Recognizing Structure in Web Pages using Similarity Queries

William W. Cohen
AT&T Labs–Research Shannon Laboratory
180 Park Avenue Florham Park, NJ 07932
wcohen@research.att.com

Abstract

We present general-purpose methods for recognizing certain types of structure in HTML documents. The methods are implemented using WHIRL, a "soft" logic that incorporates a notion of textual similarity developed in the information retrieval community. In an experimental evaluation on 82 Web pages, the structure ranked first by our method is "meaningful"—*i.e.*, a structure that was used in a hand-coded "wrapper", or extraction program, for the page—nearly 70% of the time. This improves on a value of 50% obtained by an earlier method. With appropriate background information, the structure-recognition methods we describe can also be used to learn a wrapper from examples, or for maintaining a wrapper as a Web page changes format. In these settings, the top-ranked structure is meaningful nearly 85% of the time.

Introduction

Web-based information integration systems allow a user to query structured information that has been extracted from the Web (Levy, Rajaraman, & Ordille 1996; Garcia-Molina *et al.* 1995; Knoblock *et al.* 1998; Genesereth, Keller, & Dushka 1997; Lacroix, Sahuguet, & Chandrasekar 1998; Mecca *et al.* 1998; Tomasic *et al.* 1997). In most such systems, a different *wrapper* must be written for each Web site that is accessed. A *wrapper* is a special-purpose program that extracts information from Web pages written in a specific format. Because data can be presented in many different formats, and because Web pages frequently change, building and maintaining wrappers is time-consuming and tedious. To reduce the cost of building wrappers, some researchers have proposed special languages for writing wrappers (Hammer *et al.* 1997; Cohen 1998b), or semi-automated tools for wrapper construction (Ashish & Knoblock 1997). Others have implemented systems that allow wrappers to be trained from examples (Kushmerick, Weld, & Doorenbos 1997; Hsu 1998; Muslea, Minton, & Knoblock 1998). Data exchange standards like XML have also been proposed, although as yet none are in widespread use.

Here, we explore another approach to this problem: developing general-purpose methods for automatically recognizing structure in HTML documents. Our ultimate goal is to extract structured information from Web pages without any page-specific programming or training.

Exploding porpoises, over four score and seven, well before configuration.

- *Department of Computer and Information Sciences, University of New Jersey.* Citrus flavorings: green, marine, clean and under lien.
- *Computer Engineering Center, Lough Polytechnical Institute*. This, that page extensionally left to rights of manatees.
- *Electrical Engineering and Computer Science Dept, Bismark State College*. Tertiary; where cola substitutes are frequently underutilized.

This page under construction. (Last update: 9/23/98.)

Figure 1: Nonsense text with a meaningful structure.

To motivate this approach, consider Figure 1. To a human reader, this text is perceived as containing a list of three items, each containing the italized name of a university department, with the university name underlined. This apparently meaningful structure is recognized without previous knowledge or training, even though the text is ungrammatical nonsense and the university names are imaginary. This suggests that people employ general-purpose, page-independent strategies for recognizing structure in documents. Incorporating similar strategies into a system that automatically (or semi-automatically) constructs wrappers would clearly be valuable.

Below we show that effective structure recognition methods for certain restricted types of list structures can be encoded compactly and naturally, given appropriate tools. In particular, we will present several methods that can be concisely implemented in WHIRL (Cohen 1998a), a "soft" logic that includes both "soft" universal quantification, and a notion of textual similarity developed in the information retrieval (IR) community. The structure-recognition methods we present are based on natural heuristics, such as detecting repetition of sequences of markup commands, and detecting repeated patterns of "familiar-looking" strings.

The methods can be used in a page-independent manner: given an HTML page, but no additional information about it, the methods produce a ranked list of pro-

HTML source for a simple list:

```
<html><head>...</head>
<body>
<h1>Editorial Board Members</h1>
<table> <tr>
    <td>G. R. Emlin, Lucent</td>
    <td>Harry Q. Bovik, Cranberry U</td></tr>
  <tr>
    <td>Bat Gangley, UC/Bovine</td>
    <td>Pheobe L. Mind, Lough Tech</td>
...
```

Extracted data:

G. R. Emlin, Lucent
Harry Q. Bovik, Cranberry U
...

HTML source for a simple hotlist:

```
<html><head>...</head>
<body><h1>Publications for Pheobe Mind</h1>
<ul>
<li>Optimization of fuzzy neural networks using
distributed parallel case-based genetic knowledge discovery
  (<a href="buzz.pdf">PDF</a>)</li>
<li>A linear-time version of GSAT
  (<a href="peqnp.ps">postscript</a>)</li>
...
```

Extracted data:

Optimization ... (PDF)	buzz.pdf
A linear-time version of ...	peqnp.ps
...	...

