Record Location and Reconfiguration in Unstructured Multiple-Record Web Documents

D.W. Embley and L. Xu
Department of Computer Science
Brigham Young University
Provo, Utah 84602, U.S.A.
{embley,lx,}@cs.byu.edu

ABSTRACT

Record extraction from data-rich, unstructured, multiple-record Web documents works well [8], but only if the text for each record can be located and isolated. Although some multiple-record Web documents present records as contiguous, delineated chunks of text (which can thus be located and isolated [9]), many do not. When some values of textual records are factored out, are split unnaturally across boundaries, are joined unnaturally within boundaries, or are linked by off-page connectors, or when desired records are interspersed with records that are not of interest, it is difficult to automatically cull records and piece values together to form clean, delineated chunks of text that each represent a single record of interest. In this paper we attack this problem and propose an algorithm to find and rearrange (if necessary) records in an HTML document by attempting to maximize a record-recognition heuristic with respect to a given application ontology. Tests we conducted show that this technique properly locates and reconfigures records for all classified types of rearrangements both for artificial and for actual multiple-record Web documents.

1. INTRODUCTION

The World Wide Web contains abundant repositories of information in Web documents. Many of these Web documents contain lists of unstructured text whose contents can be extracted to form database records. As an example, Figure 1 shows a list of car advertisements; each ad is an unstructured record that contains values such as the year of the car, its make and model, its features, its selling price, and a contact number. We call these Web documents *multiple-record Web documents*.

Over the past two years, we have experimented successfully with extracting record data from data-rich, unstructured, multiple-record Web documents. Applications have included car ads, job ads, obituaries, real estate, precious gems, computer monitors, games, musical instruments, stocks, and personals [8, 13]. As a measure of success, we have computed recall and precision ratios for each attribute for each application. We achieved recall ratios in the range of 90% and precision ratios near 98% for both car ads and job ads. For obituaries, a much more complex challenge, recall ratios ranged from 70% to 100%, and precision ratios ranged from 93% to 100% (except for names of relatives, which dropped to 71%). Our results compare favorably with the results others have obtained (e.g. [1, 2, 3, 4, 5, 7, 11, 14, 15, 16, 12, 19, 20]).[1]

In our experiments, however, we have assumed that the input is a set of clean, plain-text record chunks. Initially, we obtained these unstructured records by hand, but we have since developed an algorithm to discover record boundaries automatically and to clean and present unstructured records to our downstream data-extraction system [8]. This record-boundary-discovery algorithm works well (near 100%) for Web documents such as the one in Figure 1 in which each unstructured record exists as an isolated text chunk. Because it only finds record delimiters, however, it fails for Web documents that have characteristics like the ones in Figure 2, where the car ads are: (1) *factored*—the year values 1999 and 1998 appear only once for each group of car ads in Figure 2(a); (2) *joined*—the next to last ad in Figure 2(a) mentions three cars jointly; (note also that the dealer and phone number are factored to the end of the ad); (3) *off-page*—the second ad in Figure 2(a) contains an off-page link to the page in Figure 2(b); (4) *split*—the natural white-space boundary splits the third ad in Figure 2(b); and (5) *interspersed*—the second 1998-ad in Figure 2(a) is an ad for a washer and dryer, which is interspersed among the car ads.

The contribution of this paper is that it shows how to resolve each of these problems. The technique for each is similar. The main idea is to define a record-recognition heuristic that measures how well an unstructured record matches a predefined ontology description for the application of interest. We adapt the Vector Space Model (VSM) [18] from the field of information retrieval to define our record-recognition heuristic. The heuristic measure (1) increases when we "defactor" a set of records (e.g. distribute the years in Figure 2(a) to each of the records); (2) increases when we divide joined records that appear within a delimited record group and increases even more when we defactor end values in joined record groups; (3) increases when we pull in off-page text; (4) increases when we group records split across apparent boundaries; and (5) increases when we discard unsuitable interspersed records. As a result, when we rearrange a document to "maximize" the record-recognition heuristic for a multiple-record Web document, the result becomes a list of text chunks, one for each individual unstructured record.

