Record-Boundary Discovery in Web Documents

D.W. Embley*
Y.S. Jiang
Y.-K. Ng†

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

Abstract

Extraction of information from unstructured or semistructured Web documents often requires a recognition and delimitation of records. (By "record" we mean a group of information relevant to some entity.) Without first chunking documents that contain multiple records according to record boundaries, extraction of record information will not likely succeed. In this paper we describe a heuristic approach to discovering record boundaries in Web documents. In our approach, we capture the structure of a document as a tree of nested HTML tags, locate the subtree containing the records of interest, identify candidate separator tags within the subtree using five independent heuristics, and select a consensus separator tag based on a combined heuristic. Our approach is fast (runs linearly for practical cases within the context of the larger data-extraction problem) and accurate (100% in the experiments we conducted).

Keywords: data extraction, data structuring, unstructured data, data records, record boundaries, record-boundary discovery, World-Wide Web, semistructured data.

*Research funded in part by Novell, Inc.
†Contact author: (801) 378-2835 (Phone), (801) 378-7775 (FAX)

1 Introduction

The amount of data available electronically on the Web has increased dramatically in recent years. Users commonly retrieve this data by browsing and keyword searching, which are intuitive, but present severe limitations [Ape94]. To overcome these limitations, some researchers have resorted to database techniques. But databases require structured data and most Web data is unstructured, or at best semistructured [BDFS97], and cannot be queried using traditional query languages.

To structure Web data for traditional database query languages, one of the most promising approaches is to build wrappers for Web documents [Ade98, AK97a, AK97b, AM97, DEW97, ECJ+98, GHR97, HGMC+97, KWD97, HGMC+97, MMK98, Sod97]. In building wrappers, we often need to divide source documents into chunks of information that correspond to records (i.e., groups of information relevant to some entity). In a Web document that lists multiple car advertisements, for example, we need to identify each individual advertisement before we can extract the information from the ad. This record identification task, by itself, is nontrivial [AK97a, AK97b].

In this paper we propose an approach to discover boundaries of records in a Web document. Once found we can separate these records and pass them on for further processing. Our main contribution here is to provide a set of individual heuristics and a way to combine these heuristics into a method for discovering record boundaries. We assume that each Web document we process (1) has multiple records and (2) contains at least one record-separator tag. We note that it is an entirely different problem to check these assumptions or to solve similar document classification problems such as to determine if a record spans multiple Web documents or if a record resides in a single Web document. We leave these issues for future research [WWW].

This is not the first time the problem of separating records in a Web document has been addressed. [AM97, HGMC+97] detect record boundaries manually. They first examine the documents, find the HTML tags that separate the records of interest, and then write a program to separate the records. [Ade98, AK97a, AK97b, DEW97, KWD97, Sod97] separate records with some degree of automation. Their approaches focus primarily on using syntactic clues, such as HTML tags, to identify record boundaries. None of these approaches is fully automatic.

Our approach differs markedly from these proposals. We first provide a heuristic for locating groups of records within a Web document[1] (Section 3). This heuristic builds a "tag tree" based on the nested structure of start- and end-tags and locates the subtree that contains the records of interest. We restrict our search for a separator tag to candidate tags found in this subtree. We next apply five different heuristics, which each individually attempts to locate a separator tag among the candidate tags (Section 4). We label these five heuristics: OM (ontology matching), SD (standard deviation), IT (identifiable "separator" tags), HT (highest-count tags), and RP (repeating-tag pattern). Each of these heuristics returns one or more candidate separator tags with a measure of certainty/uncertainty attached to each candidate. Finally, we provide a way to combine these individual heuristics to determine a consensus separator tag (Section 5) and hence discover the record boundaries. For practical cases and in the context of our overall extraction process, the entire process is $\mathcal{O}(n)$, where n is the size of an input document. We applied this approach in four different application areas using Web documents obtained from twenty different sites, which together contained thousands of records (Section 6). The results were uniformly good, attaining 100% accuracy on all sites we examined.

Before explaining the details of our approach, we begin in Section 2 with a short description of the larger context in which we use our record-boundary-detection heuristics. This short description is necessary to provide the context for our record-boundary-discovery research and to explain what we mean by an ontology and how use it in our work.

2 Context for Record-Boundary Discovery Problem

Figure 1, which we take from [ECJ+98], shows the overall process we use for extracting and structuring Web data. As depicted in the figure, the input (upper left) is a Web page, and the output (lower right) is a populated database. The figure also shows that an application ontology is an independent input. For us, an application ontology is a conceptual model augmented with additional information to describe constants and keywords for the application. This ontology describes the application of interest. When we change applications, for example from car ads, to job ads, to obituaries, to university courses, we change the ontology, and we apply the process to different Web pages. Significantly, everything else

[1] We have done all our work with HTML documents, but most of this work should carry over directly to other document type definitions, such as XML.

remains the same: the routines that extract records, parse the ontology, recognize constants and keywords, and generate the populated database instance do not change. In this way, we make the process generally applicable to any application domain.

Specifically, our approach consists of the following steps. (1) We develop the ontological model instance for the domain of interest (the *Application Ontology* in the figure). (2) We parse this ontology to generate a database scheme (the *Database Description* in the figure) and to generate rules for matching constants and keywords (the *Constant/Keyword Matching Rules* in the figure). (3) To obtain data from the Web, we invoke a *Record Extractor* (see figure) that separates an unstructured Web document into individual record-size chunks, cleans them by removing markup-language tags, and presents them as individual unstructured documents for further processing. (It is the record separation task in this component that we discuss in this paper.) (4) We invoke recognizers that use the matching rules generated by the parser to extract from the cleaned individual unstructured documents the objects and relationships from which we obtain the raw, as-yet-unorganized data to populate the model instance. The result is the *Data-Record Table* in the figure. (5) Finally, we populate the generated database scheme by using heuristics to determine which constants populate which records in the database scheme. These heuristics correlate extracted keywords with extracted constants and use cardinality constraints in the ontology to determine how to construct records and insert them into the database.

