Automatic Location and Separation of Records: A Case Study in the Genealogical Domain

Troy Walker and David W. Embley

Department of Computer Science
Brigham Young University, Provo, Utah 84602, U.S.A.
{troywalk,embley}@cs.byu.edu

Abstract. Locating specific chunks (records) of information within documents on the web is an interesting and nontrivial problem. If the problem of locating and separating records can be solved well, the longstanding problem of grouping extracted values into appropriate relationships in a record structure can be more easily resolved. Our solution is a hybrid of two well established techniques: (1) ontology-based extraction [ECJ+99] and (2) vector space modeling [SM83]. To show that the technique has merit, we apply it to the particularly challenging task of locating and separating records for genealogical web documents, which tend to vary considerably in layout and format. Experiments we have conducted show this technique yields an average of 92% recall and 93% precision for locating and separating genealogical records in web documents.

1 Introduction

When looking for information on the web, the challenge is usually not scarcity of data; rather, it is locating the specific data we want. Searching for genealogical information is a prime example.[1] In March 2003, a search for "Walker Genealogy" on Google returned 199,000 documents; just one year later, the same search returned 338,000 documents. Rather than requiring a human to sift through this mountain of data, it would be helpful to have a software agent that could locate and extract desired information automatically.

There are, however, many obstacles to the ideal situation in which software agents return to their owners only the data they want at the time they desire it. One large obstacle a software agent must overcome is to be able to deal with the format of each page and translate it to its own representation.[2] The web presents information in a variety of formats. Even within the HTML standard, information is presented in many ways. We can classify HTML documents by the way they present data. For our genealogy application we classify HTML documents as follows.

[1] Genealogy is also a popular example. We choose it, however, not so much because of its popularity, but because of the challenge it provides for locating, separating, and extracting records.

[2] Ontologies on the semantic web promise to make this easy, but the need to automatically or at least semi-automatically transform regular web documents into semantic web documents leaves us with the same format-recognition and translation problems.

– *Single-Record Document*: contains a record for one person (e.g. Figure 1).
– *Multiple-Record Document*: contains many records in a list (e.g. Figure 2).
– *Complex Multiple-Record Document*: contains many records organized in some sort of chart, usually an ancestral chart (e.g. Figure 3).

As humans view these pages, they effortlessly adapt to these and other presentation formats. How can we endow software agents with this same ability?

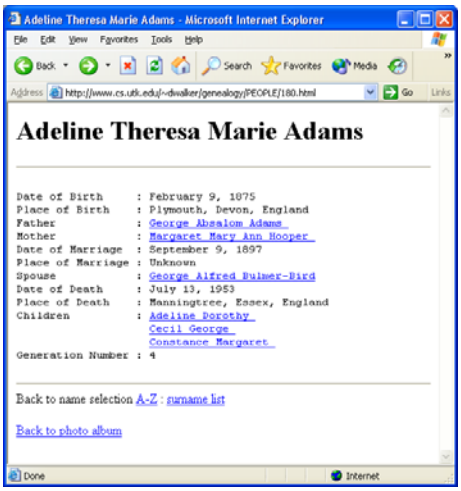

Fig. 1. Single-Record Document

Although we do not claim to be able to endow software agents with human-like format-reading ability, we do offer as our contribution in this paper a way to solve the problem of locating and separating the individual records so that the related information can be extracted as a unit. We base our solution on data extraction ontologies [ECJ+99] and on vector space modeling [SM83]. We [EJN99] and others [BLP01] have previously offered partial solutions to the problem of record separation. These solutions, however, depend on a document having multiple records consistently separated by some HTML tag such as `<hr>`. For many domains, genealogy being one good example, this assumption fails. The assumption holds for the web page in Figure 2, but fails for the web page in Figure 3 because there is no HTML tag to separate the record groups. Surprisingly, it also fails for the web page in Figure 1 because it assumes multiple records and attempts to incorrectly split the information into subcomponent groups. Recognizing the difference between a page with a large amount of genealogical information about one person and a page with a small amount of information for a few people is not as easy as it at first sounds.

