Using HTML Formatting to Aid in Natural Language Processing on the World Wide Web

Dan DiPasquo

Senior Honors Thesis

June 23, 1998

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Because of its magnitude and the fact that it is not computer understandable, the World Wide Web has become a prime candidate for automatic natural language tasks. This thesis argues that there is information in the layout of a web page, and that by looking at the HTML formatting in addition to the text on a page, one can improve performance in tasks such as learning to classify segments of documents. A rich representation for web pages, the HTML Struct Tree, is described. A parsing algorithm for creating Struct Trees is presented, as well as a set of experiments that use Struct Trees as a feature set for learning to extract a company's name and location from its Web pages. Through these experiments we found that it is useful to consider the layout of a page for these tasks.

Contents

1. Acknowledgements

There are many people who helped me accomplish the work for this thesis. I would like to thank the Text Learning Group at Carnegie Mellon for all the support that I received this year. Specifically I would like to thank Dayne Freitag, Rosie Jones, Johannes Frnkranz, Andrew McCallum, Kamal Nigam, and Sean Slattery all the ideas and inspiration. Thanks also to Lauren Riley for helping revise the final draft of this thesis.

Over this year, there were two people to who I owe the greatest thanks. During the year both Tom Mitchell and Mark Craven truly took me under their wings and helped me in countless ways. Mark, I would like to thank you for all the times you helped to give me direction. And finally Tom, I would like to thank you for the motivation, inspiration, and all the time you gave me throughout this year.

2. Motivations

The World Wide Web is defined by the World Wide Web Consortium (W3C) as "the universe of network-accessible information, the embodiment of human knowledge" [1]. While the Web may not currently live up to this definition, it is true that with over 200,000,000 pages and its incredible rate of growth, it is quickly moving in this direction.

Unfortunately, because of the nature of the Web, even if it were the embodiment of human knowledge, accessing this information would be a difficult task because there is no ordering or index for Web pages. Currently, to find interesting information on the Web, one must rely on a keyword search (Lycos, Altavista) or on the small percentage of the Web which has been indexed by hand(Yahoo). Both of these options are an ineffective means of accessing the information available now, and as the Web continues to grow so will this problem.

For these reasons, the Web has become a prime candidate for systems that attempt to automatically catalog information into an ontology [14], automatically extract knowledge from highly regular [9, 16] and semi-regular [5] sources, and use agents to actively search the Web [6, 21].

Traditional natural language processing techniques seem immediately applicable to the Web, but problems arise when applying an algorithm designed for flat text for processing Web pages. These techniques often expect complete, well formed sentences, but on the Web the norm is short informative sentence fragments. Often, information is conveyed through visual cues such as lists, tables, and hyperlinks. For example "The following people are members of this group." would likely be expressed as a heading "Members" followed by an itemized list of members names.

We argue that there is information inherent in the layout of each page, and that by parsing the HTML formatting appropriately one can improve upon traditional text processing techniques. In general we argue that looking at this formatting could help for many tasks that involve natural language on the Web, but in this thesis, we consider the more specific problems of information extraction and information retrieval over over a Web documents. The next section describes the current state of the art in hyper-text processing. We will then present a novel representation for Web pages, a heuristic based parsing algorithm for this representation, and a set of experiments over two domains that support our claim.

3. Related Work

Our research is closely related to several areas which have been heavily investigated. In this section, we will look at the current state of the art in text processing, and specifically text processing on the web.

The traditional information extraction task (IE) is defined by the Message Understanding community (MUC) [2, 3] as given a document, automatically find the essential details in the text (i.e. given a newswire article about a terrorist attack, find the names of the attacking group and the victims of the attack). This has been shown to be applicable on the Web for extracting specific information from a Web page such as the name from a personal home page. This differs, however, from the traditional work done on flat text. Most of these early techniques relied on hand crafted algorithms for extracting information. Recently, much research has been focussed on using machine learning for information extraction tasks [7, 10, 12, 19, 20, 23], and this has shown the most promise for information extraction from the Web.

Attempts to extract information from the Web have been of primarily two flavors: extraction from highly regular sources such as humanly readable text that has been automatically produced from a database, and generalization of machine learning techniques to handle unstructured text. The first of these is often reduced to the more general problem of "wrapper induction," for which much work has been developed [5, 13].

