structures learned by stochastic optimization. In *AAAI/IAAI*, pages 584–589, 2000.

[9] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, New York, 1997.

[10] C. M. Hoffmann and M. J. O'Donnell. Pattern matching in trees. *J. Association of Computing Machinery*, 29(1):68–95, January 1982.

[11] J. E. HopCroft and R. E. Tarjan. *Isomorphism of Planar Graphs*, pages 131–152. Plenum Press, 1972.

[12] S. R. Kosaraju. Efficient tree pattern matching. In *Proceedings of the 30th annual IEEE Symposium on Foundations of Computer Science*, pages 178–183, New York, 1989. IEEE.

[13] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[14] D. W. Matula. An algorithm for subtree identification. *SIAM Rev.*, 10:273–274, 1968.

[15] Robert C. Miller and Brad A. Meyers. Lightweight structured text processing. In *Proc. of USENIX 1999 Annual Technical Conference*, pages 131–144, Monterey, CA, USA, June 1999.

[16] Ion Muslea, Steve Minton, and Craig Knoblock. A hierarchical approach to wrapper induction. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proc. of the Third International Conference on Autonomous Agents*, pages 190–197, Seattle, WA, USA, 1999. ACM Press.

[17] D. Quan, D. Huynh, and D. R. Karger. Haystack: A platform for authoring end user semantic web applications. In *Proc. 2nd International Semantic Web Conference*, 2003.

[18] Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI 99 Workshop on Machine Learning for Information Extraction*, 1999.

[19] Lawrence Shih and David Karger. Learning classes correlated to a hierarchy. Technical report, MIT AI Lab, 2001.

[20] Kuo-Chung Tai. The tree–to–tree correction problem. *J. Association of Computing Machinery*, 26(3):422–433, July 1979.

[21] Esko Ukkonen. Approximate string–matching over suffix trees. In *Proc. 4th Annual Symposium on Combinatorial Pattern Matching*, pages 229–242, 1993.

[22] S. Wu and U. Manber. Agrep – a fast approximate pattern-matching tool. In *Proceedings USENIX Winter 1992 Technical Conference*, pages 153–162, San Francisco, CA, 1992.

[23] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Computing*, 18(6):1245–1262, December 1989.