Section 2 describes the application ontologies we use in our

[1]For a specific comparison between our work and the work cited here, see [8].

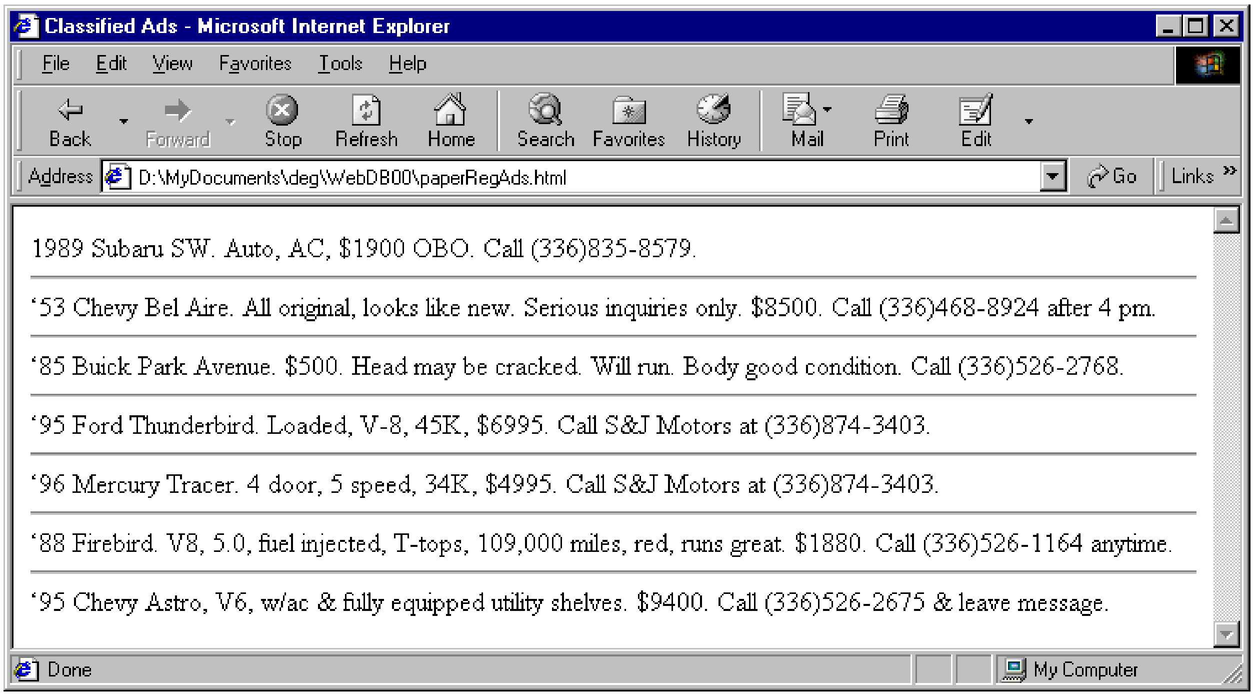
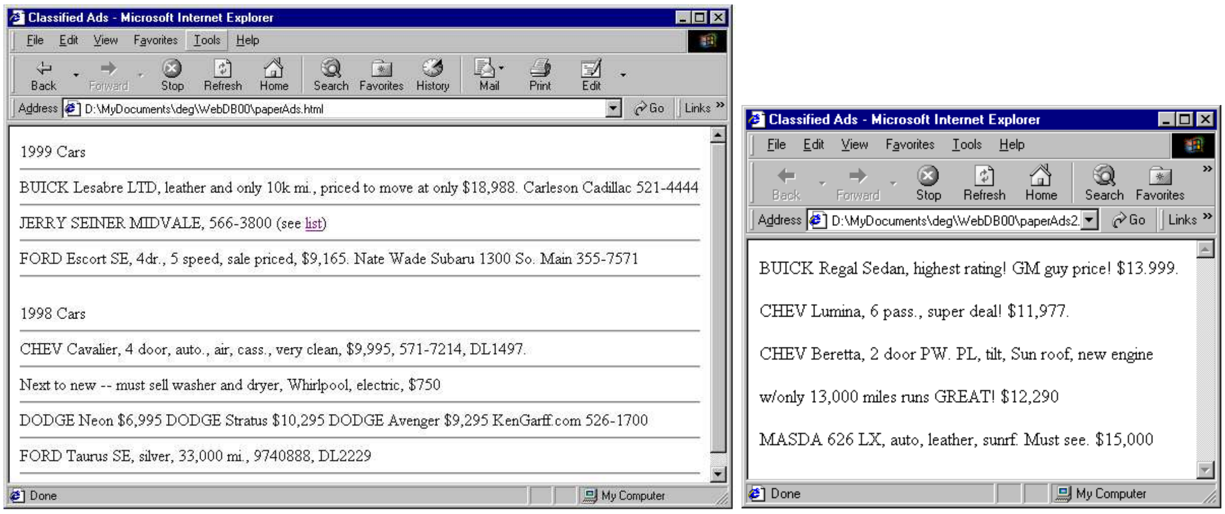