Webfoot is an attempt at more general information extraction on the Web. It uses CRYSTAL, an information extraction system that takes parsed annotated sentences and finds patterns for extraction in novel sentences. Rather than taking parsed sentences, Webfoot takes fragments which are broken up by looking at HTML [24].

Finally, **SRV** is a learning architecture for information extraction. It takes a user defined feature set, and hand tagged training documents, and attempts to learn rules for extraction [11]. When **SRV** was given Web Pages, and feature sets that included some HTML information, it was able to learn rules for extracting from the Web [10].

At recent Text REtrieval Conferences (TREC), much work has been reported on document classification. Craven, *et al.* [14] shows that document classification is useful for finding interesting ontology entries on the Web. Many traditional document classification techniques view a document as a "bag of words" in which order does not matter [15]. These simple techniques generalize well to the Web because they are not affected by lack of traditional syntactic structure. However, on the Web, these techniques can be improved by taking advantage of other sources of information. Craven, Slattery, and Nigam report in [8] that greater accuracy can be achieved by representing the Web as a graph in which each page is a node and each hyperlink an edge. By giving a first order learner the words on each page and this graph as data, one can exploit this information. Favorable results were also obtained in IE tasks on the Web by using this technique. In this paper, we propose a representation that accounts, instead, for the internal structure of each page. We also suggest a method for combining these representations to account for both internal page structure and external graph structure.

4. HTML Struct Trees

4.1. The Representation

When processing flat text, several syntactic conventions such as the grammatical structure of sentences are exploited. The field of linguistics explores these conventions in written text and speech genres; however, it has been proposed that hypertext is a different genre than flat text and that we must explore its linguistic conventions [4]. Our proposal is that the hypertext on the World Wide Web (HTML) has some linguistic conventions not only in its hyperlinks, but also in its page layout. This section proposes a representation that could be useful in exploiting these conventions. It is not our goal to study explicitly what these conventions are, but rather to create a representation that would allow a system to take advantage of them.

The HTML Struct Tree is intended to represent how a human would parse word relationships on a Web page due to its layout. Because Web pages are designed to be read by people, if we could parse them properly, we would be able to capture the relationships between objects on the page and provide ourselves with a framework for exploring existing conventions on the Web (i.e. an HTML Struct Tree should recognize the relationship between a heading above a list and a list item, much like a parser of written English can detect a relationship between an adjective and the noun it is modifying).

An HTML Struct Tree is a n-ary tree in which each node corresponds to an HTML tag on the page. The parent-child relationship implies that the child is in the scope of the parent. Figure 1 shows a Web page as viewed through the Netscape 3.01 browser, and the resulting Struct Tree. Each word on the page belongs to the innermost HTML tag in which it is enclosed. For example under the heading **Research: Machine Learning, Computer Science**, the first item in the nested list has the following HTML source:

```
<li> How can software agents learn to help their users?
 <ul>
 <li> See our <A
 href="http://www.cs.cmu.edu/afs/cs/project
      /theo-4/text-learning/www/index.html">
 text learning and software agents </a> research
</ul>
```

In this example, the text `See our text learning and software agents research` is enclosed by several tags (<ul>   ), but in our representation, it only belongs to the innermost <li> tag.

Each parent-child relationship means loosely that the parent tag and the child tag are directly related. Because each sub-tree is a section of the page and a parent child relation means that the two nodes are closely related, we can view the root node of each sub-tree as descriptive of the that section of the page, and each child of that node as descriptive of a subsection of the page. In Figure 1, the title node is an ancestor of all other nodes, which is consistent with the intuition that the title of the page is used to briefly describe the contents of the page. We see that the <H3> headings **Textbook: Machine Learning** and **Courses** are siblings sharing the <H1> heading **Tom Mitchell's Home Page** as a parent. The reason for this is that the <H1> heading describes a large section of the page, and each <H3> heading describes a subsection of that.