We present our contribution as follows. In Section 2 we explain what an extraction ontology is and argue that the approach is likely to be best among all

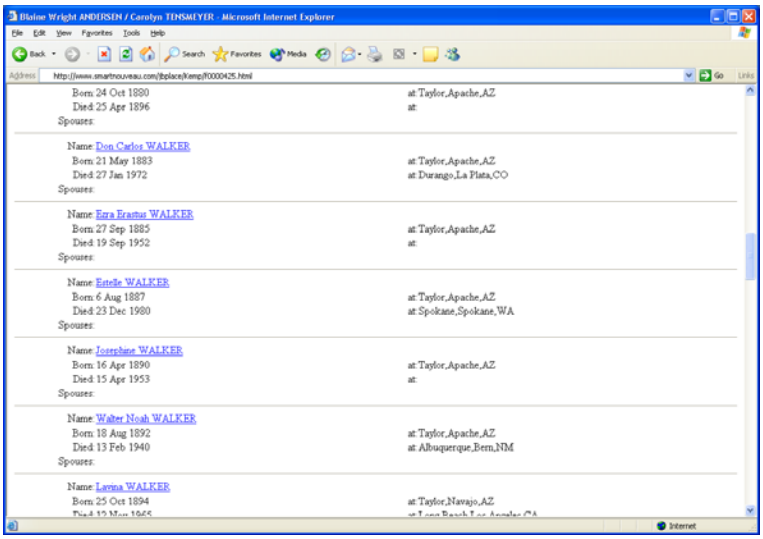

Fig. 2. Multiple-Record Document

extraction techniques for domains like genealogy where there are a huge number of continually changing web pages with a wide variety of structural formats. In Section 3 we show how to use information from extraction ontologies together with the cosine measure of vector space modeling to locate and separate records. In Section 4 we present and discuss experimental results, and in Section 5 we make concluding remarks.

2 Extraction Ontologies

Currently, the most prevalent approach to data extraction from the web is by using page-specific wrappers. (See [LRNdST02] for a survey.) Since page-specific wrappers extract data based on page layout clues, they are sensitive to changes in formatting. Because wrappers are tedious to write and must be written for each new page as well as every time a page changes, researchers have focused on semi-automatic generation of wrappers.[3] Even with semi-automatically generated wrappers, the use of page-specific wrappers requires a substantial amount of work to produce and maintain, especially for an application like genealogy where pages change often, and a substantial number of new pages continually appear. Extraction tools based on natural language processing avoid the problems of page-specific wrappers. They need clues from parsed sentences, however, to identify data of interest, which does not help for applications like genealogy where most web pages use a terse assortment of terms and fragments instead of complete sentences.

[3] There are at least 39 commercial and non-commercial wrapper generators in existence [KT02].