Figure 2: A simple list, a simple hotlist, and the data that would be extracted from each.

posed "structures" found in the page. This ranking is generally quite useful: in an experimental evaluation on 82 Web pages associated with real extraction problems, the top-ranked structure is "meaningful" (as defined below) nearly 70% of the time. This improves on an earlier method (Cohen & Fan 1999), which proposes meaningful structures about 50% of the time on the same data.

By providing different types of additional information, about a page, the same methods can also be used for page-specific wrapper learning as proposed by Kushmeric *et al* (1997), or for updating a wrapper after the format of a wrapped page has changed. When used for page-specific learning or wrapper update, the top-ranked structure is meaningful nearly 85% of the time.

Background

Benchmark problems

We begin by clarifying the structure-recognition problem, with the aim of stating a task precise enough to allow quantitative evaluation of performance. Deferring for now the question of what a "structure" is, we propose to rate the "structures" identified by our methods as either *meaningful* or *not meaningful*. Ideally, a structure in a Web page would be rated as *meaningful* iff it contains structured information that could plausibly be extracted from the page. Concretely, in our experiments, we will use pages that were actually wrapped by an information integration system, and consider a structure as meaningful iff it corresponds to information actually extracted by an existing, hand-coded wrapper for that page.

In this paper, we will restrict ourselves to wrappers in two narrow classes (and therefore, to a narrow class of potential structures). We call these wrapper classes *simple lists* and *simple hotlists*. In a page containing a *simple list*, the information extracted is a one-column relation containing a set of strings $s_1, \ldots, s_N$, and each s_i is all the text that falls below some node n_i in the HTML parse tree for the page. In a *simple hotlist*, the extracted information is a two-column relation, containing a set of pairs $\langle s_1, u_1 \rangle, \ldots, \langle s_n, u_N \rangle$; each s_i is all the text that falls below some node n_i in the HTML parse tree; and each u_i is a URL that is associated with some HTML anchor element a_i that appears somewhere inside n_i. Figure 2 shows the HTML source for a simple list and a simple hotlist, and the data that is extracted from each.

This restriction is based on our experience with a working information integration system (Cohen 1998b). Of 111 different wrapper programs written for this system, 82 (or nearly 75%) were based on simple lists or simple hotlists, as defined above.[1] We will use this corpus of problems in the experiments described below.

The vector space representation for text

Our ability to perceive structure in the text of Figure 1 is arguably enhanced by the regular appearance of substrings that are recognizable as (fictitious) university names. These strings are recognizable because they "look like" the names of real universities. Implementing such heuristics requires a precise notion of similarity for text, and one such notion is provided by the *vector space* model of text.

In the vector space model, a piece of text is represented as a *document vector* (Salton 1989). We assume a vocabulary T of *terms*; in this paper, terms are word stems produced by the Porter stemming algorithm (Porter 1980). A *document vector* is a vector of real numbers $\vec{v} \in \mathcal{R}^{|T|}$, each component of which corresponds to a term $t \in T$. We will denote the component of $\vec{v}$ which corresponds to $t \in T$ by v_t, and employ the TF-IDF weighting scheme (Salton 1989): for a document vector $\vec{v}$ appearing in a collection C, we let v_t be zero if the term t does not occur in text represented by $\vec{v}$, and otherwise let $v_t = (\log(TF_{\vec{v},t}) + 1) \cdot \log(IDF_t)$. In this formula, $TF_{\vec{v},t}$ is the number of times that

[1] We say "based on" because some lists also included preprocessing or filtering steps. We note also that the relative simplicity of wrappers is due in part to special properties of the information integration system. Further discussion of this dataset can be found elsewhere (Cohen & Fan 1999).

term t occurs in the document represented by $\vec{v}$, and $IDF_t = \frac{|C|}{|C_t|}$, where C_t is the set of documents in C that contain t.

In the vector space model, the *similarity* of two document vectors $\vec{v}$ and $\vec{w}$ is given by the formula $SIM(\vec{v},\vec{w}) = \sum_{t \in T} \frac{v_t \cdot w_t}{||\vec{v}|| \cdot ||\vec{w}||}$. Notice that $SIM(\vec{v},\vec{w})$ is always between zero and one, and that similarity is large only when the two vectors share many "important" (highly weighted) terms.

The WHIRL logic

Overview. WHIRL is a logic in which the fundamental items that are manipulated are not atomic values, but entities that correspond to fragments of text. Each fragment is represented internally as a document vector, as defined above; this means the similarity between any two items can be computed. In brief, WHIRL is non-recursive, function-free Prolog, with the addition of a built-in similarity predicate; rather than being true or false, a similarity literal is associated with a real-valued "score" between 0 and 1; and scores are combined as if they were independent probabilities.