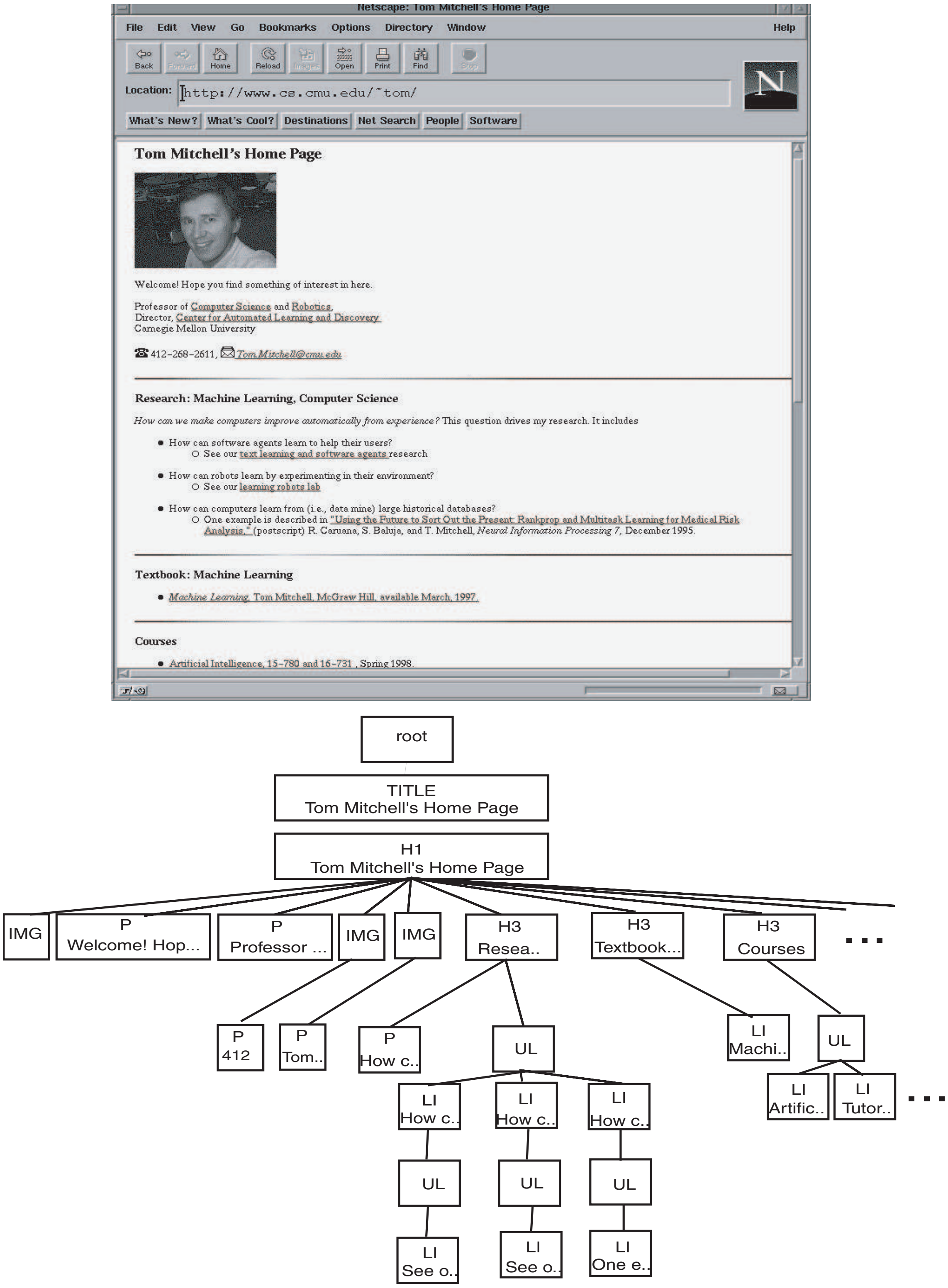

Figure 1: This page is parsed into an HTML Struct Tree. Each node in the tree corresponds to a HTML tag and its enclosed text on the Web page. The parent-child relations mean roughly that the nodes "are related."

contains_address(Node) :- parent_of(Node1, Node), has_phrase(Node1, "Contact Information"), has_number(Node), has_city(Node).

Figure 2: A hand coded rule for extracting addresses from corporate home pages.