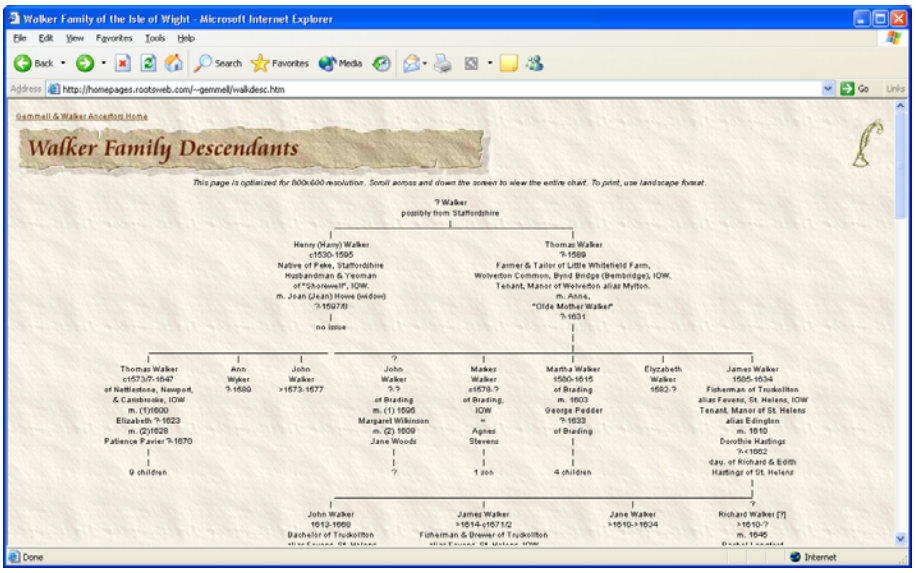

Fig. 3. Complex Multiple-Record Document

Unlike page-specific wrappers, ontology-based extraction tools are resilient to page changes and need no training or adaptation to work with new pages. Rather than using a generated page-specific wrapper for each site related to an application domain, they use an extraction ontology that wraps all pages related to a domain. For applications like genealogy, this is ideal because the effort[4] to create the ontology can be amortized over thousands (even millions) of sites.

An extraction ontology is an augmented conceptual model of the information pertaining to a narrow domain of interest (e.g. genealogy). We represent the conceptual model of our ontologies in OSM (Object-oriented System Model) as described in [EKW92]. In particular, we use a sub-model of OSM, the Object Relationship Model, which models objects, relationships among objects, and constraints. We augment this model with data frames [Emb80], which are descriptions of the data types to which objects adhere. Figure 4 shows the OSM diagram of our genealogy ontology.

OSM diagrams contain object sets, which are drawn as boxes, and relationship sets, which are drawn as lines connecting the boxes. Each object set may be either lexical (represented by a solid border) or non-lexical (represented by a dashed border) depending on their contained objects. A lexical object is an object that is indistinguishable from its representation. The object set *Name* in Figure 4 is lexical because a name is indistinguishable from its representation as a string of characters. Object sets containing numbers, dates, and even images

[4] Although the amount of effort required to create an extraction ontology is sometimes criticized, it is usually no more than the effort required to create training data for machine-learned wrapper generators. It does, however, require more expertise.

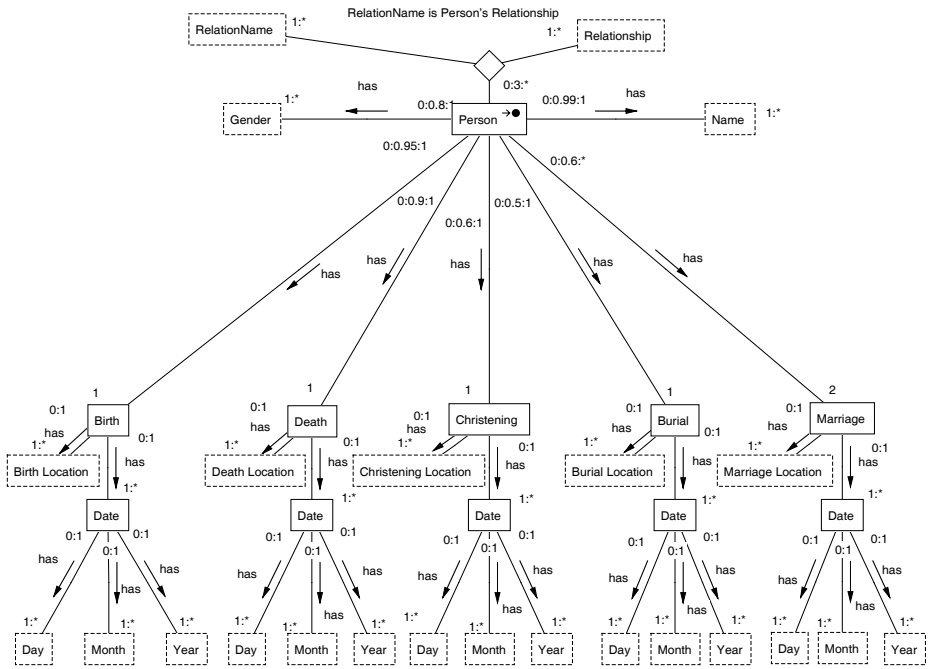

Fig. 4. Conceptual Model Diagram of Genealogy Ontology

are also lexical. Non-lexical objects are those that must be represented by a surrogate within a computer. The object set *Person* is non-lexical because there is no way to store a person in a computer. Instead, the system generates some identifier to represent the person.