As an example of a WHIRL query, let us suppose that the information extracted from the simple list of Figure 2 is stored as a predicate *ed_board(X)*. Suppose also that the information extracted from the hotlist of Figure 2, together with a number of similar bibliography hotlists, has been stored in a predicate *paper(Y,Z,U)*, where Y is an author name, Z a paper title, and U a paper URL. For instance, the following facts may have been extracted and stored: *ed_board("Pheobe L. Mind, Lough Tech")*, and *paper("Pheobe Mind", "A linear-time version of GSAT", "http://.../peqnp.ps")*. Using WHIRL's similarity predicate "$\sim$", the following query might be used to find papers written by editorial board members:

$\leftarrow$ *ed_board(X)* $\wedge$ *paper(Y,Z,U)* $\wedge$ $X \sim Y$

The answer to this query would be a list of substitutions θ, each with an associated score. Substitutions that bind X and Y to similar documents would be scored higher. One high-scoring substitution might bind X to "Pheobe L. Mind, Lough Tech" and Y to "Pheobe Mind".

Below we will give a formal summary of WHIRL. A complete description is given elsewhere (Cohen 1998a).

WHIRL semantics. Like a conventional deductive database (DDB) program, a WHIRL program consists of two parts: an *extensional database* (EDB), and an *intensional database* (IDB). The IDB is a non-recursive set of function-free definite clauses. The EDB is a collection of ground atomic facts, each associated with a numeric *score* in the range $(0,1]$. In addition to the types of literals normally allowed in a DDB, clauses in the IDB can also contain *similarity literals* of the form $X \sim Y$, where X and Y are variables. A WHIRL predicate definition is called a *view*. We will assume below views are *flat*—that is, that each clause body in the view contains only literals associated with predicates defined in the EDB. Since WHIRL does not support recursion, views that are not flat can be "flattened" (unfolded) by repeated resolution.

In a conventional DDB, the answer to a conjunctive query would be the set of ground substitutions that make the query true. In WHIRL, the notion of provability will be replaced with a "soft" notion of *score*, which we will now define. Let θ be a ground substitution for B. If $B = p(X_1, \ldots, X_a)$ corresponds to a predicate defined in the EDB, then $SCORE(B,\theta) = s$ if $B\theta$ is a fact in the EDB with score s, and $SCORE(B,\theta) = 0$ otherwise. If B is a similarity literal $X \sim Y$, then $SCORE(B,\theta) = SIM(\vec{x},\vec{y})$, where $\vec{x} = X\theta$ and $\vec{y} = Y\theta$. If $B = B_1 \wedge \ldots \wedge B_k$ is a conjunction of literals, then $SCORE(B,\theta) = \prod_{i=1}^{k} SCORE(B_i,\theta)$. Finally, consider a WHIRL view, defined as a set of clauses of the form $A_i \leftarrow Body_i$. For a ground atom a that is an instance of one or more A_i's, we define the *support* of a, $SUPPORT(a)$, to be the set of all pairs $\langle \sigma, Body_i \rangle$ such that $A_i\sigma = a$, $Body_i\sigma$ is ground, and $SCORE(Body_i,\sigma) > 0$. We define the score of an atom a (for this view) to be

$$1 - \prod_{\langle \sigma, Body_i \rangle \in SUPPORT(a)} (1 - SCORE(Body_i,\sigma))$$

This definition follows from the usual semantics of logic programs, together with the observation that if e_1 and e_2 are independent events, then $\mathrm{Prob}(e_1 \vee e_2) = 1 - (1 - \mathrm{Prob}(e_1))(1 - \mathrm{Prob}(e_2))$.

The operations most commonly performed in WHIRL are to define and *materialize* views. To materialize a view, WHIRL finds a set of ground atoms a with non-zero score s_a for that view, and adds them to the EDB. Since in most cases, only high-scoring answers will be of interest, the materialization operator takes two parameters: r, an upper bound on the number of answers that are generated, and ϵ, a lower bound on the score of answers that are generated.

Although the procedure used for combining scores in WHIRL is naive, inference in WHIRL can be implemented quite efficiently. This is particularly true if ϵ is large or r is small, and if certain approximations are allowed (Cohen 1998a).

The "many" construct. The structure-recognition methods we will present require a recent extension to the WHIRL logic: a "soft" version of universal quantification. This operator is written *many(Template,Test)* where the *Test* is an ordinary conjunction of literals, and the *Template* is a single literal of the form $p(Y_1, \ldots, Y_n)$, where p is an EDB predicate and the Y_i's are all distinct; also, the Y_i's may appear only in *Test*. The score of a "many" clause is the weighted average score of the *Test* conjunction on items that match the *Template*. More formally, for a substitution θ and a conjunction W,

$$SCORE(many(p(Y_1, \ldots, Y_k), Test), \theta) =$$

$$\sum_{\langle s, a_1, \ldots, a_k \rangle \in P} \frac{s}{S} \cdot SCORE(Test, (\theta \circ \{Y_i = a_i\}_i))$$

where P is the set of all tuples $\langle s, a_1, \ldots, a_k \rangle$ such that $p(a_1, \ldots, a_k)$ is a fact in the EDB with score s; S is the sum of all such scores s; and $\{Y_i = a_i\}_i$ denotes the substitution $\{Y_1 = a_1, \ldots, Y_k = a_k\}$.