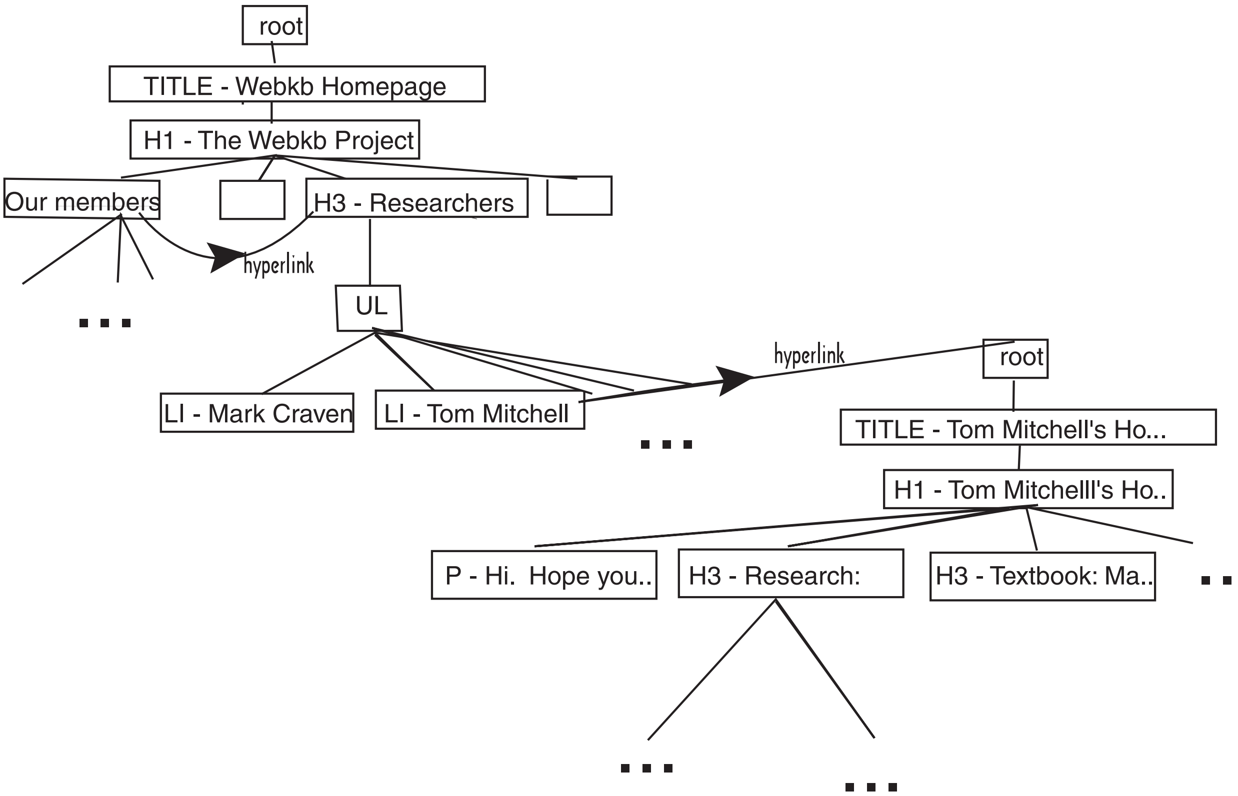

Figure 3: A view of the Web as a forest of Struct Trees in which any two nodes can be connected by a hyperlink.

The ordering on the page is preserved in that children are ordered from left to right in the order that they appear on the page.

The utility of representing a Web Page by a Struct Tree can be illustrated by looking at an example in which Web authors often use page structure to express important information. A common bit of information that companies publish on their Web Pages is a mailing address. There are a number of different ways that the author can display this; however, most of these involve marking the information with the words "Contact Information" or "Address." This is commonly done through the use of a HTML table or a heading followed the address. Because there are different heading sizes and ways of representing this through a table, it would be difficult to write an extractor that recognized every case of this. But by viewing the page as a struct tree, we can write a simple rule that accounts for most of these cases. Figure 2 shows a hand coded rule for this purpose.