In earlier papers [ECLS98, ECJ+98], we have presented these ideas for extracting and structuring data from unstructured documents. We noted in these papers, and we reiterate here that our ontologies are assumed to be narrow in breadth (meaning that the ontology is small, having no more than a few dozen object and relationship sets in its conceptual model) and that our target documents are assumed to be rich in data (meaning that there is an abundance of recognizable constants such as email addresses, phone numbers, names of automobile makes and models, and so forth). We also presented in those papers results of experiments we conducted on three different types of unstructured documents taken from the Web, namely, car ads, job ads, and obituaries. In those experiments, our approach attained recall ratios in the range of 90% and precision ratios near 95% (except for names in obituaries, which had precision ratios near 75%). For our first paper [ECLS98], we separated car and job ads by hand, and for our second paper [ECJ+98], we used a preliminary version of the record-boundary processor, which we describe in this paper.

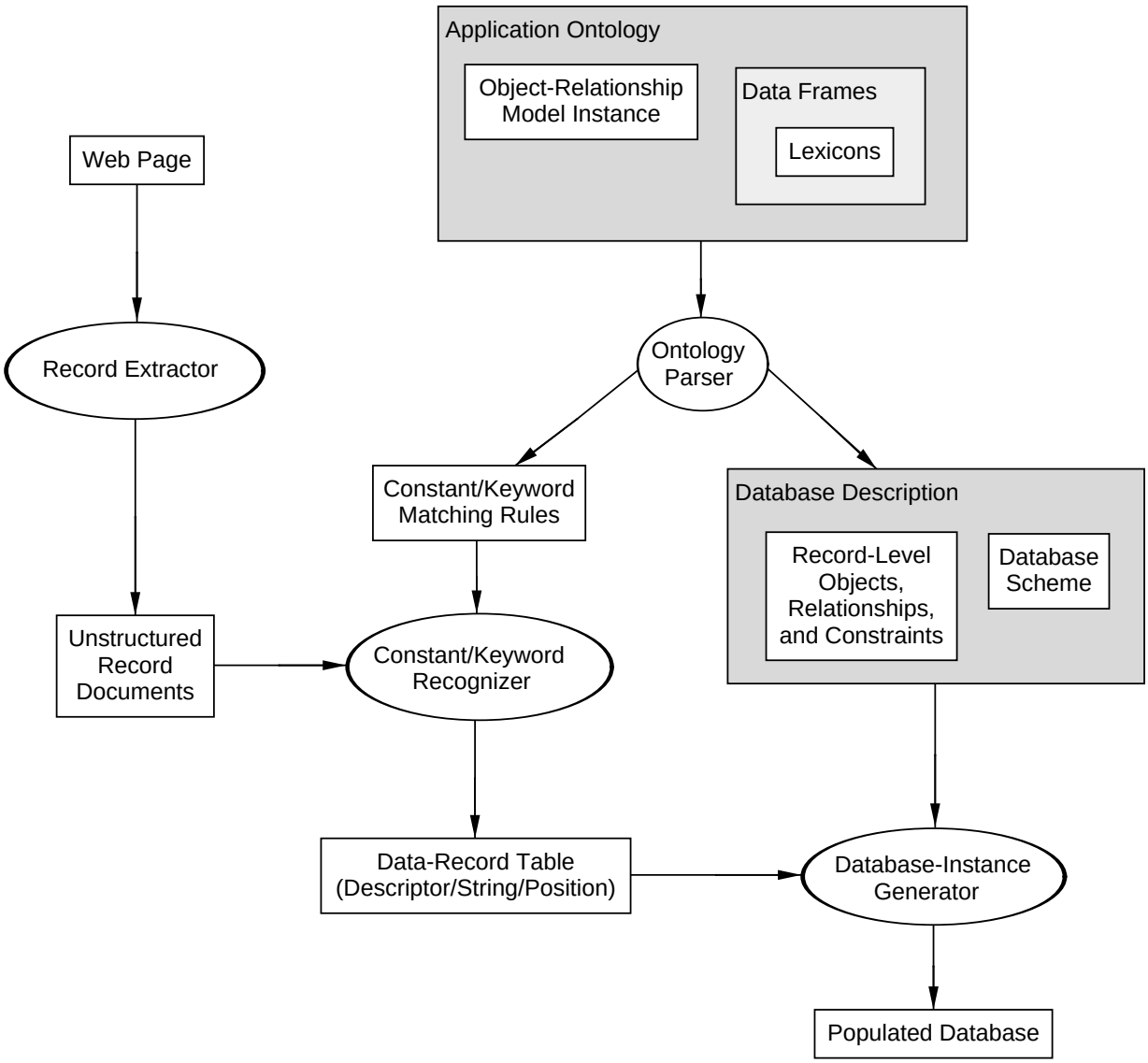


Figure 1: Data extraction and structuring process.

3 Heuristics for Locating Groups of Records

Most Web documents are hypertext documents that are written according to a document type definition such as HTML that includes plain text and tags. A *tag* in a Web document consists of a pair of opening and closing brackets, i.e., "<" and ">", that enclose a tag name, sometimes followed by a list of tag attributes, whereas *plain text* in a Web document is a sequence of characters not embedded within any tag. We distinguish each tag in a Web document as either a start-tag or an end-tag. A *start-tag* is a tag whose name does not start with a forward slash (i.e., "/"), whereas the name of an *end-tag* is the name of its corresponding start-tag preceded by "/". Some start-tags have no corresponding end-tag. In our processing we discard and thus totally ignore two special tags: (1) comment tags that start with <! and (2) any end-tag that has no corresponding start-tag.