In an extraction ontology, one object set is designated as the primary object set. This is the highest-level concept we are interested in extracting. An arrow and a dot designate the primary object set in our diagrams. When extracting, we only keep information that relates to an instance of this primary object set. In our genealogy ontology in Figure 4, *Person* is the primary object set.

Relationship sets connect the object sets in an extraction ontology. The relationship sets have labels and reading-direction arrows that tell how to construct the name of the relationship set. The relationship between *Person* and *Gender* in Figure 4 reads, *Person has Gender*. In relationship sets connecting more than two object sets, we replace the arrow with a diamond and the label expands to include the names of all object sets involved. The relationship among *Person*, *Relationship*, and *RelationName* has the name *RelationName is Person's Relationship*.

The relationships within a relationship set link one member of each object set to which the relationship set connects. On each connection to an object set, a relationship set has participation constraints. A participation constraint indicates how many times an object in the connecting object set may participate

in this relationship set. The participation constraint consists of a *minimum*, optional *average*, and *maximum* number separated by colons. In our notation, a star (*) represents an arbitrarily large number. When the minimum and maximum are the same, they may be represented by one number. The participation constraint next to *Person* on the relationship set between *Person* and *Gender* in Figure 4 indicates that a person may be related to at most one gender. The average value of 0.8 indicates that on pages that include gender information, we expect to find a gender for 80% of the people. A person, we know, must have exactly one gender, but we must allow no gender because we may have partial data or mistakes in extraction.

In addition to the components seen in the OSM diagram, an extraction ontology also has a data frame for each object set. A data frame contains recognizers to identify data that belongs to an object set. These recognizers consist of extended regular expressions that match values, the context typically surrounding values, and keywords usually located close to a value. Our matchers support macro substitution and the inclusion of lexicons. This helps extraction-ontology engineers keep their regular expressions manageable.

Macros:
CapPhrase: (([A-Z][A-Za-z]*)|of|the|on)(\s+(([A-Z][A-Za-z]*)|of|the|on)){0,3}
Value Phrases:
1. {CapPhrase}\,(\s+){CapPhrase}(\.?)\,(\s+){State}
2. {CapPhrase}\,(+){State}
3. {CapPhrase}\,(+){CapPhrase}\,(+){CapPhrase}\,(+){Country}
4. {CapPhrase},(+){CapPhrase},(+){Country}
5. {State}

Fig. 5. Regular Expressions for Matching Locations

Figure 5 shows the regular-expression matchers used to find locations in our extraction ontology. The words within braces are labels of macros or lexicons defined elsewhere. The macro named *CapPhrase* matches strings of capitalized words while allowing some non-capitalized prepositions. *State* and *Country* are lexicons. These are external text files containing all the states and countries plus any abbreviations and variations of spelling we could anticipate. Our genealogy ontology also uses lexicons for given names, surnames, and months.

The regular expressions in Figure 5 recognize acceptable values for locations. Along with each of these value matchers, an extraction-ontology engineer optionally specifies other regular expressions to refine the matches. Refining expressions allow the system to rule out invalid matches and to recognize right and left context surrounding legitimate values.

3 Record Location and Separation

In order to extract the information related to each person in a genealogical document, the computer needs to separate the document into records. Each record should contain information on only one person. This greatly simplifies the task of selecting values and linking them together as objects and relationships. We divide web pages into three categories based on how they present information: single-record documents (Figure 1), simple multiple-record documents (Figure 2), and complex multiple-record documents (Figure 3).

Two clues aid in locating and separating records: structure and content. A well-designed page has a format that assists readers in distinguishing records. Human readers can also guess where record divisions occur based on the data they contain—when they see a second name and another birth date in a genealogical document, they know they are reading about a new person. Previous approaches [EJN99, BLP01] primarily use structural clues to locate and separate records. Straightforward structural clues, however, may either not exist (single-record documents) or may require extensive heuristics to recognize (complex multiple-record documents. On the other hand, it is also likely to be difficult to create an accurate record separator based purely on content. In a domain where data always appears in a certain order or all data is always present, it may be straightforward, but in genealogy, this is not the case. Missing data, imperfect matchers, and unpredictable order combine to make this approach infeasible. We thus adopt a hybrid approach.