This representation deals with the internal structure of HTML, but it is incomplete, since it ignores hyperlinks, which are a primary component of hypertext. To account for this, we view the Web as a directed graph in which each node, as defined above, is a vertex. In the graph there are two types of edges, Struct edges and hyperlink edges. Struct edges connect each Struct Tree parent to its children creating a forest of Struct Trees. Hyperlink edges connect the first node in which the anchored text is located to the root of Struct Tree that this link points to. Often, this edge points to the root of an entire Struct Tree, but the representation extends to the case of internal links to anchored text. If there is a link that points into the middle of a page, the hyperlink edge simply points to the root of the smallest subtree which contains the anchored text. (see Figure 3)

The next section describes an algorithm that can properly parse a class of Web pages into this representation.

4.2. The Parsing Algorithm

This section looks at the parsing algorithm we used to obtain approximations of HTML Struct Trees for HTML source. This procedure is non-trivial because it must decide which nodes are "related." Lets first assume that we were given a function called Parent_Of(Tag1, Tag2) that can tell us if Tag1 is the parent of Tag2. We can then build a simple algorithm for parsing Struct Trees. We will first describe this algorithm, and then explain the Parent_Of function that we used.

Another concern when processing Web Pages is that the vast majority of HTML on the World Wide Web is invalid when parsed with the W3C's official specifications. Because Web browsers use a much more flexible parsing algorithm, this invalid HTML still appears as the author intended it to. For this reason, the first step to parsing Web pages into Struct Trees was to correct for invalid HTML. One example of this is that most web authors do not use end tags for certain elements (li, td, ect.). This preprocessing creates mostly valid HTML by relaxing approximately the same rules that the Netscape Navigator Version 3.04 does.

After this step, the text on the page is tokenized using alphanumeric lexing (tokens are

element	*value*
root	-1
title	0
h1-h6	1-6
other	7
dd	8

Table 1: A partial ordering of HTML elements.

separated by non-alphanumeric characters). Although the HTML tags are not tokenized, the type, start position, and end position of all [1] are saved in a vector in the order of their starting position. Once this vector is obtained, the algorithm is straightforward.

For each page, a root node is created, and the tag in the first position is made a child of the root node. We then make a linear pass down the vector. For each tag, we check to see if the preceding tag is its parent using the Parent_Of function. If the preceding tag is not the parent of the current tag, we then check to see if the parent of the preceding tag is the parent of the current tag. We continue up the ancestry of the preceding tag checking to see if a tag is the parent of the current tag until we either find a parent, or hit the root node, at which point we make that node the parent of the current tag.

4.3. The Parent_Of Function

In this paper, we make use of two simple heuristics to determine if HTML tags are directly related, though one could imagine attempting to learn a better measure for this automatically. These heuristics were developed by hand by parsing numerous Web pages from various company and university sites. If either if these two conditions were true, Parent_Of returns TRUE, otherwise it will return FALSE.

The first is that tag1 is the parent of tag2 if tag2 is *included in* tag1, where *included in* means that

$$(\mathsf{tag1.start\_position} < \mathsf{tag2.start\_position})\mathrm{AND}(\mathsf{tag1.end\_postion} > \mathsf{tag2.end\_position})$$

The reason for this heuristic is that many people use lists and tables to organize their pages into sections. By looking at which tags *include* others, we can get a picture of the organization of the page, and therefore which areas are related to which.

The other heuristic is based on the convention that people use certain tags, and particularly headings to organize text. We defined a partial ordering for HTML tags, and if value of tag1 was less than that of tag2, then tag1 is the parent of tag2. Figure 1 shows the ordering we used for the experiments in the next section.

[1] Actually tags that affect the font such as <i> and <b> are not included.

Although these two heuristics can not find all the relationships between tags, we think they generally don not find relationships where there are none.

5. Learning Rules for Information Extraction

In the previous section, we provided a representation that can express the formatting of a Web Page. In this section we attempt to use this representation for a real world text processing task. We will show that this representation can be used in conjunction with a first order rule learner for information extraction tasks on the Web. Given various company home pages collected on the Web, we will attempt to extract the company's name and each country in which it has operations. We will attempt these extraction tasks by learning from just the text on the page and from the text and various amounts of information derived from formatting the page. Through this we will attempt to show the utility in looking at the HTML formatting of Web pages. Our hypothesis is that we will be able to achieve better performance by looking at the text and the HTML.