Tags in a Web document D define discrete regions in D. A *region* R in D begins where a start-tag S appears and ends either where the the corresponding end-tag E of S appears or (if E does not exist) just before the next tag. Between a start-tag and its corresponding end-tag, other start- and/or end-tags can be nested. Regions, as defined here, do *not* necessarily correspond to regions over which a tag applies for display purposes. Our purpose here is not to display a document, but to build a convenient structure for discovering record boundaries.

Based on this nested structure, we construct a data structure, called *tag tree*, to represent a document D according to the nested regions in D. A node in the tag tree of D identifies a region in D. Using the tag tree of D along with the heuristic approaches (to be presented) in Sections 4 and 5, we attempt to detect the region containing the records of interest in D.

Figure 2(b) shows a short, sample HTML document and its corresponding tag tree T. The document in Figure 2(a) has additional plain text as indicated by the ellipses, but all the HTML tags in the document are present. In the tag tree in Figure 2(b), we use only the name of the start-tag in a node in T as the label of the node to simplify the drawing of each node in the figure. A node also contains the plain text within a tag's region. Since the tags <html> and </html> embed all of D, <html> is the start-tag of the root node R of T. Since a <head> tag exists between the <html> and </html> tags in D, a child node of R is constructed which has <head> as its start-tag. The tag <title> is the only start-tag embedded between the <head> and </head> tags in D and is thus the only child

of node *head*. The node labeled *body* is another child node of R, and the descendant nodes of the node *body* are as Figure 2(b) shows.

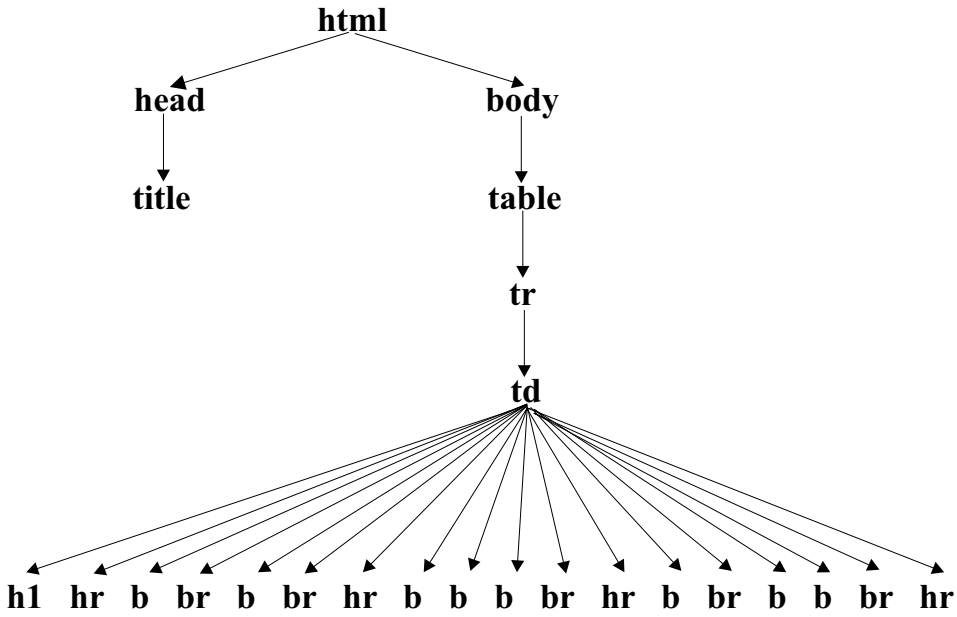
We construct the tag tree T of a Web document D as follows. (1) We initialize a stack and a table indexed by tag names and other needed data structures. (2) We scan through D to discard "useless" tags and insert *all* "missing" end tags. A "useless" tag is a tag that either start with <! or is an end-tag that has no corresponding start-tag. We use the stack, table, and a linked list to insert missing end-tags in D. All start-tags that are encountered in this pass through D are pushed onto the stack. Also, each of these start-tags is inserted into the table along its relative position in D and its location on the stack. The insertion is at the beginning of a linked list whose head is in the table at the location indexed by start-tag name. (Note that there can be multiple appearances of the same start-tag in D and that their relative positions in D and their locations on the stack should be maintained for inserting missing end-tags in D.) (3) As a final step, we scan through D again, which now has every "missing" end-tag. In this pass, we create T according to an in-order traversal. (By inserting end-tags, note that we are not preparing the document for display; instead we are preparing it to help in our search for record boundaries. The updated document is discarded once the tag tree is built.) Appendix A contains the Tag-Tree Construction algorithm which provides a more detailed description of this tag-tree construction process.

The Tag-Tree Construction algorithm has time complexity $\mathcal{O}(n)$, where n is the length of the input Web document D. Step 1, the initialization, is $\mathcal{O}(n)$, because we scan D to obtain the start-tags for initializing the table. For an appropriate list representation, adding a new entry to the table for each new start-tag and linking a new node to an existing linked list takes a constant amount of time. Since we do not have to consider a tag more than once after it has been put in the table, inserting a missing end-tag into D for its corresponding start-tag which appears in the stack is at worst $\mathcal{O}(t)$, where t is the number of tags in D. Hence, Step 2 takes at most $\mathcal{O}(t)$. In the construction Step (Step 3) of building the tag tree T of D, the number of nodes to be constructed in T is proportional to t, and the plain text we need to insert is proportional to n. Since $n > t$, the Tag-Tree Construction algorithm has time complexity $\mathcal{O}(n)$.

