Locating and Reconfiguring Records 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 [9], 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 [10]), 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 address this problem and propose an algorithm to find and rearrange (if necessary) records in an HTML document. The essential idea is to attempt to maximize a record-recognition heuristic with respect to a given application ontology. Tests we conducted for two widely differing applications show that this technique properly locates and reconfigures records.

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, and its selling price. 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 [9, 15]. 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

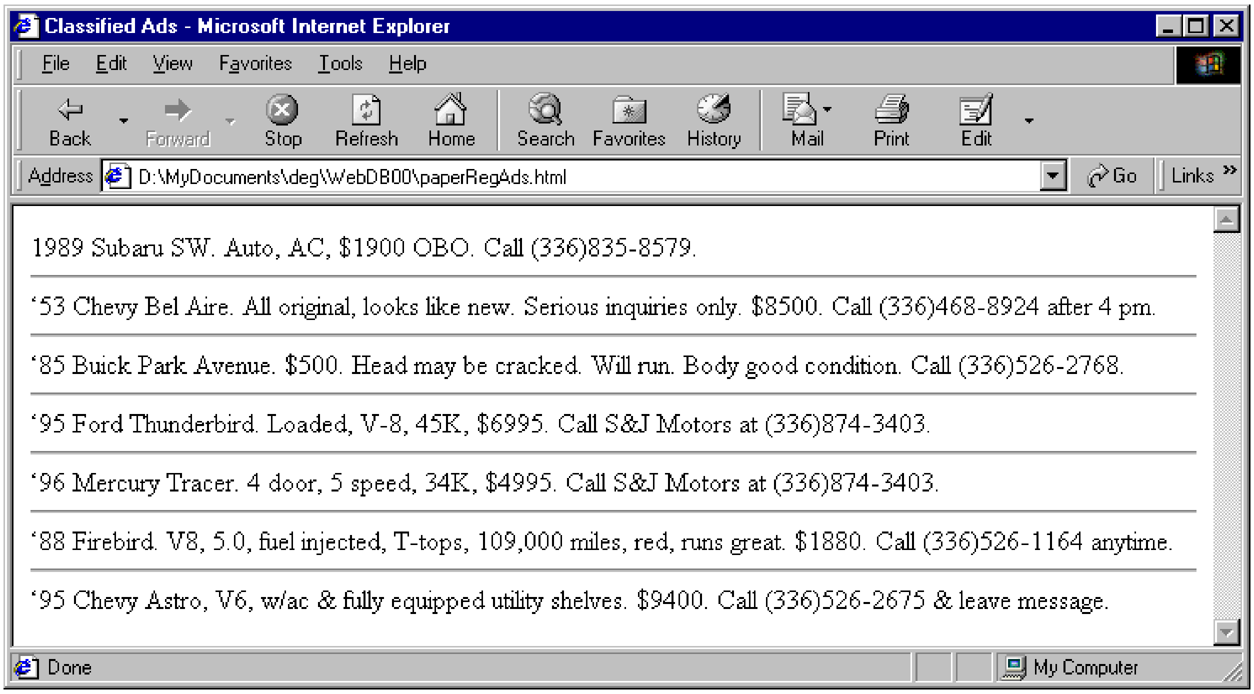
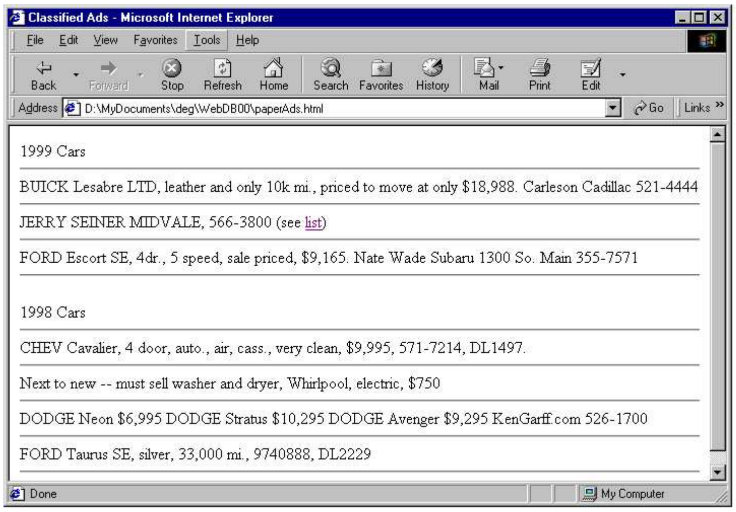