5.1. The Data Set

We attempted one task in which given a Web page, we tried to find the Struct nodes from that page that contained the name of the company. For these experiments we looked only at the index page for 906 companies from multiple economic sectors. Because we had each company's name recorded in a knowledge base, each positive instance was automatically tagged. Overall, there were 3352 such instances.

In the other task, we attempted to extract the names of countries in which a company had some sort of operation. For these experiments we looked at the complete Web site[2] of 38 oil companies. This collection consisted of 788 pages. A dictionary search was performed on each page for country names, and each of these hits was then labeled by hand as a positive or negative instance. The criterion used for labeling an instance, X, was the following: "Does the section of text around X describe an operation that this company has in the country whose name was matched?" Overall, there were 338 positive instances and 358 negative instances. Therefore, a simple key word search would have an accuracy at this task of 49%.

5.2. The FOIL-PILFS Algorithm

We used the FOIL-PILFS [22] algorithm which combines Quinlan's FOIL relational learning algorithm [17, 18] with a naive bayes text classifier.[15] This algorithm was designed for learning in hypertext domains.

FOIL-PILFS, like FOIL, conducts a hill climbing search through possible predicates; however, unlike FOIL, at each step in the search, FOIL-PILFS invents a new predicate by

[2]We considered a site the pages in that company's domain.

training a naive bayes classifier to distinguish between positive and negative instances. This predicate, like the others, will be chosen if it results in a clause that has a lower m-estimate error rate than any other alternative.

5.3. The Feature Set

We parsed each page into Struct Trees using the algorithm described, and from these trees we obtained the following background relations.

Tree Relations:

ancestor(Node1, Node2): Node1 is an ancestor of Node2 in the struct tree.

parent(Node1, Node2): Node1 is a parent of Node2 in the struct tree.

sibling(Node1, Node2): Node1 is a sibling of Node2 in the struct tree.

ancestorhead(Node1, Node2): Node1 is an ancestor of Node2 and a heading in the struct tree.

parenthead(Node1, Node2): Node1 is the closest ancestor of Node2 in the struct tree that is also a heading.

HTML Relations:

heading(Node, Page): Node is in an H1-H6 heading on Page.

li(Node, Page): Node is list item on Page.

list(Node, Page): Node is ul, ol, or dl element on Page.

listortable(Node, Page): Node is a "list" or a table element on Page.

paragraph(Node, Page): Node is a paragraph on Page.

table(Node, Page): Node is a table element on Page.

td(Node, Page): Node is a td element on Page.

title(Node, Page): Node is a title element on Page.

tr(Node, Page): Node is a tr element on Page.

The rest of the feature set was the text associated with each node in the Struct Tree, on which FOIL-PILFS could train its text classifiers.

5.4. Finding Company Names

Our task with this data set was to find Struct Tree nodes in which the name of each company was located. We ran FOIL-PILFS giving it all the background relations, holding out the Tree Relations such as parent and ancestor, and giving it no relations except for the text in each node. The reason we ran the latter two experiments was to see if there was information in different types of formatting data. If the learner does better when presented with data about the HTML formatting then it does without it we can conclude that the formatting data does contain useful information in this domain. Because our goal is to find the name of each company, we divided the companies into 5 sets, such that each company's nodes were in the same set. We then ran leave one set out cross validation.

method	Precision	Recall	Coverage
FOIL-PILFS HTML and Tree Relations	66.0%	27.1%	66.1%
FOIL-PILFS HTML Relations	66.4%	27.3%	65.9%
FOIL-PILFS No Relations	57.5%	21.2%	46.7%
Random Selection	9.82%	26.5%	9.82%

Table 2: Precision, Recall, and Coverage results from the company name experiments

In table 2 we see the for each of these runs. To analyze these we calculate numbers for Recall (R) and Precision (P) as follows:

$$R = \frac{\text{\# correct positive predictions}}{\text{\# of positive examples}} \quad , \quad P = \frac{\text{\# correct positive predictions}}{\text{\# of positive predictions}} \quad .$$

The column labeled Coverage is the percent of company pages on which at least one correct name was located. The Random Selection row are the results obtained by randomly picking one node from each page in the data set.