Figure 1: Regular Car Ads



(a) Base Car Ads

(b) Off-Page Car Ads

Figure 2: Irregular Car Ads

```
 1. Car [-> object];
 2. Car [0:0.975:1] has Year [1:*];
 3. Car [0:0.925:1] has Make [1:*];
 4. Car [0:0.908:1] has Model [1:*];
 5. Car [0:0.45:1] has Mileage [1:*];
 6. Car [0:2.1:*] has Feature [1:*];
 7. Car [0:0.8:1] has Price [1:*];
 8. PhoneNr [1:*] is for [1:1.15:*];
 9. Year matches [4]
10.       constant {extract "\d{2}";
11.             context "\b'[4-9]\d\b";
12.             substitute "^" -> "19"
13.             ...
14. Mileage matches [8]
15.       ...
16.       keyword "\bmiles\b", "\bmi\.", "\bmi\b",
17.             "\bmileage\b";
18. ...
```

Figure 3: Car-Ads Application Ontology (Partial)

work. Section 3 explains how we adapt VSM for use as a record-recognition heuristic. Section 4 presents an algorithm that uses our adapted record-recognition heuristic to rearrange a multiple-record HTML document and to produce automatically the ontology-applicable unstructured records contained within the document. Section 5 discuss the results of applying our heuristic algorithm to several Web documents. We make concluding remarks in Section 6.

2. APPLICATION ONTOLOGY

For our work in data extraction, we define an *application ontology* to be a conceptual-model instance that describes a real-world application in a narrow, data-rich domain of interest (e.g. car advertisements, obituaries, job advertisements) [8]. Each of our application ontologies consists of two components: (1) an *object/relationship-model instance* that describes sets of objects, sets of relationships among objects, and constraints over object and relationship sets, and (2) for each object set, a *data frame* that defines the potential contents of the object set. A data frame for an object set defines the lexical appearance of constant objects for the object set and establishes appropriate keywords that are likely to appear in a document when objects in the object set are mentioned. Figure 3 shows part of car-ads application ontology, including object and relationship sets and cardinality constraints (lines 1-8) and a few lines of the data frames (lines 9-18). (The full ontology for car ads is about 600 lines in length.)

An object set in an application ontology represents a set of objects which may either be lexical or nonlexical. Data frames with declarations for constants that can potentially populate the object set represent lexical object sets, and data frames without constant declarations represent nonlexical object sets. *Year* (Line 9) and *Mileage* (Line 14) are object sets of length 4 characters and 8 characters respectively; *Car*, *Make*, *Model*, *Price*, and *PhoneNr* are the remaining object sets in our car-ads application.

We describe the constant lexical objects and the keywords for an object set by regular expressions using Perl syntax. When applied to a textual document, the **extract** clause in a data frame causes a string matching a regular expression to be extracted, but only if the **context** clause also matches the string and its surrounding characters. A **substitute** clause lets us alter the extracted string before we store it in an intermediate file, in which we also store the string's position in the document and its associated object set name. One of the nonlexical object sets is designated as the *object set of interest*—*Car* for the car-ads ontology. The notation "[-> object]" in Line 1 designates the object set of interest.