Fig. 1. Regular Car Ads

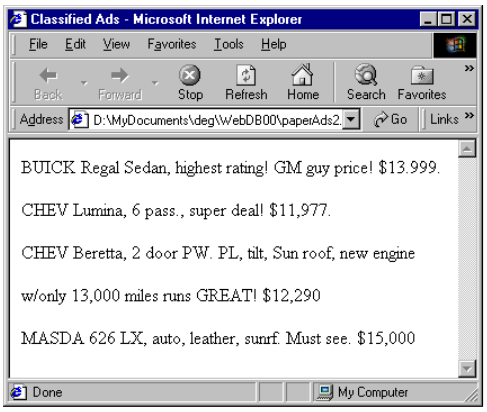
to 71%). Our results compare favorably with the results others have obtained (e.g. [1, 2, 4–6, 8, 12, 16–18, 13, 21, 22]). [1]

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 [9]. 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.

[1] For a specific comparison between our work and the work cited here, see [9].



(a) Base Car Ads



(b) Off-Page Car Ads

Fig. 2. Irregular 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 which we can use in an algorithm to locate and reconfigure unstructured records. We adapt the Vector Space Model (VSM) [20] from the field of information retrieval to define our record-recognition heuristic.

We use VSM measures to locate the information of interest in a Web document with respect to an application description by analyzing the Web document's tree structure, i.e., the document's DOM tree. The basic idea is to compute VSM measures for each subtree and find the subtree whose VSM measures most closely match the application description. As part of traversing the DOM tree and computing VSM measures, we also gather in the text of all hyperlinked documents whose VSM measures exceed an application-dependent threshold. This included text becomes part of an augmented DOM tree for the document both for computing VSM measures to locate the records of interest and for further processing in which records are rearranged.

To reconfigure records, we use VSM measures to detect how well individual unstructured records match an application description. Record reconfiguration works only within the located (possibly augmented) subtree containing the information of interest with respect to the application description. We base our approach to reconfiguring records on the idea of hill climbing. VSM measures increase when we "defactor" a set of records (e.g. distribute the years in Figure 2(a) to each of the records); increase when we divide joined records that appear within a delimited record group and increase even more when we defactor end values in joined record groups; and increase when we group records split across apparent boundaries. As a result, when we rearrange a multiple-record Web document to "maximize" VSM measures, the result becomes a list of cleaned text chunks, one for each individual unstructured record.

Section 2 describes application ontologies, which are the application descriptions we use in our work. Section 3 explains how we adapt VSM for use in record location and record reconfiguration. Section 4 presents an algorithm that uses our adapted record-location and record-reconfiguration VSM measures to find the records in a multiple-record HTML document and to rearrange them (if necessary) to produce automatically the ontology-applicable, cleaned, unstructured records contained within the document. Section 5 discuss the results of applying our record location and reconfiguration algorithm to several Web documents for car-ads and obituaries. We chose these two applications because together they cover the patterns of interest: patterns (1), (2), and (5) in Figure 2 are common in car-ad Web documents, whereas patterns (3) and (4) are common in obituary Web documents. Further, applying the heuristic algorithm in two different applications helps justify our claim that the algorithm has general applicability independent of the application domain. 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) [9]. 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 (if any) for the object set and establishes appropriate keywords that are likely to appear in a document when objects in the object set are mentioned.

In this paper, we consider two application ontologies. One is for car-ads; the other is for obituaries. Figure 3 shows part of the car-ad 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.) Figure 4 shows the graphic version for the obituary ontology. A graphic ontology provides only the object and the relationship sets and the constraints among the object and relationship sets—the data frames must have separate specifications [14]. Observe that the obituary ontology, which has 18 object sets, is much more complex than the car-ad ontology, which has only 8.

```
 1. Car [-> object];
 2. Car [0:0.911:1] has Year [1:*];
 3. Car [0:0.708:1] has Make [1:*];
 4. Car [0:0.775:1] has Model [1:*];
 5. Car [0:0.538:1] has Mileage [1:*];
 6. Car [0:2.328:*] has Feature [1:*];
 7. Car [0:0.866:1] has Price [1:*];
 8. PhoneNr [1:*] is for [1:1.061:*];
 9. Year matches [4]
10.     constant {extract ”\d{2}”;
11.         context ”\b’[4-9]\d\b”;
12.         substitute ”ˆ” -> ”19”
13.         ...
14. Mileage matches [8]
15.     ...
16.     keyword ”\bmiles\b”, ”\bmi\.”, ”\bmi\b”,
17.         ”\bmileage\b”;
18. ...
```

Fig. 3. Car-Ads Application Ontology (Partial)

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 lexical object sets whose character representations have a maximum of length 4 characters and 8 characters respectively. *Make*, *Model*, *Price*, *Feature* and *PhoneNr* are the remaining lexical object sets in our car-ads application, and *Car* is the only nonlexical object set.