Given a tag tree T, our first task is to locate the subtree of T that contains the records of interest. It is our conjecture that in a Web document with multiple records of interest, the subtree of T whose root has the highest fan-out should contain the records. Indeed,

<html><head><title>Classifieds</title></head>
<body bgcolor="#FFFFFF">
<table><tr><td>
<h1 align="left">Funeral Notices - </h1> October 1, 1998
<hr>
Lemar K. Adamson
 died on March 20, 1998. Lemar was born on September 5, 1913 ...
...
church. ... BRING'S MEMORIAL CHAPEL, ...
<hr>
Our beloved Brian Fielding Frost, age 41, passed away on March 20, 1998, ...
...
held at ... in the Howard Stake Center,
Wasatch Lawn Mortuary, ...
Wasatch Lawn Memorial Park.
<hr>
Leonard Kenneth Gunther
 passed away on March 19, 1998. ...
...
... at HEATHER MORTUARY, ...
... at 11:00 a.m. at HEATHER MORTUARY, on Thursday, March 19, 1998. ... .
<hr>
</td></tr></table>
All material is copyrighted.
</body>
</html>

(a) A sample Web document



(b) The tag tree of the Web document in Figure 2(a)

Figure 2: A sample document and its tag tree

we do not consider Web documents that do not satisfy this conjecture. In Figure 2(b), the subtree rooted at **td** is the highest-fan-out subtree. Since we can find the highest fan-out subtree by a traversal of the tag tree, this operation is $\mathcal{O}(t)$, where t is the number of tags. Since t is less than n, the size of the document, this operation is bounded by $\mathcal{O}(n)$.

We next count the number of appearances of each start-tag in the immediate child nodes of N, the root node of the highest fan-out subtree, and distinguish each of these tags as either an irrelevant tag or a candidate tag. An *irrelevant* tag is a start-tag with relatively few appearances ($< 10\%$ of the total number of tags in the subtree rooted at N). In Figure 2(b), **h1** is an irrelevant tag. All tags that are not irrelevant are called *candidate* tags, because these become our candidates for record separators. The candidate tags in Figure 2(b) are **hr**, **b**, and **br**. This operation is clearly dominated by $\mathcal{O}(n)$ since a single scan of the child nodes of N is sufficient to obtain the candidate tags.

If there is only one candidate tag, we treat it as the record separator. Otherwise, we apply the heuristic approach described in the next two sections to discover the record separator.

4 Record-Boundary Discovery: Individual Heuristics

To discover the record separator, we first apply five heuristics, which individually and independently, rank the candidate tags. We then apply a consensus heuristic to combine the rankings of these individual heuristics. This section describes the individual heuristics; the next section describes the combined heuristic.

The individual heuristics span a broad range of possible techniques for discovering record boundaries. Our HT (highest-count tags) heuristic simply ranks the candidate tags based on the number of appearances; the separator tag is likely to rank high on this list when there are a large number of records. Our IT (identifiable "separator" tags) heuristic uses a predetermined list of likely HTML separator tags. Both hand-created HTML documents and tool-generated HTML documents tend to consistently use common separator tags (e.g., **hr**). Our SD (standard deviation) heuristic makes use of the observation that when multiple records about an entity appear in a document, the records are typically about the same size. Thus, the candidate tag with the minimum standard deviation based on the size of the plain text between identical tags tends to be the separator. Our RP (repeating-tag pattern) heuristic makes use of the observation that divisions between records often include several

tags that consistently appear in the same order (e.g., a **br** followed immediately by an **hr**). Our OM (ontology matching) heuristic considers the content of a record. Items that are in a one-to-one correspondence or are functional with respect to the entity of interest tend to appear once and only once in a record. If we can recognize these items, we can look for candidate tags that best separate these items into individual records.

4.1 HT: Highest-Count Tags

For the highest-count-tags heuristic, we construct an ordered list of candidate tags sorted in descending order by number of appearances in the highest-fan-out subtree. This operation is $\mathcal{O}(c\ log\ c)$ where c is the number of candidate tags. Since c is small (usually less than a dozen or so for practical cases, where c is not pathologically large) and is also much smaller than the document size n ($c << n$), we consider the cost of this operation to be negligible. Thus, for practical cases we ignore this cost in our overall estimation of the running time.

4.2 IT: Identifiable "Separator" Tags

Some Web documents are generated by using authoring tools (such as Microsoft FrontPage), and their formats (i.e., layouts) are regular. Furthermore, even in hand-generated documents, authors tend to use regular layouts. Thus, for documents with multiple records, there tends to be a few tags that consistently separate these records. By looking at these documents and keeping track of separator tags and how often authors use these tags to separate records, we can create an ordered list of the most commonly used tags that separate records of interest in Web documents.

To create our ordered separator tag list, we looked at one hundred Web documents in two application areas (obituaries and car advertisement) from ten different Web sites. Our current list is as follows:

hr td tr a table p br h4 h1 strong b i

We simply rank the candidate tags according to this list and discard any candidate tags that are not in this list. Thus, for our example document in Figure 2 we rank **hr** first, **br** second, and **b** third. We use the obvious $\mathcal{O}(cl)$ algorithm to rank the candidates according to this list. Since both the number of candidate tags c and the length of our tag list l is small compared to the size of a document, the cost of this operation is negligible.

4.3 SD: Standard Deviation

For this heuristic, we compute the standard deviation of the interval (in terms of the number of characters) between each candidate separator. For the sample document in Figure 2, we calculate the number of characters between each occurrence of **hr**, between each occurrence of **b**, and between each occurrence of **br**. We then rank the tags with the smallest standard deviation first.

We can obtain the text counts for calculating the standard deviations by a linear scan of the text in the nodes of the highest-fan-out subtree, an $\mathcal{O}(n)$ operation. Sorting the resulting standard deviations is $\mathcal{O}(c\ log\ c)$, where c is the number of candidate tags. As before, we consider this sort to be negligible for practical cases.

4.4 RP: Repeating-Tag Pattern

We base our heuristic for a repeating-tag pattern on the idea that record boundaries often have consistent patterns of two or more adjacent tags. Some tag may consistently appear before or after the record separators. In Figure 2, for example, we have the combinations <hr><b> and
<hr>.