The first thing we notice is that by our approach clearly beats random selection, and looking at these results we can also see that the two runs that had the HTML relations to work with clearly beat out the run without them. This shows that there was information in the HTML formatting of the page. Specifically, this experiment shows that it is useful to segment the page based on the HTML, and also that it can be useful to label these segments based on the HTML elements associated with them.

The run that had the tree relations, however, was slightly worse than the run without these relations. This is not necessarily evidence that there is no information in the tree structure of these pages. The highest recall reported for any of the three runs was 27.3%. This shows that over 70% of the positive instances were not found. We explain this because utilizing structure for the desired gain in information will often require the addition of more than one predicate, and an algorithm that adds predicates greedily will likely never find such a relation. Perhaps if we could better learn rules about the structure of these pages, we could raise our recall.

5.5. Finding Company Locations

In this experiment we attempt to find countries in which a company has operations. We use two techniques for this. One technique is a simple key word search for any country names located in a company's Web site. The other technique, which was also used in the name finding task, is to segment the page by its HTML elements and then to decide which of these segments contained correct country hits. We hypothesize that by splitting segmenting the page in this fashion, we will attain better results.

method	Precision	Recall
FOIL-PILFS HTML and Tree Relations	71.6%	64.4%
FOIL-PILFS HTML Relations	68.9%	60.7%
FOIL-PILFS No Relations	82.8%	58.9%
Dictionary Search	48.6%	100%

Table 3: Precision and Recall results from the company location experiments

In the segemented experiments, we trained and tested only on segments that contained a country name. To decide whether a segment was a positive instance, we again used the FOIL-PILFS algorithm, and as in the name finding task, we gave the learner the three different feature sets to help determine the importance of varying amounts of structure information. We also used a stoplist containing the names of all countries which forced our learner to recognize important words other than these names. To test these we split the nodes into 5 sets and used leave one set out cross validation.

```
haslocation(Node1, Page1) :- naivebayes(Node1), parent(Node2, Node1), ancestor(Node1, Node3), not(table(Node2, Page)).
```

where the ten most highly weighted words for the naive bayes classifier (after stemming) were:
in, explor, field, safat, compani, refineri, project, occident, ventur, and petrochem.

Figure 4: A learned rule that takes advantage of structural information

In table 3 we see the results for each of these runs. All three of the experiments that make use of formatting clearly beat the dictionary search baseline, but, we achieve a much higher precision by ignoring much of the information in the Struct Tree. We can explain this in two ways.

First, many companies express information about where their operations are located in large chunks of text. A naive bayes classifier is very good at differentiating this type data. When we couple this with the fact that FOIL is a covering algorithm without a strong stopping criterion, it follows that the extra information about nodes could make it easier to over fit the training data.

The second explanation for this may be the learner itself and specifically its use of a greedy search through the hypothesis space. Proof for this lies in comparing the results from the run including all the relations and the one that included only the HTML relations (and not the Struct Tree relations). Although the prior only beat the latter by a small percent, if we examine one of the rules produced by the learner using all the relations, we can see that there could be information in the structure that the FOIL-PILFS learner could not find.

The rule in figure 4 demonstrates both that there is information in the structure of these pages, and also that a hill climbing search is ineffective for finding it. The learner figured out that nodes in certain positions in a table are unlikely to contain a country location, given the naive bayes classifier it learned earlier. By looking at the most indicative words in the naive bayes model, we can see that the positive nodes that this classifier is learning from are probably in sentences or at least sentence fragments. Because sentences are most likely less frequent in table rows, this shows information in the structure of the page. This rule also shows the difficulty in finding structure in this approach. To find gain in adding the not table(Node2, Page) literal, the learner had to first "get lucky" by adding parent(Node2, Node1). The problem with a greedy search in the HTML domain, is that there is often no immediate gain in adding a single literal, but there could be in adding multiple literal.

We can see from the variance in our two experiments that the amount our system can learn from the formatting is domain specific. One interesting point is that in the company location task, much of the text was formatted in large paragraphs. This would account for the fact that the text itself, once broken up, was enough to achieve good results, but in the name finding task, more information was present in how the text was presented.