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-ad ontology and *Deceased Person* for the obituary ontology. The notation "[-> object]" in Line 1 of Figure 3 and the notation "-> •" in the *Deceased Person* object set in Figure 4 designate the object sets of interest.

Textually, we denote a relationship set by a name that includes its object set names (e.g. *Car has Year* and *PhoneNr is for Car*). Graphically, we denote a relationship set by a labeled edge connecting object sets (e.g. *Person has Age* and *Deceased Person has Relationship to Person*).

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* in Figure 3, for example, specifies that a car need not have any listed features, that a car has 2.328 features on the average, and that there is no specified maximum for the number of features listed for a car.

We call a lexical object set with a 1 as the maximum participation constraint with respect to the object of interest a *1-max lexical object set*. Otherwise, if a lexical object set has a maximum participation constraint larger than 1, we call it a *n-max lexical object set*.

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

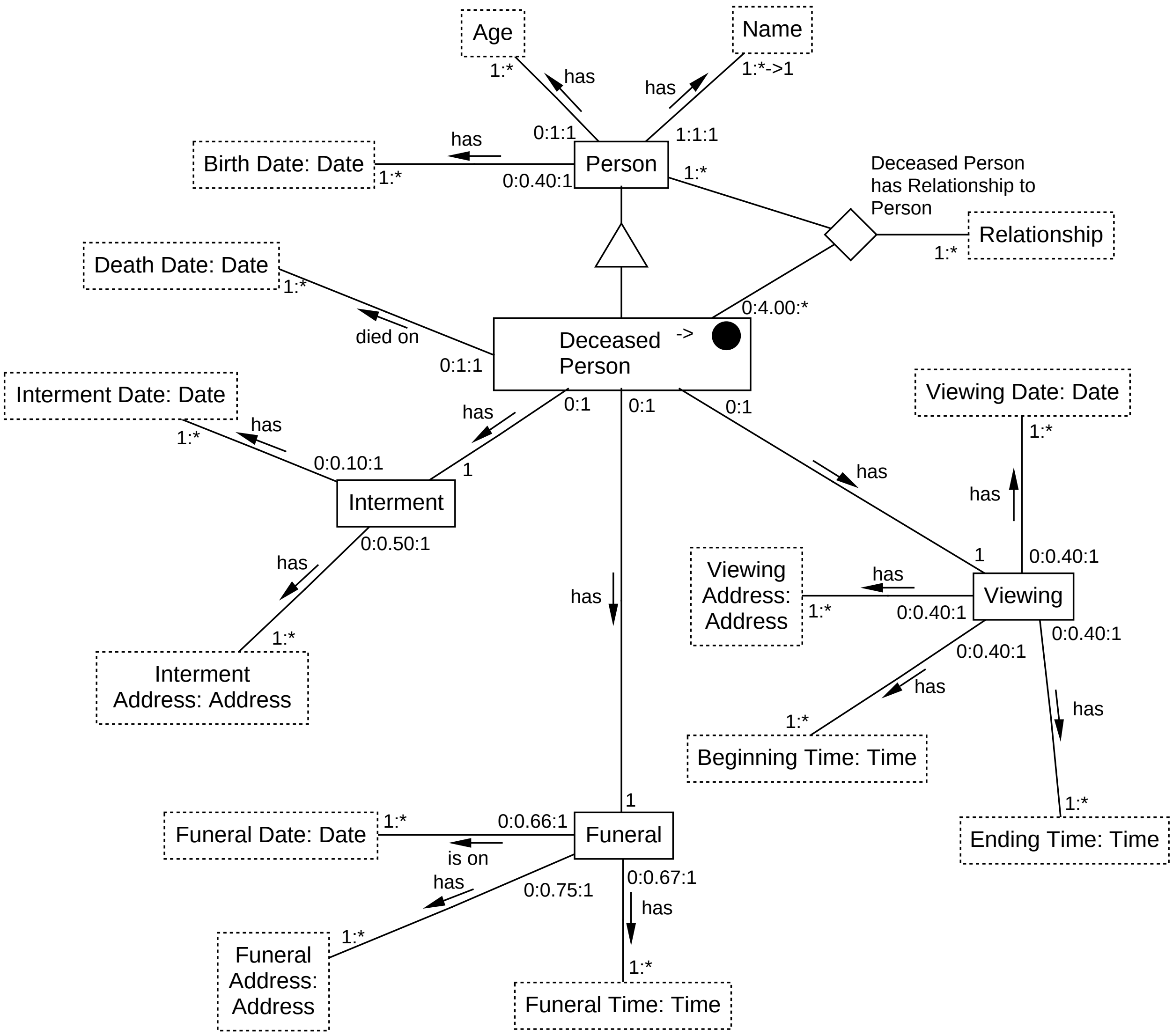


Fig. 4. Obituary Application Ontology(Graphic)

3 Record-Location and Record-Recognization Heuristics

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) [20], 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.

We construct the ontology vector OV based on the static specification of the predefined application ontology. 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. For example, the ontology vector for the car-ads application in Figure 3 is $<$ Year:0.911, Make:0.708, Model:0.775, Mileage:0.538, Price:0.866, Feature:2.328, PhoneNr:1.061 $>$.

The names of the coefficients of a document vector 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 in the document of constant values that belong to each lexical object set. By applying the car-ad 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 can construct a document vector for an entire document or for any subpart part of a document.

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 using regular expressions 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 [9]. 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. 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, instead of choosing one of the object sets arbitrarily, we discard this lexical value v. (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 [20], 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. So, if n is the number of lexical object sets and $OV =< o_1, o_2, ..., o_n >$ and $DV =< d_1, d_2, ..., d_n >$, then