Vector Space Modeling (VSM) comes from the field of information retrieval [SM83]. A set of features from a document makes up the values in a vector from which useful cosine and magnitude measures are derived. For record location and separation, we use the object sets from the extraction ontology as dimensions in vector space.

First, we create a vector that represents a prototypical record of genealogical data. We call this vector the *Ontology Vector* (OV). To create the OV, we use the average value given in participation constraints. We do not include dimensions for all object sets; instead, we include dimensions only for those object sets most closely related to the primary object set. These object sets give us the information most helpful for locating and separating instances of the primary object set while more indirectly related object sets have more of a potential for ambiguity and conflicts or for being completely unrelated. Specifically, we create the OV by following relationships from the primary object set until we encounter an object set with a keyword matcher or a value matcher. The dimensions and dimension values for the OV selected for the ontology in Figure 4 are (<$Gender$, 0.8>, <$Name$, 0.99>, <$Birth$, 0.95>, <$Death$, 0.9>, <$Christening$, 0.6>, <$Burial$, 0.5>, <$Marriage$, 0.6>, <$Relationship$, 3.0>, <$RelationName$, 3.0>), or, when we fix the number and order of dimensions, simply (0.8, 0.99, 0.95, 0.9, 0.6, 0.5, 0.6, 3.0, 3.0).

For each candidate record, another vector contains the number of matches found in that candidate record's portion of the document. We call this vector the *Subtree Vector* (SV) because it gives the number of value matches for each

dimension object set found within a subtree in the document's DOM tree (Document Object Module tree). Thus, for example, if in a subtree we find 50 gender references, 100 names, 100 birth dates, 100 death dates, 20 christening dates, 40 burial references, 60 marriages, and 250 pairs of relationship references and names of relatives, our SV for this subtree would be (50, 100, 100, 100, 20, 40, 60, 250, 250).

We judge each SV by its cosine score relative to the OV and by its magnitude score. The cosine of the acute angle between any two vectors in n-space is the dot product of the two vectors divided by the product of their magnitudes. This provides a reliable method of determining the similarity of SV to OV. Cosine measures close to one show that the subtree likely contains data that relates to the ontology, i.e. has the proportion of values expected by the ontology. The magnitude of SV divided by the magnitude of OV yields a rough estimate of the number of records in SV, which turns out to be accurate enough to decide whether to split a subtree into multiple records. Because large values along any dimension skew these measures, object-set dimensions that are expected to have an average of many more values than other object-set dimensions have too much weight in these measures. We therefore normalize all vectors, including the OV, prior to finding VSM scores by dividing each value in the vectors by its corresponding average in the OV. Thus our sample vector (50, 100, 100, 100, 20, 40, 60, 250, 250) becomes (63, 101, 105, 111, 33, 80, 100, 83, 83) and its cosine score is computed with respect to the normalized OV, (1,1,1,1,1,1,1,1,1).

Our method of record location and separation starts at the root of the tree and evaluates the subtree rooted at each node. If its magnitude measure is less than an empirically determined threshold value, we accept it as a record. If not, we split the subtree using the heuristics of [EJN99] to find a separator tag. In cases where these heuristics fail to find a separator (usually when a node has fewer than four child nodes), we simply use the subtrees that are children of the current node. We then use a technique from [EX00] to recombine these subtrees where appropriate. This technique combines pairs of subtrees if the combination has a better cosine measure with the OV than either of the subtrees alone. Finally, we discard the subtrees with low cosine scores (empirically determined to be less than 0.6), and we repeat the process with the remaining subtrees.