We apply our RP heuristic to the highest-fan-out subtree of Web document with c candidate tags as follows. We count the number of occurrences of all (up to c^2) pairs of candidate tags that have no intervening plain text. If a pair <a><b> occurs at a record boundary and <a> is the record separator, the count for this pair should be about the same as the count of the number of occurrences of <a> alone. Of course, it is possible that some (or even all) counts for pairs are zero or close to zero. We only consider pairs whose count is greater than 10% of the lowest-count candidate tag. For each considered pair <a><b>, we calculate the absolute value of the difference between the count for the pair and the count for <a> alone and also the absolute value of the difference between the count for the pair and the count for <b> alone. We then rank the candidate tags in ascending order on this absolute value. Since a particular candidate tag may appear more than once in this ranking, we discard all but the best ranking for the tags in the list. We note that the list may be empty, in which case our RP heuristic simply does not supply an answer.

To analyze the running time for the RP heuristic, we first observe that we can create a tag-pair table indexed by all pairs of candidate tags in $\mathcal{O}(c^2)$ time, where c is the number of candidate tags. Next we observe that we can make a single pass through the tags in

the highest-fan-out subtree and obtain all the counts we need, an operation bounded by $\mathcal{O}(n)$, where n is the size of the Web document. Taking the absolute value for each tag of a pair and checking it for possible consideration requires a pass through the table of pairs, an $\mathcal{O}(c^2)$ operation. Since there are up to $2c^2$ tags indexing the tag-pair table, sorting each of these tags in ascending order on their absolute value and removing duplicates may take as much as $\mathcal{O}(c^2\ log\ c^2)$. For the overall cost, we note that in practical cases c is small and $c << n$. We thus treat all the operations based on c as negligible and obtain $\mathcal{O}(n)$ as the estimated cost for the RP heuristic.

4.5 OM: Ontology-Matching

We may expect one or more fields of a record to appear once and only once in the record. We call such fields *record-identifying fields*. For each record-identifying field, if we can locate a value for the field or even just an indication that the value exists, we can count the number of such occurrences. Then, if we take the average number of occurrences for several record-identifying fields in a Web document D, we have a good chance of correctly estimating the number of records in D. With this estimate, we can rank the candidate separators by how closely their number of appearances corresponds to the estimated number of records.

As an example, the death date for an obituary is a record-identifying field because there should be one and only one death date in each record. As an indication that this field exists, we use a keyword set that includes "died on" and "passed away on" (see Figure 2(a)) to indicate the existence of the field. We do not use the date itself because there may be many other fields in the record such as birth date and funeral date that are also dates. Although date values themselves are not record-identifying indicators for obituaries, the keywords that distinguish among the various dates are excellent indicators for the existence of record-identifying fields. We note that a record-identifying field is *not* the same as a key for a record, but rather is a field that is likely to occur once and only once for each record. A death date, for example, occurs once in every obituary, but a death date is not a key that identifies deceased persons in a genealogical database.

A given application ontology contains the information needed to determine the record-identifying fields. All object sets whose objects have a one-to-one correspondence with the entity of interest designate record-identifying fields as well as all object sets whose objects are functionally dependent on the entity of interest. We are selective in choosing which

record-identifying fields to consider in our ontology-matching heuristic. First, we limit the number of fields to be at least 3 and no more than 20% of the number of sets of objects in the ontology. We want at least 3, so that we can obtain a reasonable average (if we do not have at least 3 record-identifying fields, we do not use our ontology-matching heuristic). We also set an upper bound because we want to use only a few of the "best" record-identifying fields. We order the potential record-identifying fields from "best" to "worst" by first considering fields that are in a one-to-one correspondence with the entity of interest and then considering those that are functionally dependent on the entity of interest. Then, within these groups we consider keyword indicators first followed by identifiable values, except that we do not consider identifiable values that share a common type (e.g., dates in the obituary example).

To apply our ontology-matching heuristic, we first count the number of appearances of each record-identifying field in a Web document D and calculate the average number of appearances of all the record-identifying fields in D. We then consider the number of appearances of each candidate tag and rank these tags in order by how close they come to the average.

We check for the existence of a keyword or constant value by matching a regular expression with the plain text in the highest-fan-out subtree. Since this matching process is at best $\mathcal{O}(pr)$, where p is the size of the plain text and r is the number of regular expressions, the running time of the ontology-matching heuristic is not linear. We observe, however, that in the overall data-extraction process in Figure 1 we must run the regular expressions over all the plain text in the highest-fan-out subtree. We further observe that if we integrate processes, we can run the regular-expression matching process before separating records at no additional cost. This is because the entries in the *Data-Record Table* (see Figure 1) are ordered by position in the document. Once we discover the separator tag, we can use the position of the separator tags in the document to partition the *Data-Record Table* into sets of entries that are in a one-to-one correspondence with the records, and use these sets of entries for further downstream processing by the *Database-Instance Generator*. Therefore, we claim that the contribution of the ontology-matching heuristic within the overall process is no more than the contribution given the *Data-Record Table*. Since the *Data-Record Table* contains all recognized keywords and values, along with their associated object sets and their positions within the plain text of the document, a single scan through the table allows

us to obtain the counts we need. Thus, the ontology-matching heuristic is $\mathcal{O}(d)$, where d is the number of lines in the *Data-Record Table* for the plain text in the highest-fan-out subtree. Although d may be large, for practical cases it is not typically larger than n, the document size; thus, we assume $\mathcal{O}(d) < \mathcal{O}(n)$. We note that we must also sort the candidate tags, which is an $\mathcal{O}(c\ log\ c)$ operation, where c is the number of candidate tags, but as before, this operation is negligible.