$$cos\ \theta = \frac{\sum_{i=1}^{n}(o_i d_i)}{(\sqrt{\sum_{i=1}^{n} o_i^2})(\sqrt{\sum_{i=1}^{n} d_i^2})} \tag{1}$$

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 cosine measure is 0.894. 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.

The VSM measures used in this paper also includes one other measure—the magnitude. The magnitude measures the vector lengths $|OV|$ and $|DV|$, which can be calculated by the following formulas.

$$|OV| = \sqrt{\sum_{i=1}^{n} o_i^2} \tag{2}$$

$$|DV| = \sqrt{\sum_{i=1}^{n} d_i^2} \tag{3}$$

When applying the record-location and record-reconfiguration heuristics, both OV and DV are adjusted but the calculation of the VSM measures remain the same as given in formulas (1), (2), and (3). We adjust OV and DV differently and for different reasons when applying record-location and record-reconfiguration heuristics. We explain these adjustments next.

3.1 Record-Location Heuristic

When we construct the two vectors OV and DV for record location, we would like to apply the VSM measure to detect which subpart of the document contains the data related to our application ontology. We have found, however, as others have also [3], that it is difficult to use the ontology and document vectors directly. Our particular problem is that the n-max object sets tend to carry much more weight than they should. We therefore normalize our vectors by letting

$$OV =< \frac{o_1}{o_1}, \frac{o_2}{o_2}, ..., \frac{o_n}{o_n} >=< 1, 1, ..., 1 >$$

and

$$DV = < \frac{d_1}{o_1}, \frac{d_2}{o_2}, ..., \frac{d_n}{o_n} >$$

before we apply our VSM measures. Using these normalized vectors,

$$cos\ \theta = \frac{\sum_{i=1}^{n}(\frac{o_i}{o_i}\frac{d_i}{o_i})}{\sqrt{\sum_{i=1}^{n}(\frac{o_i}{o_i})^2}\sqrt{\sum_{i=1}^{n}(\frac{d_i}{o_i})^2}} = \frac{\sum_{i=1}^{n}\frac{d_i}{o_i}}{\sqrt{n}\sqrt{\sum_{i=1}^{n}(\frac{d_i}{o_i})^2}} \tag{4}$$

and the lengths of the adjusted vectors OV and DV are

$$|OV| = \sqrt{\sum_{i=1}^{n}(\frac{o_i}{o_i})^2} = \sqrt{n} \tag{5}$$

$$|DV| = \sqrt{\sum_{i=1}^{n}(\frac{d_i}{o_i})^2} \tag{6}$$

Since 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 car-ads example for the document in Figure 1, using formulas (5) and (6), $|DV| \div |OV| = 7.365$.

3.2 Record-Reconfiguration Heuristic

For record reconfiguration we need to be able to determine how well the data extracted from a text component matches a single record. As will be explained when we present our algorithm in Section 4, we use the idea of hill climbing to determine whether to combine adjacent text components and candidate factored text components. The main idea is to rearrange the text in an attempt to maximize the VSM measures for one record, making the record as close to perfect as possible.