We denote a relationship set by a name that includes its object set names (e.g. *Car has Year* and *PhoneNr is for Car*). The *min:max* pairs and *min:ave:max* triples in the relationship-set name are *participation constraints*: *min* designates the minimum number of times an object in the object set can participate in the relationship set; *ave* designates the average number of times an object is expected to participate in the relationship set; and *max* designates the maximum number of times an object can participate, with * designating an unknown maximum number of times. The participation constraint on *Car* for *Car has Feature*, for example, specifies that a car need not have any listed features, that a car has 2.1 features on the average, and that there is no specified maximum for the number of features listed for a car.

For our car-ads application ontology, we obtained participation constraints as follows. We selected 10 different regions covering the United States and found a car-ad page from each of these regions. From each of these pages we selected 12 individual car-ads by taking every $n/12$-th car-ad where n was the total number of car-ads on the page. We then counted by hand and obtained minimum, average, and maximum values for each object set in each relationship set.

3. A RECORD-RECOGNITION HEURISTIC

We are interested in recognizing the existence of chunks of unstructured text that constitute record information both for individual records and for groups of records. We have adapted the Vector Space Model (VSM) [18], a common information-retrieval measure of document relevance, for this purpose. VSM measures the cosine between two vectors—in our case, between an ontology vector OV representing what we expect to find and a document (or subdocument) vector DV representing we actually find. VSM also measures the magnitudes of the two vectors.

To construct the ontology vector OV, we (1) identify the lexical object-set names—these become the names of the coefficients of OV, and (2) determine the average participation for each lexical object set with respect to the object set of interest specified in O—these become the values of the coefficients of OV. The ontology vector for the car-ads application in Figure 3 is $<$ Year:0.975, Make:0.925, Model:0.908, Mileage:0.45, Price:0.8, Feature:2.1, PhoneNr:1.15 $>$.

We can construct a document vector DV for an entire document or for any subpart part of a document (even down to single lines of text and single sub-line phrases). The names of the coefficients of DV are the same as the names of the coefficients of OV. We obtain the value of each coefficient of DV by automatically counting the number of appearances of constant values that belong to each lexical object set. By applying the car-ads application ontology in Figure 3 to the entire document in Figure 1 we find 7 $Year$ values, 6

Make values, and so forth, and obtain the document vector $<$ Year:7, Make:6, Model:6, Mileage:3, Price:6, Feature:7, PhoneNr:7 $>$.

We have discussed the creation of a document vector as if correctly detecting and classifying the lexical values in a document is easy—but it is not easy. We identify potential lexical values for an object set as explained in Section 2; this can be error-prone, but we can adjust the regular expressions to improve this initial identification and achieve good results [8]. After initial identification, we must decide which of these potential object-set/constant pairs to accept. In our downstream processes, we use sophisticated heuristic based on keyword proximity, application-ontology cardinalities, record boundaries, and missing-value defaults to best match object sets with potential constants. For upstream rearrangement of records we use techniques that are far less sophisticated and thus also far less costly. In our simple upstream procedures we consider only, two cases: (1) a recognized string has no overlap either partially or completely with any other recognized string, and (2) a recognized string does overlap in some way with at least one other recognized string. For Case 1, we accept the recognized string for an object set even if the sophisticated downstream processes would reject it. For Case 2, we resolve the overlap simplistically, as follows. There are three subcases: (1) exact match, (2) subsumption, and (3) partial overlap. (1) If a lexical value v is recognized as potentially belonging to more than one lexical object set, we use the closest keyword that appears before or after v to determine which object set to choose; if no applicable keyword is found, we choose one of the object sets arbitrarily. (2) If a lexical value v is a proper substring of lexical value w, we retain w and discard v (3) If lexical value v and lexical value w appear in a Web document, such that a suffix of v is a prefix of w, we retain v and discard w.