5 Record-Boundary Discovery: Combined Heuristics

Each heuristic presented in Section 4 is independent of the others but works well only for some particular Web documents. We therefore consider combining these individual, independent heuristics to improve our chances of locating a correct record separator in a Web document. To determine the best combination of the five heuristics, we adopt *Stanford certainty theory* [LS97] to help us make the decision. In Section 5.1 we explain our adaptation of the Stanford certainty theory. As will be evident in this section, we will need to have certainty factors for each of our individual heuristics. To obtain these certainty factors, we conducted some initial experiments. In Section 5.2 we describe these initial experiments and how we used them to obtain the certainty factors for each of our heuristics. Given these certainty factors, we present in Section 5.3 the algorithm for our compound heuristic.

5.1 Certainty Measure

Stanford certainty theory defines a confidence measure and generates some simple rules for combining independent evidence. If evidence from two independent observations support the same result, Standard certainty theory gives the following rule to combine the evidence from these two independent observations. Suppose $CF(E_1)$ is the certainty factor associated with evidence E_1 for some observation B and $CF(E_2)$ is the certainty factor associated with evidence E_2 for the same observation B, then the new certainty factor CF of B, called the *compound certainty factor* of B, is calculated by: $CF(E_1)+CF(E_2)-(CF(E_1)\times CF(E_2))$. By using this rule repeatedly, it is possible to combine the results of evidence from any number of independent events that are used for determining B. For example, if the certainty factors are 88%, 74%, and 66% that a tag T is a record separator in a document, then the compound certainty factor for T is 98.93%. (The compound certainty factor is computed

using the Stanford certainty theory on these three factors as 88% + 74% + 66% - 88% × 74% - 88% × 66% - 74% × 66% + 88% × 74% × 66% = 98.93%.)

5.2 Initial Experiments

In order to determine the certainty factors for the individual heuristics, we considered two application areas: obituaries and car advertisements. To achieve geographical diversity (and thus hopefully a reasonable sampling of different kinds of Web documents), we chose ten on-line newspaper sites (listed in Table 1) located in different regions of the United States. For each application, we retrieved five Web documents from one site. Thus, there were 100 experimental Web documents. After scanning through these documents, we manually located the correct record separators of the documents. (Note that a Web document may have more than one record separator.) We then applied each individual heuristic on each experimental Web document and compared the output with the manually determined record separators.

Table 2 gives the results for obituaries, and Table 3 gives the results for car ads. The first row of Table 2 shows that 83% of the time the OM heuristic ranked a correct record separator of an experimental Web document as its first choice and 17% of the time the OM heuristic ranked a correct record separator as its second choice. Similarly, for the other heuristics, we calculated the percentage of Web documents in which a correct record separator was the first, second, third, or fourth choice of the ranking obtained from each of the heuristics. In these initial experiments, a correct record separator was always among the four highest ranked choices for all the heuristics.

By comparing the percentages of the two applications in Tables 2 and 3, we can see that the results are reasonably consistent in both applications. We obtained our certainty factors by averaging the percentages in Tables 2 and 3. Table 4 shows the resulting certainty factors. This table asserts that the highest ranking candidate tag chosen by the OM heuristic has a certainty factor of 84.5%, that the second highest ranking candidate tag has a certainty factor of 12.5%, and so on for the OM heuristic and also for all other heuristics.

5.3 The Compound Heuristic

For our compound heuristic we had the choice of any combination of two, three, four, or all five of the heuristics. It might seem that choosing, say, the top two or three heuristics

On-line Newspaper	*URL*
The Salt Lake Tribune	http://www.sltrib.com
The Arizona Daily Star	http://www.azstarnet.com
The Houston Chronicle	http://www.chron.com
The San Francisco Chronicle	http://www.sfgate.com
The Seattle Times	http://www.seatimes.com
GoCincinnati.com	http://classifinder.gocinci.net/
The Standard Times	http://www.s-t.com/
The Detroit Newspapers	http://www.dnps.com
The Connecticut Post	http://www.connpost.com
Access Atlanta	http://www.accessatlanta.com

Table 1: On-line newspapers for initial experiments

Heuristic Approach\Ranking	*1*	*2*	*3*	*4*
OM	83%	17%	0%	0%
RP	83%	7%	10%	0%
SD	59%	27%	14%	0%
IT	92%	8%	0%	0%
HT	58%	23%	17%	2%

Table 2: Experimental results for obituaries

Heuristic Approach\Ranking	*1*	*2*	*3*	*4*
OM	86%	8%	4%	2%
RP	72%	18%	8%	2%
SD	72%	18%	10%	0%
IT	100%	0%	0%	0%
HT	40%	42%	16%	2%

Table 3: Experimental results for car advertisements application

Heuristic Approach\Ranking	*1*	*2*	*3*	*4*
OM	84.50%	12.50%	2.00%	1.00%
RP	77.50%	12.50%	9.00%	1.00%
SD	65.50%	22.50%	12.00%	0.00%
IT	96.00%	4.00%	0.00%	0.00%
HT	49.00%	32.50%	16.50%	2.00%

Table 4: Certainty factors, as selected by our initial experiments

Compound Heuristic	*Success Rate*	*Compound Heuristic*	*Success Rate*
OR	85.83%	OSI	95.00%
OS	88.00%	OSH	87.50%
OI	95.00%	OIH	95.00%
OH	79.00%	RSI	95.00%
RS	79.50%	RSH	85.50%
RI	95.00%	RIH	95.00%
RH	76.33%	SIH	95.00%
SI	95.00%	ORSI	100.00%
SH	69.50%	ORSH	82.50%
IH	95.00%	ORIH	100.00%
ORS	81.50%	OSIH	95.00%
ORI	93.33%	RSIH	100.00%
ORH	84.83%	ORSIH	100.00%

Table 5: Experimental results for all the compound heuristics

and ignoring the rest might produce the best results. Because we did not know what combination to choose, we continued with our initial experiments and tried all combinations on the same 100 Web documents. There are $\sum_{i=0}^{5} C(5,i) - 6 (= 26)$ possible combinations (minus 6 because we cannot have none and we already have the results for the five individual heuristics).