To implement the record-reconfiguration heuristic, we first normalize OV and DV as described in Section 3.1. Then, we use the VSM magnitude measure $|DV|$, which can be calculated by formula (6), to decide if a text component contains multiple records based on an application dependent threshold value. If a text component contains multiple records, we split the text component into fine-grained components that can each be a single record or a subpart of a single record.

At this level of granularity, the cosine measure is sensitive to variations in the number of appearances of objects for each object set. To mitigate this problem, we reduced the document vector to essentially only note whether an object for an object set occurs in a text component by letting

$$e_i = \begin{cases} 0 & \text{if } d_i = 0 \\ 1 & \text{if } d_i > 0 \text{ and the } i\text{th coefficient is for a 1-max object set} \\ \lceil \frac{d_i}{C_i} \rceil & \text{if } d_i > 0 \text{ and the } i\text{th coefficient is for an } n\text{-max object set} \end{cases} \tag{7}$$

for $1 \leq i \leq n$ where C_i is the expected maximum number of occurrences for the ith n-max object set and is obtained by taking the maximum seen for object set i in the training set. Hence, $DV =< e_1, e_2, ..., e_n >$ where e_i is normally 0 or 1, with the rare exception of an integer larger than 1 if more than the the expected maximum number of objects appear in a text component for an n-max object set.

When considering whether to combine two text components TC_1 and TC_2 together into $TC_{1,2}$, we consider the respective document vectors DV_1 for TC_1, DV_2 for TC_2, and $DV_{1,2}$ for TC_1 and TC_2 combined together:

$$DV_1 =< f_{11}, f_{12}, ..., f_{1n} >$$

$$DV_2 =< f_{21}, f_{22}, ..., f_{2n} >$$

$$DV_{1,2} =< f_1, f_2, ..., f_n >$$

For DV_1, $f_{1i} = e_{1i}$ as defined in formula (7). DV_2, however, may represent previously combined components, numbered for notational convenience as components 2, 3, ..., p. Thus,

$$f_{2i} = \begin{cases} \sum_{j=2}^{p} e_{ji} & \text{if the } i\text{th coefficient is for a 1-max object set} \\ \lceil \frac{\sum_{j=2}^{p} d_{ji}}{C_i} \rceil & \text{if the } i\text{th coefficient is for an } n\text{-max object set} \end{cases}$$

where e_{ji} is the the e-value computed by formula (7) for the ith coefficient in the jth text component under consideration, and d_{ji} is the occurrence count for the ith coefficient in the jth text component under consideration. $DV_{1,2}$ represents the combination of DV_1 and DV_2 and is thus

$$f_i = \begin{cases} e_{1i} + \sum_{j=2}^{p} e_{ji} & \text{if the } i\text{th coefficient is for a 1-max object set} \\ \lceil \frac{d_{1i} + \sum_{j=2}^{p} d_{ji}}{C_i} \rceil & \text{if the } i\text{th coefficient is for an } n\text{-max object set} \end{cases} \tag{8}$$

Using these vectors for DV_1, DV_2, and $DV_{1,2}$, we obtain the following adjusted cosine measures.

$$cos_1 = \frac{\sum_{i=1}^{n} f_{1i} \frac{o_i}{o_i}}{\sqrt{\sum_{i=1}^{n} (\frac{o_i}{o_i})^2} \sqrt{\sum_{i=1}^{n} f_{1i}^2}} = \frac{\sum_{i=1}^{n} f_{1i}}{\sqrt{n} \sqrt{\sum_{i=1}^{n} f_{1i}^2}} \tag{9}$$

$$cos_2 = \frac{\sum_{i=1}^{n} f_{2i} \frac{o_i}{o_i}}{\sqrt{\sum_{i=1}^{n} (\frac{o_i}{o_i})^2} \sqrt{\sum_{i=1}^{n} f_{2i}^2}} = \frac{\sum_{i=1}^{n} f_{2i}}{\sqrt{n} \sqrt{\sum_{i=1}^{n} f_{2i}^2}} \tag{10}$$

$$cos_{1,2} = \frac{\sum_{i=1}^{n} f_i \frac{o_i}{o_i}}{\sqrt{\sum_{i=1}^{n} (\frac{o_i}{o_i})^2} \sqrt{\sum_{i=1}^{n} f_i^2}} = \frac{\sum_{i=1}^{n} f_i}{\sqrt{n} \sqrt{\sum_{i=1}^{n} f_i^2}} \tag{11}$$