The *VSM measure* defined in [18], calculates the acute angle between an ontology vector OV and a document (or subdocument) vector DV as $\cos\theta = P/N$, where P is the inner product of the two vectors and N is the product of the lengths of the two vectors. When OV is the ontology vector for the car-ads application ontology (given above) and DV is the document vector for the car-ads document in Figure 1 (given above), the VSM measure is 0.948. When the distribution of values among the object sets in DV closely matches the expected distribution specified in OV, the angle θ is close to zero degrees, and $\cos\theta$ is close to one.

As a by-product of computing the VSM measure, we obtain the lengths of the vectors OV and DV. Because the vector coefficients for OV are the estimates for one record and the vector coefficients for DV are the values identified for one or more records, the length $|DV|$ divided by the length $|OV|$ is a rough estimate of the number of records in the document (or the part of the document) being measured. For our example, $|DV| \div |OV| = 5.355$. This is a little low as an estimate for the seven records in Figure 1, but these car ads are a little shorter than typical car ads.

```
input: Application ontology O and applicable Web document D.
output: Data file F containing reorganized individual records.
1. Parse O and compute the magnitude of the ontology vector;
2. Parse D and compute the magnitude of the document vector;
3. Count occurrences, NrOcc, for each HTML tag in D
     and obtain the best-guess record boundary, btag;
4. For each text component TC between successive btag's in D:
     Obtain the linked documents for all off-page links in TC;
     Let TC' be TC with the text of the linked documents
          inserted for the off-page links in TC;
     If VSMmeasure(TC') > VSMmeasure(TC),
          Replace TC by TC';
5. For each text component TC in D:
     Let pTC be the previously text component, if any;
     Let fv be the current potential outside-boundary
          factored value, if any;
     5a. Use the VSM measure to reorganize and process
              joined records in TC:
          Obtain any inside-boundary factored values;
          Divide the joined records into individual records;
          Distribute any inside-boundary factored values to the
               previously-joined but now individual records;
     5b. Use the VSM measure to recognize and combine
              adjacent record fragments and distribute
              outside-factored values to individual records:
          If VSMmeasure(pTC + TC) > VSMmeasure(TC),
              Combine pTC and TC as the next pTC;
          Else
              add pTC to F;
              If VSMmeasure(fv + TC) > VSMmeasure(TC)
                   Combine fc and TC as the next pTC
              Else
                   If the VSM measure determines that TC
                             is a new potential factored value;
                        fv is set to TC;
                        pTC is set to empty;
                   Else pTC is set to TC;
   add pTC to F
6. For each record R in F:
        Use the VSM measure to discard inapplicable
             interspersed records;
7. Output each remaining record in R.
```

Figure 4: Record Location and Rearrangement (RLR) Algorithm

4. RECORD LOCATION AND REARRANGEMENT ALGORITHM

Figure 4 shows our algorithm for locating and rearranging records within a given Web document D. We assume that D has been filtered with respect to the application ontology and is thus a multiple-record Web document suitable for the application.[2] D may contain only a list of regular car ads, in which case the algorithm simply outputs the list, or it may contain a mixture of any or all of the categorized problems: having records that are factored, joined inside boundaries, off-page or partially off-page, split across boundaries, or interspersed among inapplicable records. The algorithm is based on a hill-climbing search that improves the VSM measure for a single record. When processing a text component within D, the possibility that the current record is factored, split, or joined is checked based on the VSM measure. The adjusted record maximizes the VSM measure by the operations in the algorithm.

[2]In previous work [10], we have shown how to identify car-ads Web documents with over 90% accuracy.

Comments on the steps of our Record Location and Rearrangement (RLR) Algorithm follow:

Step 1: The ontology vector OV is constructed based on the cardinality information in the parsed ontology.

Step 2: We use a standard algorithm [6] to parse D. The document vector DV is constructed based on the extracted data from D.