As an example, Figure 6 shows part of the DOM tree for the document in Figure 2 as well as the corresponding SVs, cosine scores, and magnitude scores. At the first level, there are only two children: `<!DOCTYPE>` and `<html>`. `<!DOCTYPE>` has a low cosine score (zero), so we discard it; `<html>` has a comparatively high cosine score so we keep it. Since `<html>` has a comparatively large magnitude, we split it into `<head>` and `<body>`. We discard `<head>` because its cosine score is comparatively low. We split `<body>` because its magnitude is comparatively high. Because `<body>` has many child nodes, we find a separating tag, `<div>`, and divide the children accordingly. We continue processing with the second `<div>` tag since its cosine score is comparatively high. After repeating this process and dividing based on table rows and then table cells, we eventually start finding individual records. The `<td>` in the example is a table cell that happens

	Subtree Vector	Cosine	Magnitude
<!DOCTYPE...>	(0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)	0.00	0.00
<html>	(0.0,150.5,93.7,84.4,0.0,0.0,80.0,7.7,7.7)	0.67	70.77
<head> ... </head>	(0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)	0.33	0.34
<body>	(0.0,150.5,93.7,84.4,0.0,0.0,80.0,7.7,7.7)	0.67	70.53
 ...header... 	(0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0)	0.00	0.00
 ...	(0.0,147.5,92.6,84.4,0.0,0.0,80.0,7.7,7.7)	0.67	69.90
<td>	(0.0,1.0,1.1,1.1,0.0,0.0,0.0,0.0,0.0)	0.58	0.61

Fig. 6. DOM Tree Selections and Vector Scores

to contain one record. Note that its magnitude score is considerably lower than the magnitude of the scores of the subtrees we split. Other table cells in this document actually contain multiple records and need to be split.

As we implemented this algorithm, we encountered three problems that limited its effectiveness. First, differences between our ontology and the schema used within documents often caused cosine measures to be too low. Second, the overrichness of data particularly in single-record documents often caused magnitude measures to be too high. Third, when separating simple multiple-record documents, our algorithm was sometimes outperformed by the old method ([EJN99]) because it did not take advantage of simple patterns in the data. We introduced refinements into our algorithm to cope with these problems.

Schema Differences. Not everyone agrees on what attributes should describe a person in genealogy. We designed our genealogy ontology to hold typical data we were likely to find on the web. It contains many attributes such as *Gender*, *Burial*, and *Christening* for which most web sites do not include data. The fact that these attributes do not appear on a page does not mean that the page is not about genealogy. It does affect the cosine scores of the document, however, and can cause valid records to have cosine scores below the threshold resulting in valid records being discarded. In Figure 6, for example, the untrimmed score of even our largest cosine scores are not as high as we would like them to be. If we detect these differences in schema at the page level and compensate for them, we can more accurately find the records within the page. We do this by pruning dimensions in our vector space. In order to detect which object sets do have matches in a document's schema, we count the object set matches and prune any dimension with no matches. Since a few erroneous matches are possible, we also prune dimensions with counts less than 5% of the

average count, weighted according to the participation constraints.

Over-richness of Data. Single-record documents tend to include more complete information than multiple-record documents. It is not unusual to find seven name matches in one single-record document. Single-record documents may also repeat information or have multiple instances of one keyword. Magnitude measures for records in single-record documents are therefore much higher than measures for multiple-record documents. To overcome this, we programmed our record locator and separator to require a higher magnitude to split a document than that needed to split at the highest level within a document.

Missed Simple Patterns. Simple multiple-record documents are distinguished by a simple list pattern. On some simple multiple-record documents, our old technique ([EJN99]) is able to produce more correct records than our new technique. In any document, some records contain more or fewer details than others. Sometimes our matchers do not accept all the valid data such as when names are incomplete or contain name components not in our lexicon. At times, this variation is enough to cause our record separator to erroneously discard or split a valid record. We can take advantage of the pattern in a simple multiple-record document if we can detect the pattern. We do so when we split a subtree by counting the records and computing the ratio of records with sufficient cosine scores and low enough magnitudes to be a single record to the total number of records. If there are more than three records and at least two-thirds of them are single records, we consider all of them to be single records. As a further refinement, we eliminate headers and footers by discarding records with low cosine scores at the head and tail of the list.