Here n is the number of the lexical object sets. Observe that in formulas (9), (10), and (11), $\sqrt{n}$, which represents the $|OV|$, is static. We can therefore ignore this while computing cosine measures for hill climbing. Observe also that the adjusted vector lengths for $|DV_1|$, $|DV_2|$, and $|DV_{1,2}|$ are respectively the following.

$$|DV_1| = \sqrt{\sum_{i=1}^{n} f_{1i}^2} \tag{12}$$

$$|DV_2| = \sqrt{\sum_{i=1}^{n} f_{2i}^2} \tag{13}$$

$$|DV_{1,2}| = \sqrt{\sum_{i=1}^{n} f_{i}^2} \tag{14}$$

Intuitively, what is happening here is that we combine text components so long as we do not get "too many" 1-max objects or "too many" n-max objects. In making this test, the VSM measures let us consider all the object sets together as a unit as we attempt to hill climb by concatenating text components together. We stop concatenating whenever we start going down hill.

4 Record Location and Reconfiguration Algorithm

Figure 5 shows our algorithm for locating and reconfiguring records within a given Web document D. We assume that D has been identified as a multiple-record Web document suitable for the application.[2] D may contain only a list of regular car ads or obituaries, in which case the algorithm simply outputs the list, or D 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 locates the appropriate subpart of the document that contains the data of interest for the application ontology and brings in any hyperlinked pages with data of interest for the application ontology.. After a best-guess boundary tag is detected within the (possibly augmented) subpart of the document containing the data of interest, the algorithm uses hill climbing to improve the VSM measures for each record. When processing a text component within D, the algorithm checks for the possibility that the current record is factored, split, or joined and rearranges record components to maximize VSM measures.

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

Step 1: We construct the ontology vector OV as a normalized vector as described in Section 3.1. The magnitude of the ontology vector OV, $|OV|$, is calculated by formula (5) in Section 3.1. Thus $|OV| = \sqrt{n}$, where n is the number of the lexical object sets for the application ontology.

[2] In other work [11], we have shown how to identify both car-ad Web documents and obituary Web documents with over 90% accuracy.

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 OV;
2. Parse D into a DOM tree T, then:
    For each off-page link
        Compute the VSM measure for the linked page;
        Retain the page if the VSM measure exceeds a threshold;
    Compute the VSM measure for every subtree (including retained link pages);
3. Analyze T and the VSM measures of the tree elements and locate the subtree S
    that contains the data of interest for O;
4. Obtain the best-guess record boundary, $btag$, in S;
5. For each text component TC in S between successive $btag$s:
    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 and inside-boundary factored 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 $VSMmeasure(pTC + TC) > VSMmeasure(TC)$ and
            $VSMmeasure(pTC + TC) > VSMmeasure(pTC)$
            Combine pTC and TC as the next pTC;
        Else
            Add pTC to F;
            If $VSMmeasure(fv + TC) > VSMmeasure(TC)$ and
                $VSMmeasure(fv + TC) > VSMmeasure(fv)$
                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 TC;
    After considering all text components, add the last pTC to F
6. Use the VSM measure to discard inapplicable interspersed records in F;

Fig. 5. Record Location and Rearrangement (RLR) Algorithm

Step 2: We use a standard algorithm [7] to parse D into a DOM tree T. We then follow each off-page link and check the data in the linked document as follows. We construct the normalized document vector for the linked page as described in Section 3.1. We then compute the VSM measures for the off-page document by formulas (4) and (6) and determine based on an application-dependent threshold value whether we should retain the linked document as part of D.

Then, for each leaf node or internal node in the DOM tree T, we collect the text corresponding to the node. For a leaf node, the collected text is the text of the leaf node. For an internal node, the text is recursively collected from each node within the subtree rooted at the internal node. While collecting text, if the node contains an off-page link and the corresponding off-page document is retained, the text of the off-page document is collected as part of the text. Finally, for each node we construct the normalized document vector as described in Section 3.1 and calculate the VSM measures by formulas (4) and (6).

Step 3: Given these VSM measures for each node in T, we analyze T to locate the subtree S in T that contains the data of interest. We find S according to how close the VSM measures are to the VSM measures of T and according to whether the VSM measures of the siblings of S show that they do or do not contain useful data for the application ontology. We use the subtree S rather than T for further processing.