6. Conclusions

We started this paper with the proposition that there is information inherent in the HTML formatting on the World Wide Web. By creating a representation, we were able to exploit this information to aid in two information extraction tasks.

We also showed that relational learning can be a useful tool for exploiting the internal structure of Web pages in addition to the graph structure of hypertext. In this paper, we looked only at the internal structure of each page, but in the future we plan to combine the internal and external structure as suggested in section 3.1.

In the name location task, the HTML labels were important in achieving good results, while in the location extraction, it was useful only in splitting the page into small regions. In both cases, the tree structure was rarely used, but this can be attributed to the learning algorithm. Another area for future work could be to apply other algorithms to text processing tasks that do not utilize a strictly greed search strategy.

A final area for future research is to better our representation for Web Pages. In this paper we defined a representation that could be expanded. The parsing algorithm to approximate this representation was also very simple. An interesting problem might be to automatically learn such a structure.

References

[1] About the world wide web. http://www.w3.org/WWW/.

[2] *Proceedings of the Fourth Message Understanding Conference (MUC-4)*, McLean, Virginia, June 1992. Morgan Kaufmann Publisher, Inc., San Francisco.

[3] *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, Baltimore, Maryland, August 1993. Morgan Kaufmann Publisher, Inc., San Francisco.

[4] E. Amitay. Hypertext: The importance of being different. Master's thesis, University of Edinburgh, Centre for Cognitive Science, 1997.

[5] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. In *ACM SIGMOD Workshop on Management of Semi-structured Data*, 1997.

[6] M. Balabanovic and Y. Shoham. Learning information retrieval agents: Experiments with automated web browsing. In *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, 1995.

[7] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Papers of ACL-97 Workshop on Natural Language Learning*, 1997.

[8] M. Craven, S. Slattery, and K. Nigam. First-order learning for web mining. In *Proceedings of the 10th European Conference on Machine Learning*, Chemnitz, GERMANY, April 1998. Springer Verlag.

[9] R. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the world-wide web. In *Proc. Autonomous Agents*, 1997.

[10] D. Freitag. Information extraction from HTML: Application of a general learning approach. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, WI, 1998. AAAI Press.

[11] D. Freitag. Toward general-purpose learning for information extraction. In *Proceedings of COLING/ACL-98*, 1998.

[12] J.-T. Kim and D. I. Moldovan. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Transactions on Knowledge and Data Engineering*, 1995.

[13] N. Kushmerick. *Wrapper Induction for Information Extraction*. PhD thesis, University of Washington, 1997. Tech Report UW-CSE-97-11-04.

[14] M.Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Knigam, and S. Slattery. Learning to extract symbolic knowledge from the world wide web. Technical report, Department of Computer Science, Carnegie Mellon University, 1998.

[15] T. Mitchell. *Machine Learning*. The McGraw-Hill Companies, Inc., 1997.

[16] M. Perkowitz and O. Etzioni. Category translation: learning to understand information on the Internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 930–936. Morgan Kaufmann, 1995.

[17] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–2666, 1990.

[18] J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *Proceedings of the European Conference on Machine Learning*, pages 3–20, Vienna, Austria, 1993.

[19] E. Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1044–1049. AAAI/MIT Press, 1996.

[20] E. Riloff. An empirical study of automated dictionary construction for information extraction in three domains. *Arificial Intelligence*, 85:101–134, 1996.

[21] Shakes, J. Langheinrich, M., and O. Etzioni. Dynamic reference sifting: a case study in the homepage domain. In *Proceedings of Sixth International World Wide Web Conference*, Santa Clara, CA, 1996.

[22] S. Slattery and M. Craven. Combining statistical and relational methods for learning in hypertext domains. In *Proceedings of the 8th International Conference on Inductive Logic Programming*. Springer-Verlag, 1998.

[23] S. Soderland. *Learning Text Analysis Rules for Domain-specific Natural Language Processing*. PhD thesis, University of Massachusetts, 1996. Available as Department of Computer Science Technical Report 96-087.

[24] S. Soderland. Learning to extract text-based information from the World Wide Web. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, 1997.