4 Experimental Results

While implementing our system, we used a few example documents to debug our code. Once our system was ready, we gathered 16 additional documents to test our algorithm and made further refinements. When our system performed adequately on this tuning set, we were confident it would perform well on any genealogical page from the web. We gathered test documents by searching the web for common surnames and genealogy. To ensure stability and reproducibility throughout our test and to reduce load on the web hosts, we created a local cache of our test pages. When collecting pages, we found that there were about three generators commonly used for genealogical web pages. Since we were interested in evaluating our system on a wide variety of sources, we only included a few pages generated by each program. Further, some pages contained close to a hundred records while others contained just one. To reduce skewing for record counts, we trimmed long documents to between ten and twenty records. We were careful to preserve any footers that might exist on each document.

We divided our test documents into three groups: single-record documents, simple multiple-record documents, and complex multiple-record documents. To test record location and separation, we compared the records produced to records that should have been produced. Because of the nested nature of genealogical

data, this was not always simple. A name by itself in some contexts could be considered to be a record while in other contexts the name may just be the name of a relative within a valid record. As a general rule, we considered information about a relative to be a distinct record if it contained more than just a name.

Single-Record Documents. We tested 21 single-record documents. As can be seen in Table 1, our record separator correctly handled most of these documents resulting in 90% recall and 73% precision. This success is due to the refinement we made to compensate for over-richness of data. In two cases, data was still rich enough to overwhelm even this refinement. Attempting to split these records, the system destroyed the relationships within these records and produced several incorrect records, which explains the relatively low precision. We could increase the threshold to cover more of these cases, but raising it too much would cause multiple-record documents not to be split. Thus, our refinement worked fairly well, but since it is not just a simple matter of finding a proper balance, it is clear that a different approach is needed to produce even better results.

	records	returned	correct	precision	recall
single1	1	1	1	100.00%	100.00%
single2	1	1	1	100.00%	100.00%
single3	1	1	1	100.00%	100.00%
single4	1	1	1	100.00%	100.00%
single5	1	1	1	100.00%	100.00%
single6	1	1	1	100.00%	100.00%
single7	1	1	1	100.00%	100.00%
single8	1	4	0	0.00%	0.00%
single9	1	1	1	100.00%	100.00%
single10	1	1	1	100.00%	100.00%
single11	1	1	1	100.00%	100.00%
single12	1	3	0	0.00%	0.00%
single13	1	1	1	100.00%	100.00%
single14	1	1	1	100.00%	100.00%
single15	1	1	1	100.00%	100.00%
single16	1	1	1	100.00%	100.00%
single17	1	1	1	100.00%	100.00%
single18	1	1	1	100.00%	100.00%
single19	1	1	1	100.00%	100.00%
single20	1	1	1	100.00%	100.00%
single21	1	1	1	100.00%	100.00%
Total	21	26	19	73.08%	90.48%

Table 1. Single-Record Document Results

Simple Multiple-Record Documents. Table 2 shows the results of our

experiments on simple multiple-record documents. By using our refinement for exploiting patterns in simple documents, we were able to correctly process seven more documents than we would have otherwise and achieved 95% precision and 93% recall. In some documents, we lost the first record because it did not have a high enough cosine score and was misinterpreted as part of the header. In one case less than two thirds of the records were acceptable as single records, so the algorithm did not detect that it should have treated it as a simple pattern that could be better handled in other ways.

	records	returned	correct	precision	recall
simple1	19	20	19	95.00%	100.00%
simple2	19	17	17	100.00%	89.47%
simple3	11	11	11	100.00%	100.00%
simple4	9	9	9	100.00%	100.00%
simple5	12	13	11	84.62%	91.67%
simple6	12	11	10	90.91%	83.33%
simple7	14	10	10	100.00%	71.43%
simple8	5	7	5	71.43%	100.00%
simple9	14	14	14	100.00%	100.00%
simple10	15	15	15	100.00%	100.00%
Total	130	127	121	95.28%	93.08%

Table 2. Multiple-Record Document Results

Complex Multiple-record Documents. Since most of the genealogical documents on the web fall into this category, performance on complex multiple-record documents for our application area is critical. Table 3 shows our results. Considering the difficulty of the task, 92% recall and precision should be seen as a very good result. The most common problem we encountered stems from conflicting matches. As currently programmed, our system has no way of knowing whether a name is a member of the *Name* object set or the *RelationName* object set and must consider it a potential match for both object sets. This becomes a problem when recombining fragments of a record. Figure 7 shows two records produced from *complex4* that should have been recombined. The first record has matches for *Name*, *Birth*, and *Marriage*. The second has matches for *Name*, *Relationship*, and *Death*. Although the names in the second record are really names of relatives, they prevented our system from recombining these two records. Since this problem prevented the correct record from being returned and created two incorrect records, it affected both the precision and the recall of our record separator. Another problem arose in *complex18*. Since the document only contained four records, its magnitude measure was low enough that it appeared to be a single-record document. Our record separator did not attempt to split it so we lost three records.