For each combination, we calculated the compound certainty factor for each candidate tag in our experimental documents. We then determined the success rate of each combination. If there are X tags that have the highest compound certainty factors and only Y of these X tags are correct record separators in a Web document D, then the success for D, denoted $sc(D)$, is Y/X (i.e., there is $Y/X\%$ chance that the correct record separator in D is chosen). The success rate for a combination is $(\sum_{i=1}^{n}(sc(D_i))/n)$, where D_i is the ith experimental Web document. Table 5 shows the success rates for all combinations. Note that in Table 5 we use O, R, S, I, and H to represent the OM, RP, SD, IT, and HT heuristics, respectively. For example, OR denotes the OM and RP combination.

By considering the success rates in Table 5, we see that all the combinations that include IT have high success rates (over 90%). This is not surprising since it, by itself, was the best in our initial experiments as Tables 2 and 3 show. We also see, however, that ORSI, ORIH, RSIH, and ORSIH all have 100% success rate for our experimental documents;

these combinations found a correct record separator in all 100 experimental documents. In deciding among these four best choices, we observed that any one of them could be chosen as our compound heuristic. Since all five heuristics are independent and since they may all help find a correct separator, we decided to choose ORSIH, which include all five heuristics.

Thus, our heuristic algorithm for discovering record boundaries in Web documents that contain multiple records is as follows.

Algorithm. Record-Boundary Discovery Algorithm
Input: A Web document D
Output: The consensus record separator tag of D
1. call *Tag-Tree Construction Algorithm* (see Appendix A) to create the tag tree T of D
2. count the number of children of each node in T to locate the highest-fan-out subtree HF
3. extract the set of candidate tags CT from HF
4. apply the five individual heuristics OM, SD, IT, HT, and RP on CT
5. **for** each candidate tag C in CT **do begin**
   apply the Stanford certainty theory to the results of all five heuristics (ORSIH)
   using the certainty factors in Table 4
   end
6. choose the candidate tag with the highest compound certainty factor computed in Step 5 as the record separator of D

For example, consider the Web document in Figure 2(a). The results of applying five individual heuristics are as follows.

OM: [(hr, 1), (br, 2), (b, 3)]
RP: [(hr, 1), (br, 2), (b, 3)]
SD: [(hr, 1), (br, 2), (b, 3)]
IT: [(hr, 1), (br, 2), (b, 3)]
HT: [(b, 1), (br, 2), (hr, 3)]

Combining these five individual heuristics together yields the following.

ORSIH: [(hr, 99.96%), (br, 61.37%), (b, 60.42%)]

Thus, **hr** is chosen as the record separator since 99.96% is the highest percentage among the three.

We argued earlier that the time complexity of constructing the tag tree T of a Web document D is $\mathcal{O}(n)$, where n is the length of D, which is the time complexity of Step 1 in the *Record-Boundary Discovery Algorithm*. Locating the highest fan-out subtree of T in Step 2 and creating CT in Step 3 take a constant amount of time. Applying each individual heuristic in Step 4 takes at most $\mathcal{O}(n)$ time, with the understanding that D is a document found in practice and that the regular-expression matching for the OM heuristic has already been done for the larger data-extraction problem. Computing the compound certainty factor for each of the candidate tags in Step 5 using Stanford certainty theory is $\mathcal{O}(c)$, where c is the number of candidate tags, as is choosing the candidate tag with the highest compound certainty factor. Hence, the entire process for computing the consensus record separator of D is $\mathcal{O}(n)$ for practical cases within the context of the larger data-extraction problem.

6 Experimental Results

To verify the accuracy of our heuristic approach for record-boundary discovery in Web documents, we examined four sets of Web documents in four different application areas. Each set contained five Web documents from five different Web sites, 100 documents all together. The twenty Web sites we chose are located in different regions of the United States. Two of the sets were documents for obituaries and car advertisements. These test documents, however, were from entirely different sites (compare Table 1 with the site listings in Table 6 and Table 7). The other two sets were for two entirely different applications, namely computer job advertisements and university course descriptions (see the site listings in Table 8 and Table 9).

For each of the 100 Web documents, we applied the five individual heuristics presented in Section 4 and ORSIH, the compound heuristic, selected as our combined heuristic as explained in Section 5. Tables 6 - 9 shows the results. The numbers in each column are the rank numbers of the correct record separator obtained by the heuristic approach. For the *Sioux City Journal* car ads in Table 7, for example, the OM heuristic ranked the correct record separator first, RP ranked it second, SD also ranked it second, IT ranked it first, HT ranked it fourth, and ORSIH ranked it first.

We also calculated the success rates for each heuristic approach on all experimental Web documents. Table 10 shows the results. We note that even though none of the individual

On-line Newspaper	*URL*	*OM*	*RP*	*SD*	*IT*	*HT*	*ORSIH*
Alameda Newspaper	http://www.adone.com/alameda	1	1	1	1	1	1
Idaho State Journal	http://www.journalnet.com	1	1	2	1	2	1
Sacramento Bee	http://www.sacbee.com	1	1	1	1	1	1
Tampa Tribune	http://www.tampatrib.com	1	1	1	1	1	1
Shoals Timesdaily	http://www.timesdaily.com	1	1	1	1	2	1

Table 6: Test set 1 - obituaries

On-line Newspaper	*URL*	*OM*	*RP*	*SD*	*IT*	*HT*	*ORSIH*
Arkansas Democrat-Gazette	http://www.ardemgaz.com	1	1	1	1	2	1
Sioux City Journal	http://www.siouxcityjournal.com	1	2	2	1	4	1
Knoxville News	http://www.knoxnews.com	1	1	1	1	1	1
Lincoln Journal Star	http://www.nebweb.com	1	1	1	1	1	1
Reno Gazette - Journal	http://www.nevadanet.com/renogazette	3	3	1	1	3	1