Step 4: We use several heuristics to obtain the best-guess record boundary tag, *btag*, within S. We apply each of these heuristics and then use Stanford Certainty Theory to calculate a combined value for each possible boundary tag. The best-guess record boundary is the HTML tag that has the largest value. The details of this step are in [10].

Step 5: We split the text of subtree S between successive *btags* into a list of text components TCs. Initially, we construct a document vector and calculate VSM measures for each text component by formulas (4) and (6).

Step 5a: We use threshold values for the VSM measures to check whether a text component TC contains multiple records. We also check the head and the tail of a 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: We look (1) for adjacent components, which when joined together would improve the VSM measures, and (2) for outside-boundary factored values, which when distributed to the text components would improve the VSM measures. In this step we construct the document vectors for text components by formula (7), and we adjust the document vectors when we combine components by formula (8). In the presentation of our algorithm in Figure 5, we denote the VSM measures for a text component TC by $VSMmeasure(TC)$. We decide whether to combine a text component TC we are considering with a previous (possibly already combined) text component pTC by computing the VSM measures for TC by formulas (9) and (12), by computing the VSM measures for pTC by formulas (10) and (13), and by computing the VSM measures for TC and pTC together by formulas (11) and (14).

Thus, if we have a previous text component, we may have already joined it with even earlier text components, if appropriate, and we may 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 VSM measures to make this determination. Either we assign the current text component to be the current outside-boundary factored value and also assign it as the prior text component for the next iteration, or we simply let the current text component be the prior text component for the next iteration.

Step 6 After the rearrangements in Step 5, we have a list of potential records. We use a C4.5 [19], machine-learned threshold [11] to check each one. To calculate the VSM measures, we construct document vector for each potential record as discussed in Section 3.1. We thus compute the VSM measure by formulas (4) and (6). If the record exceeds the threshold, we keep it; otherwise we discard the record.

5 Results

For both car ads and obituaries we provided two sets of documents, one to act as a training set and the other to act as a test set.

5.1 Car-ads

To guide us in creating the Record Location and Rearrangement (RLR) Algorithm, we used twelve artificial car-ad 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-ad 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

	Combined	Regular		Joined-Factored		Factored		Interspersed	
	Total	Total	Correct	Total	Correct	Total	Correct	Total	Correct
1.	20	20	20						
2.	66	24	24	42	41				
3.	20	20	20					3	3
4.	8	8	8						
5.	315	132	132	183	166			10	10
6.	63	47	47	16	16			1	1
7.	9	4	4	5	5				
8.	66	66	66					1	1
9.	27	25	25	2	2			3	3
10.	12	12	12						
11.	30	30	30						
12.	57	47	45	10	10				
13.	116	94	94	22	22			10	10
14.	7	7	7					7	0
15.	35	22	22	13	13				
16.	143					143	142		
17.	47	24	23	23	23				
18.	15	13	13	2	1			1	1
19.	20	20	18						
20.	9	7	7	2	2			1	1
21.	20	20	20						
22.	57	47	46	10	10				
23.	18	18	18					5	3
24.	40	40	40						
25.	10	10	10					5	4
26.	42	42	42						
27.	153	153	150					1	1
28.	55	55	55						
29.	68	43	42	25	5			1	1
30.	68	68	63						
	1616	1118	1104	355	316	143	142	49	39

Table 1. Car ontology test results.

regions in the U.S., before developing our RLR algorithm. Indeed, we selected these documents for an entirely different purpose. Table 1 shows the results.

An examination of these 30 documents revealed the following: ten 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. Altogether in these documents we found 498 car ads that needed to be rearranged and 49 non-car-ads that needed to be discarded. Our RLR algorithm correctly rearranged 92% of the 498 and correctly discarded 80% of the 49. The combined output of our RLR algorithm over all ads in the 30 documents (including regular ads) correctly produced 1,562 (1,562 = 1,104+316+142) of the grand total of 1,616 car ads, for an accuracy of 97%.

Over all 30 car-ad documents, our RLR algorithm produced 54 false drops (54 = 1,616-1,562) and 10 false positives. Seven of the ten false positives were all ads for snowmobiles, which are a lot like car ads. The other three false positives were ads for car dealers without mention of specific cars. Of the 54 false drops, 14 were regular car ads and 40 were rearranged car ads. For all 14 of the regular car ads and 20 of the 40 rearranged car ads, the ontology failed to find enough values (mostly models) to bring the VSM measures above the threshold, and these 34 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” joined record groups of one source Web document—“strange” because it repeated an identical phone number five times, once for every five car ads. This is a in-boundary factored pattern we had not anticipated. Out RLR algorithm properly split the joined records, but it lost the factored information.