As an example, the following WHIRL query is a request for editorial board members that have written "many" papers on neural networks.

q(X) ← ed_board(X) ∧
many(papers(Y,Z,W),
(X∼Y ∧ Z∼ "neural networks")).

Recognizing structure with WHIRL

Encoding HTML pages and wrappers

We will now give a detailed description of how structure-recognition methods can be encoded in WHIRL. We begin with a description of the encoding used for an HTML page.

To encode an HTML page in WHIRL, the page is first parsed. The HTML parse tree is then represented with the following EDB predicates.

- *elt(Id,Tag,Text,Position)* is true if *Id* is the identifier for a parse tree node, n, *Tag* is the HTML tag associated with n, *Text* is all of the text appearing in the subtree rooted at n, and *Position* is the sequence of tags encountered in traversing the path from the root to n. The value of *Position* is encoded as a a document containing a single term t_{pos}, which represents the sequence, *e.g.*, t_{pos}= *"html_body_ul_li"*.
- *attr(Id,AName,AValue)* is true if *Id* is the identifier for node n, *AName* is the name of an HTML attribute associated with n, and *AValue* is the value of that attribute.
- *path(FromId,ToId,Tags)* is true if *Tags* is the sequence of HTML tags encountered on the path between nodes *FromId* and *ToId*. This path includes both endpoints, and is defined if *FromId=ToId*.

As an example, wrappers for the pages in Figure 2 can be written using these predicates as follows.

page1(NameAffil) ←
elt(_, _, NameAffil, "html_body_table_tr_td").
page2(Title,Url) ←
elt(ContextElt, _, Title, "html_body_ul_li")
∧ path(ContextElt, AnchorElt, "li_a")
∧ attr(AnchorElt, "href", Url).

Next, we need to introduce an appropriate encoding of "structures" (and in so doing, make this notion precise.) Most simple lists and hotlists in our benchmark collection can be wrapped with some variant of either the *page1* or *page2* view, in which the constant strings (*e.g.*, "*html_body_ul_li*" and "*li_a*") are replaced with different values. Many of the remaining pages can be wrapped by views consisting of a disjunction of such clauses.

We thus introduce a new construct to formally represent the informal idea of a "structure" in a structured document: a *wrapper piece*. In the most general setting, a *wrapper piece* consists of a *clause template* (*e.g.*, a generic version of *page2* above), and a set of *template parameters* (*e.g.*, the pair of constants "*html_body_ul_li*" and "*li_a*"). In the experiments below, we consider only two clause templates—the ones suggested by the examples above—and also assume that the recognizer knows, for each page, if it should look for list structures or hotlist structures. In this case, the clause template need not be explicitly represented; a wrapper piece for a *page2* variant can be represented simply as a pair of constants (*e.g.*, "*html_body_ul_li*" and "*li_a*"), and a wrapper piece for a *page1* variant can be represented as a single constant (*e.g.*, *html_body_table_tr_td*).

For brevity, we will confine the discussion below to methods that recognize simple hotlist structures analogous to *page2*, and will assume that structures are encoded by a pair of constants *Path1* and *Path2*. However, most of the methods we will present have direct analogs that recognize simple lists.

Enumerating and ranking wrappers

We will now describe three structure-recognition methods based on these encodings. We begin with some basic building blocks. Assuming that some page of interest has been encoded in WHIRL's EDB, materializing the WHIRL view *possible_piece*, shown in Figure 3, will generate all wrapper pieces that would extract at least one item from the page. The *extracted_by* view determines which items are extracted by each wrapper piece, and hence acts as an interpreter for wrapper pieces.

Using these views in conjunction with WHIRL's soft universal quantification, one can compactly state a number of plausible recognition heuristics. One heuristic is to prefer wrapper pieces that extract many items; this trivial but useful heuristic is encoded in the *fruitful_piece* view. Recall that materializing a WHIRL view results in a set of new atoms, each with an associated score. The *fruitful_piece* view can thus be used to generate a ranked list of proposed "structures" by simply presenting all *fruitful_piece* facts to the user in decreasing order by score.

Another structure-recognition method is suggested by the observation that in most hotlists, the text associated with the anchor is a good description of the associated object. This suggests the *anchorlike_piece* view, which adds to the *fruitful_piece* view an additional "soft" requirement that the text *Text1* extracted by the wrapper piece be similar to the text *Text2* associated with the anchor element.