Step 3: Since OV is the vector for a single generic record and DV is the vector representing every record in a document, the length of DV divided by the length of OV ($|DV| \div |OV|$) approximates the number of records in D. To find the best-guess boundary tag, we count the occurrences of each HTML tag and compare it to $N = |DV| \div |OV|$. Our comparison uses a list of HTML tags L each of which typically separate records. We choose the HTML tag in L whose count is greater than N but smaller than the count for any other tag in L with a count greater than N. If no such tag exists, we choose the tag in L whose count is highest. If D contains no tags in L, we choose the HTML tag in a similar way, but without reference to L. After obtaining the best-guess boundary tag, we discard superfluous header and trailer text leaving just the record data.[3]

The best-guess boundary tag separates text components. Before continuing, the algorithm computes the vector for each text component TC. We denote this vector $VSMmeasure(TC)$ and we use it as the baseline measure for hill climbing.

Step 4: The idea in this step is to gather relevant off-page information into each of the records. Although cascading links beyond a single page are possible, we found none in the Web documents we encountered. Our algorithm, as presently coded, does not handle cascading links.

Step 5a: We use threshold values for the VSM measure to check whether the current text component contains multiple records. We also check the head and the tail of the current text component to see if there are inside-boundary factored values that do not appear in the middle of the joined multiple records. When distributing the data into individual records, the algorithm uses a multiple-slot template and assumes that the layout pattern for these records is regular.

Steps 5b: In this step we look (1) for adjacent components, which when joined together would improve the VSM measure, and (2) for outside-boundary factored values, which when distributed to the text components would improve the VSM measure. If we have a previous text component, we will have already joined it with even earlier text components, if appropriate, and we will have already distributed factored values into it, if appropriate. We compare this (possibly adjusted) prior text component with the current text component. If the VSM measure improves, we join the two together and proceed to the next text component. If not, we add the prior text component to the output file and continue our consideration of the current text component by checking whether the current factored value, if any, improves the VSM measure. If it does, we combine the current text component with the factored value and let it be the prior text component for the next iteration. Otherwise, we consider the possibility that the text component itself is the next factored value. We use the VSM measure to make this determination and either we assign the current text component to be the current outside-boundary factored value and the prior text component for the next iteration to be empty, or we simply let the current text component be the prior text component for the next iteration.

Step 6 After the rearrangements in Steps 4 through 5, we have a list of potential records. We use a C4.5 [17], machine-learned, VSM threshold [10] to check each one. If the record exceeds the VSM threshold, we keep it; otherwise we discard the below-threshold record.

[3]The details of how we discard superfluous header and trailer text is based on some additional heuristics described in [10] and is beyond the scope of our discussion here.

5. RESULTS

To guide us in creating the Record Location and Rearrangement (RLR) Algorithm, we used twelve artificial Web documents. We designed these documents to span the various cases covered in the algorithm—the Web document in Figure 1 represented the extreme case with no irregularities and the combination document in Figure 2 represented the extreme case with all types of irregularities. We constructed these two documents and the other ten from actual car-ads Web documents, but we made them "short" (less than a dozen car ads) and "sweet" (stripped of superfluous information beyond the basic record information).

Once our RLR algorithm successfully processed these twelve "training" documents, we tested our RLR algorithm on 30 actual Web documents. We had selected these 30 Web documents, three from each of 10 different geographic regions in the U.S., before developing our RLR algorithm. Indeed, we selected these documents for an entirely different purpose.

An examination of these 30 documents revealed the following: eight contained only regular car ads; thirteen contained joined car ads inside of record boundaries, all with inside-boundary factored values; one contained outside-boundary factoring; thirteen contained non-car-ads interspersed with car ads; and none contained split or off-page car ads.[4] Altogether in these documents we found 304 car ads that needed to be rearranged and 47 non-car-ads that needed to be discarded. Our RLR algorithm correctly rearranged 91% of the 304 and correctly discarded 94% of the 47. The combined output of our RLR algorithm over all ads in the 30 documents (including regular ads) correctly produced 1,041 of the grand total of 1,077 car ads, for an accuracy of 97%.

Over all 30 car-ads documents, our RLR algorithm produced

[4]Car ads are short and we thus did not and do not expect to find car-ads that are split across boundaries or are linked to off-page information. We have, however, found obituaries that have these characteristics, and we have done some successful testing of our RLR algorithm on these documents using an obituaries application ontology.