Table 7: Test set 2 - car advertisements

On-line Newspaper	*URL*	*OM*	*RP*	*SD*	*IT*	*HT*	*ORSIH*
Baltimore Sun	http://www.sunspot.net	1	1	1	1	2	1
Dallas Morning News	http://dallasnews.com	1	1	2	1	2	1
Denver Post	http://www.denverpost.com	4	1	1	1	4	1
Indianapolis Star/News	http://www.starnews.com	1	1	1	1	1	1
Los Angeles Times	http://www.latimes.com	2	3	2	1	2	1

Table 8: Test set 3 - computer job advertisements

University	*URL*	*OM*	*RP*	*SD*	*IT*	*HT*	*ORSIH*
Brigham Young University	http://www.byu.edu	2	2	1	1	1	1
MIT	http://registrar.mit.edu	1	1	1	1	2	1
Kansas State University	http://www.ksu.edu	1	1	2	2	2	1
USC	http://www.usc.edu	1	1	2	1	1	1
Univ. of Texas - Austin	http://www.utexas.edu	1	2	2	1	1	1

Table 9: Test set 4 - university course descriptions

Heuristic Approach	*Success Rate*
OM	80%
RP	75%
SD	65%
IT	95%
HT	45%
ORSIH	100%

Table 10: Success rates of individual heuristics and ORSIH for experimental Web documents

heuristics had a 100% success rate, the success rate for our combined heuristic approach is 100%.

7 Concluding Remarks

We have described a heuristic approach to discovering record boundaries in unstructured Web documents containing multiple records of interest separated by one (or more) tags. In our approach, we (1) defined a tag tree to capture the structure of a raw Web document, (2) located the subtree containing the records of interest by checking for highest fan-out, (3) identified candidate tags within the subtree, (4) applied five independent heuristics (OM—ontology matching, SD—standard deviation, IT—identifiable "separator" tags, HT–highest-count tags, and RP—repeating-tag pattern) to select the best candidates, and (5) combined these heuristics using an adaptation of Stanford certainty theory to select a consensus candidate. The process is $\mathcal{O}(n)$, where n is the size of a document.

We applied this approach in four different application areas (car ads, job ads, obituaries, and university courses) using Web documents obtained from twenty different sites. The experiments we conducted showed that this approach uniformly attained an accuracy of 100%.

References

[Ade98] 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.

[AK97a] N. Ashish and C. Knoblock. Semi-automatic wrapper generation for internet information sources. In *Proceedings of the CoopIS'97*, 1997.

[AK97b] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. *SIGMOD Record*, 26(4):8–15, December 1997.

[AM97] P. Atzeni and G. Mecca. Cut and paste. In *Proceedings of the 16th ACM PODS*, pages 144–153, May 1997.

[Ape94] P. M. G. Apers. Identifying internet-related database research. In *Proceedings of the 2nd International East-West Database Workshop*, pages 183–193, Klagenfurt, 1994. Springer-Verlag.

[BDFS97] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proceedings of the International Conference on Database Theory (ICDT)*, 1997.

[DEW97] R.B. Doorenbos, O. Etzioni, and D.S. 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.

[ECJ+98] D.W. Embley, D.M. Campbell, Y.S. Jiang, Y.-K. Ng, R.D. Smith, S.W. Liddle, and D.W. Quass. A conceptual-modeling approach to extracting data from the web. In *Proceedings of the 17th International Conference on Conceptual Modeling (ER'98)*, Singapore, November 1998. (to appear).

[ECLS98] D.W. Embley, D.M. Campbell, S.W. Liddle, and R.D. Smith. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *Proceedings of the Conference on Information and Knowledge Management (CIKM'98)*, Washington D.C., November 1998. (to appear).

[GHR97] A. Gupta, V. Harinarayan, and A. Rajaraman. Virtual database technology. *SIGMOD Record*, 26(4):57–61, December 1997.

[HGMC+97] 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.

[KWD97] N. Kushmerick, D.S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of the 1997 International Joint Conference on Artificial Intelligence*, pages 729–735, 1997.

[LS97] G.F. Luger and W.A. Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Third Edition.* Addison Wesley Longman, Inc., 1997.

[MMK98] I. Muslea, S. Minton, and C. Knoblock. Stakler: Learning extraction rules for semistructured, web-based information sources. In *Proceedings of AAAI'98: Workshop on AI and Information Integration*, Madison, Wisconsin, July 1998.

[Sod97] 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.

[WWW] Homepage for BYU data extraction research group. URL: http://osm7.cs.byu.edu/deg/index.html.

Appendix

A Tag-Tree Construction Algorithm

Input: A Web document D

Output: The tag tree T of D

1. Initialization

   pass through D to obtain the set of start-tags

   initialize TABLE as an array (initially empty) such that an entry of TABLE is labeled by a start-tag and associated with a linked list of nodes

2. **repeat** /* Discard useless tags and insert missing end-tags */

   locate the next tag G in D

   if G is a comment-tag **or** G is an end-tag with no corresponding start-tag in D, **then** eliminate G from D

   else if G is a start-tag, **then**

   if G is not in TABLE, **then**

   Create an entry in TABLE with label G and push G onto stack S

   Create a node of the form $[L, Sp]$, where L is the location of the next tag in D and Sp is the location of G on S, and link it to the entry with label G in TABLE

   else /* G is an end-tag */

   Search for the corresponding start-tag of G in S

   Pop each of the tags A on top of G in S and insert the corresponding end-tag of A at L in D, where L is in node N linked to the entry G in TABLE which points to A on S

   until end-of-file(D)

3. Scan D from the beginning /* Construct the tag tree T */

   repeat

   Search for the next start-tag G in D

   Create the node $N := [G, I, O]$ in T, where I is the plain text between G and the next tag in D, and O is the plain text between the corresponding end-tag of G and the next tag in D

   Create all the descendant nodes of N

   until end-of-file(D)