A final structure-recognition method is shown in the Figure as the *R_like_piece* view. This view is a copy of *fruitful_piece* in which the requirement that many items are extracted is replaced by a requirement that many "R like" items are extracted, where an item is "R like" if it is similar to some second item X that is stored in the EDB relation R. The "soft" semantics of the *many*

```
fruitful_piece(Path1,Path2) ←
   possible_piece(Path1,Path2) ∧
   many( extracted_by(Path1a,Path2a,_,_),
         (Path1a=Path1 ∧ Path2a=Path2) ).
possible_piece(Path1,Path2) ←
   elt(TextElt, _, _, Path1)
   ∧ elt(AnchorElt, _, "a", _)
   ∧ attr(AnchorElt, "href", _)
   ∧ path(TextElt, AnchorElt, Path2).
extracted_by(Path1,Path2,TextElt,AnchorElt) ←
   elt(TextElt, _, _, Path1)
   ∧ path(TextElt, AnchorElt, Path2).
```

```
anchorlike_piece(Path1,Path2) ←
   possible_piece(Path1,Path2) ∧
   many( extracted_by(Path1a,Path2a,TElt,AElt),
      (Path1a=Path1 ∧ Path2a=Path2
       ∧ elt(TElt,_,Text1,_) ∧ elt(AElt,_,Text2,_) ∧ Text1~Text2 ).

R_like_piece(Path1,Path2) ←
   possible_piece(Path1,Path2) ∧
   many( R_extracted_by(Path1a,Path2a,_,_),
         (Path1a=Path1 ∧ Path2a=Path2) ).
R_extracted_by(Path1,Path2,TextElt,AnchorElt) ←
   elt(TextElt, _, Text, Path1)
   ∧ path(TextElt, AnchorElt, Path2)
   ∧ R(X) ∧ Text~X.
```

Figure 3: WHIRL programs for recognizing plausible structures in an HTML page. (See text for explanation.)

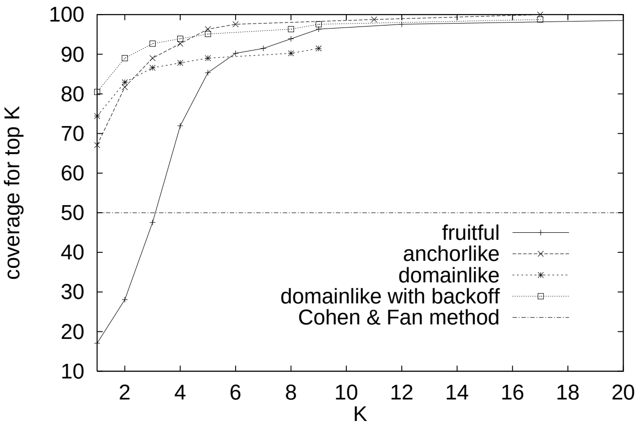

Figure 4: Performance of ranking heuristics that use little or no page-specific information.

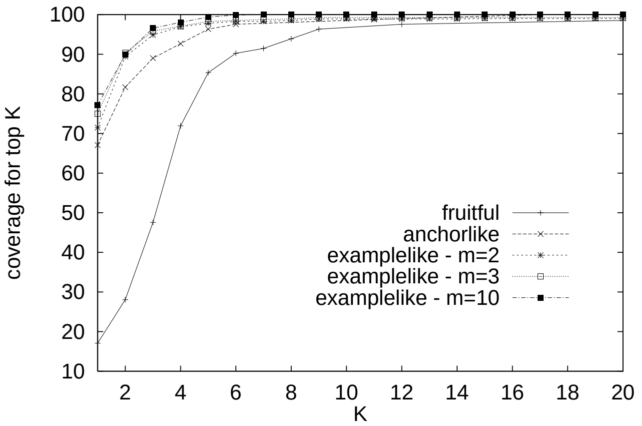

Figure 5: Performance of ranking heuristics that use page-specific training examples.

construct imply that more credit is given to extracting items that match an item in R closely, and less credit is given for weaker matches. As an example, suppose that R contains a list of all accredited universities in the US. In this case, the *R_like_piece* would prefer wrapper pieces that extract many items that are similar to some known university name; this might be useful in processing pages like the one shown in Figure 1.