	records	returned	missed	extra	correct	precision	recall
complex1	10	10	0	0	10	100.00%	100.00%
complex2	15	15	0	0	15	100.00%	100.00%
complex3	12	12	0	0	12	100.00%	100.00%
complex4	7	9	1	3	6	66.67%	85.71%
complex5	16	15	1	0	15	100.00%	93.75%
complex6	15	16	2	3	13	81.25%	86.67%
complex7	13	12	1	0	12	100.00%	92.31%
complex8	10	10	0	0	10	100.00%	100.00%
complex9	19	20	1	2	18	90.00%	94.74%
complex10	10	10	1	1	9	90.00%	90.00%
complex11	15	11	4	0	11	100.00%	73.33%
complex12	15	15	0	0	15	100.00%	100.00%
complex13	11	11	0	0	11	100.00%	100.00%
complex14	16	18	1	3	15	83.33%	93.75%
complex15	8	8	2	2	6	75.00%	75.00%
complex16	8	9	0	1	8	88.89%	100.00%
complex17	10	11	0	0	11	100.00%	110.00%
complex18	4	1	3	0	1	100.00%	25.00%
complex19	8	11	0	3	8	72.73%	100.00%
complex20	16	13	4	1	12	92.31%	75.00%
Total	238	237	21	19	218	91.98%	91.60%

Table 3. Complex Multiple-record Document Results

BROWN, Edwin, Born 18 Apr 1899 in Somerset, Kentucky, Married 1928
Ruth V. Rosenburg dau. of Johan N. and Anna Marie Eriksson Rosenberg, he died 4 Aug 1960 in Toledo, Ohio at age 61

Fig. 7. Split Record

5 Conclusion

Based on vector space modeling, we implemented a technique to accurately locate and separate records even when these records have a complex layout such as is found in genealogy web pages. The technique we developed achieved an average of 93% precision and 92% recall in our experiments over a collection of single-record documents, multiple-record documents, and complex multiple-record documents. We added this technique to our system for ontology-based extraction.

Acknowledgements: This material is based upon work supported by the National Science Foundation under grant No. IIS-0083127.

References

[BLP01] D. Buttler, L. Liu, and Calton Pu. A fully automated object extraction system for the world wide web. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDC'01)*, Mesa, Arizona, April 2001.

[ECJ+99] D.W. Embley, D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, and R.D. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data & Knowledge Engineering*, 31(3):227–251, November 1999.

[EJN99] D.W. Embley, Y.S. 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.

[EKW92] D.W. Embley, B.D. Kurtz, and S.N. Woodfield. *Object-oriented Systems Analysis: A Model-Driven Approach*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[Emb80] D.W. Embley. Programming with data frames for everyday data items. In *Proceedings of the 1980 National Computer Conference*, pages 301–305, Anaheim, California, May 1980.

[EX00] D.W. Embley and L. Xu. Record location and reconfiguration in unstructured multiple-record web documents. In *Proceedings of the Third International Workshop on the Web and Databases (WebDB2000)*, pages 123–128, Dallas, Texas, May 2000.

[KT02] S. Kuhlins and R. Tredwell. Toolkits for generating wrappers—a survey of software toolkits for automated data extraction from websites. In M. Aksit, M. Mezini, and R. Unland, editors, *Objects, Components, Architectures, Services, and Applications for a Networked World—Proceedings of the 2002 International NetObjectDays Conference*, pages 184–198, Erfurt, Germany, October 2002.

[LRNdST02] A.H.F. Laender, B.A. Ribeiro-Neto, A.S. da Silva, and J.S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31(2):84–93, June 2002.

[SM83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.