36 false drops (36 = 1,077-1,041) and 3 false positives. The 3 false positives were all ads for snowmobiles, which are a lot like car ads. Of the 36 false drops, 9 were regular car ads and 27 were rearranged car ads. For all 9 of the regular car ads and 2 of the 27 rearranged car ads, the ontology failed to find enough values (mostly models) to bring the VSM measure above the threshold, and these 11 car ads were thus improperly discarded. We can fix this problem by improving our ontology. Twenty of the false drops all came from the same "strange" within-boundary record group— "strange" because it repeated an identical phone number five times, once for every five car ads. This is a pattern we had not anticipated. Finally, five of the false drops were grouped in a joint car ad that was so badly formed that even a human had considerable difficulty trying to extract and form the car ads.

6. CONCLUDING REMARKS

We presented an approach to locating, delineating, and rearranging records in unstructured multiple-record Web documents. The key idea to the success of this approach is to heuristically maximize a VSM measure of similarity between the expectations for value occurrences specified in an application ontology and the actual occurrences found in a document. Encoding this heuristic in an algorithm allowed us to recognize and properly deal with records that had been factored, "unnaturally" split across boundaries, "unnaturally" joined within boundaries, linked to additional off-page information, and interspersed with records not applicable to the application ontology. Results from tests we conducted showed that we correctly located, delineated, and rearranged 97% of the records we encountered.

7. REFERENCES

[1] B. Adelberg. NoDoSE—a tool for semi-automatically extracting structured and semistructured data from text documents. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 283–294, Seattle, Washington, June 1998.

[2] N. Ashish and C. Knoblock. Semi-automatic wrapper generation for Internet information sources. In *Proceedings of the CoopIS'97*, 1997.

[3] S. Brin. Extracting patterns and relations from the World Wide Web. In *Proceedings of the WebDB Workshop (at EDBT'98)*, 1998.

[4] J. Cowie and W. Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, January 1996.

[5] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pages 509–516, Madison, Wisconsin, July 1998.

[6] Docuverse DOM SDK, 2000. URL: http://www.docuverse.com/htmlsdk/.

[7] R. Doorenbos, O. Etzioni, and D. Weld. A scalable comparison-shopping agent for the World-Wide Web. In *Proceedings of the First International Conference on Autonomous Agents*, pages 39–48, Marina Del Rey, California, February 1997.

[8] D. Embley, D. Campbell, Y. Jiang, S. Liddle, D. Lonsdale, Y.-K. Ng, and R. Smith. Conceptual-model-based data extraction from multiple-record Web pages. *Data & Knowledge Engineering*, 31(3):227–251, November 1999.

[9] D. Embley, Y. Jiang, and Y.-K. Ng. Record-boundary discovery in Web documents. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*, pages 467–478, Philadelphia, Pennsylvania, 31 May - 3 June 1999.

[10] D. Embley, Y.-K. Ng, and L. Xu. Filtering multiple-record Web documents based on application ontologies. 2000. (submitted for publication).

[11] D. Freitag. Information extraction from html: Application of a general machine learning approach. In *Proceedings of AAAI/IAAI*, pages 517–523, 1998.

[12] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the Web. In *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997.

[13] Home Page for BYU Data Extraction Group, 2000. URL: http://www.deg.byu.edu.

[14] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 1997 International Joint Conference on Artificial Intelligence*, pages 729–735, 1997.

[15] W. Lehnert, C. Cardie, D. Fisher, J. McCarthy, E. Riloff, and S. Soderland. Evaluating an information extraction system. *Journal of Integrated Computer-Aided Engineering*, 1(6), 1994.

[16] I. Muslea, S. Minton, and C. Knoblock. STALKER: Learning extraction rules for semistructured, Web-based information sources. In *Proceedings of AAAI'98: Workshop on AI and Information Integration*, Madison, Wisconsin, July 1998.

[17] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

[18] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[19] D. Smith and M. Lopez. Information extraction for semi-structured documents. In *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997.

[20] S. Soderland. Learning to extract text-based information from the World Wide Web. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 251–254, Newport Beach, California, August 1997.