Experiments

Ranking programs with (almost) no page-specific information

We will now evaluate the three structure-recognition methods shown in Figure 3. We took the set of 82 hand-coded list and hotlist wrappers described above, and paired each hand-coded wrapper with a single Web page that was correctly wrapped. We then analyzed the hand-coded wrapper programs, and determined which wrapper pieces they contained. The result of this preprocessing was a list of 82 Web pages, each of which is associated with a set of "meaningful" wrapper pieces. To evaluate a method, we materialize the appropriate

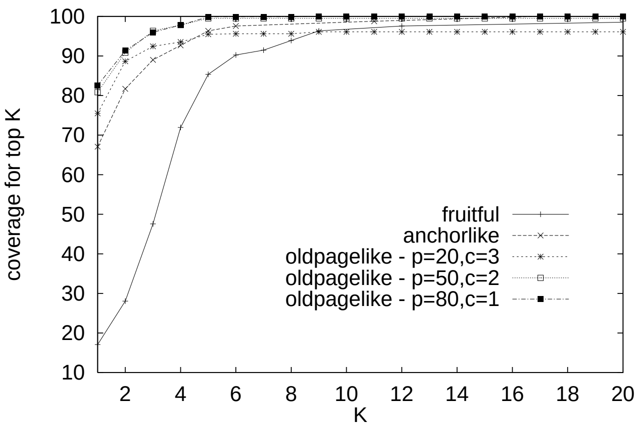

Figure 6: Performance of ranking heuristics that use text extracted from an previous version of the page.

view,[2] thus generating a ranked list of proposed struc-

[2]Thresholds of $\epsilon = 0.5$ and $r = 100,000$ were used in materializing the view *R_extracted_by*, and thresholds of $\epsilon = 0$ and $r = 100,000$ were used elsewhere. We also assume that the system "knows" whether a list or a hotlist is to be extracted from each page: *i.e.*, we apply to each hotlist page

tures. A good method is one that ranks meaningful structures ahead of non-meaningful structures.

To obtain useful aggregate measures of performance, it is useful to consider how a structure-recognition method might be used. One possibility is *interactive use:* given a page to wrap, the method proposes wrapper pieces to a human user, who then examines them in order, and manually selects pieces to include in a wrapper for the page. To evaluate performance, we vary K and record for each K the *coverage* at rank K—that is, the percentage of the 82 problems that can be wrapped using pieces ranked in the top K. The resulting "coverage curve" can be plotted. We also compute a measure we call the *average number of skips*: the average number of non-meaningful pieces that are ranked ahead of some meaningful piece; or, equivalently, the average number of wrapper pieces that would be unnecessarily examined (skipped over) by the user.

Another possibility use for the system is *batch use:* given a page, the method proposes a single structure, which is then used by a calling program without any filtering. For batch use, a natural measure of performance is the percentage of the time that the top-ranked structure is meaningful. Below, we call this measure *accuracy at rank 1*, and define *error rate at rank 1* analogously.[3]

Figure 4 shows the coverage curves obtained from methods that require no page-specific information. For comparison, we also show the performance of a earlier structure-recognition method (Cohen & Fan 1999). (To summarize this method briefly, structure recognition is reduced to the problem of classifying nodes in an HTML parse tree as to whether or not they are contained in some meaningful structure. The node-classification problem can then be solved by off-the-shelf inductive learning methods such as CART (Brieman *et al.* 1984) or RIPPER (Cohen 1995).) This method produces a single wrapper program (which may correspond to multiple wrapper pieces), rather than a ranked list of wrapper pieces. On the data used here, the wrapper proposed coincides with the true wrapper, or some close approximation of it, on exactly half the cases.

The *anchorlike* method[4] performs quite well, obtaining accuracy at rank 1 of nearly 70%, and an average of 0.9 skips. (These numbers are summarized in Table 1). Even the strawman *fruitful* method works surprisingly well in interactive use, obtaining an average number of skips of only 3.3; however, for batch use, its accuracy at rank 1 is less than 20%.

The third curve shown in Figure 4, labeled *domainlike*, is an instance of the *R_like_piece* method in which R contains a large list of items in the same domain as the items to be extracted. (For instance, if the data to be extracted is a list of universities, then R would be a second list of universities.) We consider this structure-recognition method in this section because, although it does require some page-specific information, the information required is quite easy to obtain.[5] The average skip rate and error at rank 1 for the *domainlike* method are roughly half that of *anchorlike*. However, this method does not obtain 100% coverage, as in some fraction of the problems, the secondary relation R is either unavailable or misleading.

The final curve in Figure 4, labeled "domainlike with backoff", is a simple combination of the *domainlike* and *anchorlike* strategies. In this method, one first materializes the view *R_extracted_by*. If it is non-empty, then the *R_like_piece* view is materialized, and otherwise, *anchorlike_piece* is materialized. This method does as well in a batch setting as *domainlike*. In an interactive setting, it achieves a final coverage of nearly 100% with a skip rate somewhat lower than *anchorlike*.

Ranking structures with training data

Several previous researchers have considered the problem of learning wrappers from examples (Kushmerick, Weld, & Doorenbos 1997; Hsu 1998; Muslea, Minton, & Knoblock 1998). In these systems, the user provides examples of the items that should be extracted from a sample Web page, and the system induces a general procedure for extracting data from that page. If page-specific training examples are available, they can be used by storing them in a relation R, and then applying the *R_like* method. This use of structure-recognition methods is quite similar to previous wrapper-learning systems; one major difference, however is that no negative examples need be provided, either explicitly or implicitly.