An interesting problem appeared in one source document—each record included a header and a body. Each record body contained the complete, detailed information for a car, while the header contained only the summary information for the car. The algorithm split the header and the body into two records. Indeed, a human would judge each isolated header and each isolated body as being a car ad. As a pair, however, it is clear that the information in the body subsumes the information in the header. We cannot use VSM measures to solve this problem, and we therefore do not resolve this problem as part of our work for this paper. To solve the problem, we intend to send both header records and body records downstream for data extraction. Then, once the downstream processes extract the data and place records in a database, we can check the records—if the information in one record subsumes the information in another, we can discard the subsumed record.

5.2 Obituaries

For the obituary application ontology, we used 30 Web document to guide us in tuning the RLR Algorithm for obituaries. Once we obtained obituary threshold values, we tested the RLR algorithm on 30 additional Web documents. As was the case for our car-ad documents, we had selected these 60 Web documents, six from each of ten different geographic regions in the U.S., before developing our RLR algorithm. The results are in Table 2.

The 30 test documents revealed the following: twenty two contained only regular obituaries; five contained off-page obituaries; and three contained split obituaries. Altogether in these documents we found 42 obituaries that needed to be rearranged. Our RLR algorithm correctly rearranged 100% of the 42. The combined output of our RLR algorithm over all ads in the 30 documents (including regular obituaries) correctly produced 211 of the grand total of 215 obituaries, for an accuracy of 98%.

For our obituary documents, our RLR algorithm produced four false drops and two false positives. All four false drops were for regular obituaries; two each appeared in two different documents. These false drops happened because the ontology failed to find some of the expected values in these obituaries. Thus, the algorithm dropped them because their VSM measures did not exceed threshold values as they should have. Further, these same problems also caused the two

	Combined	*Regular*		*Hyperlinked*		*Split*	
	Total	*Total*	*Correct*	*Total*	*Correct*	*Total*	*Correct*
1.	9	9	9				
2.	10	10	10				
3.	4	4	4				
4.	10	10	10				
5.	5	5	5				
6.	2	2	2				
7.	8	8	8				
8.	10	10	10				
9.	3			3	3		
10.	10	10	10				
11.	7	7	7				
12.	14	14	14				
13.	10	10	8				
14.	8	8	8				
15.	7			7	7		
16.	9	9	9				
17.	4					4	4
18.	10	10	10				
19.	10	10	8				
20.	6	6	6				
21.	10	10	10				
22.	10	10	10				
23.	2					2	2
24.	7			7	7		
25.	6	6				6	6
26.	3			3	3		
27.	5	5	5				
28.	9			9	9		
29.	3	3	3				
30.	4	4	4				
	215	174	170	29	29	12	12

Table 2. Obituary ontology test results.

false-positive errors. It just so happened that in both documents the false drops were adjacent. Since the ontology recognized only some of the expected values, a combination of the two obituaries appeared to be a better obituary (according to the VSM measures) than the two separate obituaries. Thus, our algorithm combined the two pairs of false drops creating two false-positive obituaries.

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. We encoded two variations of these VSM measures—one to locate records and discard interspersed inapplicable records and the other to combine factored and split records. Our work here shows that we can recognize and properly deal with records that have 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 car-ad records we encountered and 98% of the obituary records we encountered.

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. R. Baeza-Yates and B. Ribeiro-Neto *Modern information retrieval.* ACM Press and Addison Wesley, New York, 1999.
4. S. Brin. Extracting patterns and relations from the World Wide Web. In *Proceedings of the WebDB Workshop (at EDBT'98)*, 1998.
5. J. Cowie and W. Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, January 1996.
6. 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.
7. Docuverse DOM SDK, 2000. URL: http://www.docuverse.com/htmlsdk/.
8. 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.
9. 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.
10. 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.
11. D. Embley, Y.-K. Ng, and L. Xu. Filtering multiple-record Web documents based on application ontologies. 2000. (submitted for publication).
12. D. Freitag. Information extraction from HTML: Application of a general machine learning approach. In *Proceedings of AAAI/IAAI*, pages 517–523, 1998.
13. 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.
14. K. Hewett *An integrated ontology development environment for data extraction.* Masters Thesis, Brigham Young University, April 2000.
15. Home Page for BYU Data Extraction Group, 2000. URL: http://www.deg.byu.edu.

16. 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.
17. 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.
18. 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.
19. J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.
20. G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.
21. 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.
22. 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.