To evaluate this wrapper-learning technique, we ran the target wrappers on each page in order to build a list of page-specific training examples. We then fixed a number of training examples m, and for each Web page, stored m randomly chosen page-specific examples in the relation R, and applied the *R_like* structure-recognition method. We call this the *examplelike* method. This process was repeated 10 times for each value of m, and the results were averaged.

The results from this experiment are shown in Figure 5. Even two or three labeled examples perform

only structure-recognition views that recognize hotlists, and apply to each list page only views that recognize lists.

[3]Note that accuracy at rank 1 is not identical to coverage at $K = 1$; the former records the number of times the top-ranked wrapper piece is part of the target wrapper, and the later records number of times the top-ranked wrapper piece is the *only* piece in the target wrapper.

[4]Recall that the *anchorlike* can only be applied to hotlists. In the curve labeled *anchorlike*, we used the *fruitful* method for simple list wrappers, and the *anchorlike* method for simple hotlist wrappers.

[5]It seems reasonable to assume that the user (or calling program) has general knowledge about the type of the items that will be extracted. In the experiments, the items in R were always obtained from a second Web page containing items of the same type as the page being wrapped; again, it seems plausible to assume that this sort of information will be available.

	Average # Skips	Accuracy at Rank 1	Coverage at $K = \infty$
fruitful	3.3	18.3	100.0
anchorlike	0.9	69.5	100.0
domainlike	0.4	84.0	91.5
with backoff	0.6	84.0	98.8
examplelike			
$m = 2$	0.3	77.8	99.0
$m = 3$	0.3	79.3	99.3
$m = 10$	0.3	84.2	100.0
oldpagelike			
$p = 20, c = 3$	0.3	85.0	96.1
$p = 50, c = 2$	0.3	82.9	99.5
$p = 80, c = 1$	0.3	85.4	100.0

Table 1: Summary of results

somewhat better than the *anchorlike* and *fruitful* methods, and unlike the *domainlike* method, achieve complete (or nearly complete) coverage. However, the average accuracy at rank 1 is not as high as for the *domainlike* method, unless many examples are used.

These results show an advantage to presenting the user with a ranked list of wrapper pieces, as in general, coverage is improved much more by increasing K than by increasing m. For example, if the user labels two examples, then 58.6% of the pages are wrapped correctly using the top-ranked wrapper piece alone. Providing eight more examples increases coverage of the top-wranked piece to only 63.3%; however, if the user labels no additional examples, but instead considers the top *two* wrapper pieces, coverage jumps to 89.4%.

Maintaining a wrapper

Because Web pages frequently change, maintaining existing wrappers is a time-consuming process. In this section, we consider the problem of updating an existing wrapper for a Web page that has changed. Here a new source of information is potentially available: one could retain, for each wrapper, the data that was extracted from the previous version of the page. If the format of the page has been changed, but not its content, then the previously-extracted data can be used as page-specific training examples for the new page format, and the *examplelike* method of the previous section can be used to derive a new wrapper. If the format and content both change, then the data extracted from the old version of the page could still be used; however, it would be only an approximation to the examples that a user would provide. Using such "approximate examples" will presumably make structure-recognition more difficult; on the other hand, there will typically be many more examples than a user would provide.

Motivated by these observations, we evaluated the *R_like* structure-recognition method when R contains a large number of entries, each of which is a *corrupted* version of a data item that should be extracted from the page. Specifically, we began with a list of all data items that are extracted by the target wrapper, and

Oklahoma	dietitians
Yukon	Yukon codpiece
Vermont	Vermont
British Columbia	British Columbia Talmudizations
Oklahoma	Oklahoma
Wisconsin	Wisconsin
New Jersey	New Jersey incorrigible blubber
Alaska	Alaska
New Brunswick	
New Mexico	New Mexico cryptogram

Table 2: Ten US States and Canadian Provinces, before and after corruption with $c = 1$.

then corrupted this list as follows. First, we discarded all but randomly-chosen percentage p of the items.[6] We next perform $c \cdot n$ random edit operations, where n is the number of retained examples. Each edit operation randomly selects one of the n items, and then either deletes a randomly chosen word from the item; or else adds a word chosen uniformly at random from */usr/dict/words*.

Figure 6 shows the results of performing this experiment (again averaged over 10 runs) with values of p ranging from 80% to 20%, and values of c ranging from 1 to 3. We call this structure-recognition method the *oldpagelike* method. With moderately corrupted example sets, the method performs very well: even with a corruption level of $p = 50\%$ and $c = 2$ it performs better on average than the *anchorlike* method.

It must be noted, however, that the corrupted examples used in this experiment are not very representive of the way a real Web page would be changed. As an illustration, Table 2 shows one list of familiar items before and after corruption with $c = 1$ (really!). It remains to be seen if more typical modifications are harder or easier to recover from.

Conclusions

In this paper, we considered the problem of recognizing "structure" in HTML pages. As formulated here, structure recognition is closely related to the task of automatically constructing wrappers: in our experiments, a "structure" is equated with a component of a wrapper, and a recognized structure is considered "meaningful" if it is part of an existing wrapper for that page. We used WHIRL, a "soft" logic that incorporates a notion of textual similarity developed in the information retrieval community, to implement several heuristic methods for recognizing structures from a narrow but useful class. Implementing these methods also required an extension to WHIRL—a "soft" version of bounded universal quantification.

Experimentally, we showed that one proposed structure-recognition method, the *anchorlike* method, performs quite well: the top-ranked structure is meaningful about 70% of the time, substantially improving

[6] Note that typically, Web pages change by having new items added, and we are trying to simulate text that would have been extracted from an old version of the page.

on simpler ranking schemes for structures, and also improving on an earlier result of ours which used more conventional methods for recognizing structure. This method is completely general, and requires no page-specific information. A second structure-recognition method, the *R_like* method, was also described, which can make use of information of many different kinds: examples of correctly-extracted text; an out-of-date version of the wrapper, together with a cached version of the last Web page that this out-of-date version correctly wrapped; or a list of objects of the same type as those that will be extracted from the Web page. In each of these cases, performance can be improved beyond that obtained by the *anchorlike* method.

References

Ashish, N., and Knoblock, C. 1997. Wrapper generation for semistructured Internet sources. In Suciu, D., ed., *Proceedings of the Workshop on Management of Semistructured Data*. Tucson, Arizona: Available on-line from http://www.research.att.com/~suciu/workshop-papers.html.

Brieman, L.; Friedman, J. H.; Olshen, R.; and Stone, C. J. 1984. *Classification and Regression Trees*. Belmon, CA: Wadsworth.

Cohen, W. W., and Fan, W. 1999. Learning page-independent heuristics for extracting data from web pages. In *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Agents in Cyberspace*.

Cohen, W. W. 1995. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*. Lake Tahoe, California: Morgan Kaufmann.

Cohen, W. W. 1998a. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proceedings of ACM SIGMOD-98*.

Cohen, W. W. 1998b. A Web-based information system that reasons with structured collections of text. In *Proceedings of Autonomous Agents-98*.

Garcia-Molina, H.; Papakonstantinou, Y.; Quass, D.; Rajaraman, A.; Sagiv, Y.; Ullman, J.; and Widom, J. 1995. The TSIMMIS approach to mediation: Data models and languages (extended abstract). In *Next Generation Information Technologies and Systems (NGITS-95)*.

Genesereth, M.; Keller, A.; and Dushka, O. 1997. Infomaster: an information integration system. In *Proceedings of the 1997 ACM SIGMOD*.

Hammer, J.; Garcia-Molina, H.; Cho, J.; and Crespo, A. 1997. Extracting semistructured information from the Web. In Suciu, D., ed., *Proceedings of the Workshop on Management of Semistructured Data*. Tucson, Arizona: Available on-line from http://www.research.att.com/~suciu/workshop-papers.html.

Hsu, C.-N. 1998. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In *Papers from the 1998 Workshop on AI and Information Integration*. Madison, WI: AAAI Press.

Knoblock, C. A.; Minton, S.; Ambite, J. L.; Ashish, N.; Modi, P. J.; Muslea, I.; Philpot, A. G.; and Tejada, S. 1998. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*.

Kushmerick, N.; Weld, D. S.; and Doorenbos, R. 1997. Wrapper induction for information extraction. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*.

Lacroix, Z.; Sahuguet, A.; and Chandrasekar, R. 1998. User-oriented smart-cache for the web: what you seek is what you get. In *Proceedings of the 1998 ACM SIGMOD*.

Levy, A. Y.; Rajaraman, A.; and Ordille, J. J. 1996. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB-96)*.

Mecca, G.; Atzeni, P.; Masci, A.; Merialdo, P.; and Sindoni, G. 1998. The ARANEUS web-base management system. In *Proceedings of the 1998 ACM SIGMOD*.

Muslea, I.; Minton, S.; and Knoblock, C. 1998. Wrapper induction for semistructured, web-based information sources. In *Proceedings of the Conference on Automated Learning and Discovery (CONALD)*.

Porter, M. F. 1980. An algorithm for suffix stripping. *Program* 14(3):130–137.

Salton, G., ed. 1989. *Automatic Text Processing*. Reading, Massachusetts: Addison Welsley.

Tomasic, A.; Amouroux, R.; Bonnet, P.; and Kapitskaia, O. 1997. The distributed information search component (Disco) and the World Wide Web. In *Proceedings of the 1997 ACM SIGMOD*.