Annotating Search Results from Web Databases

Yiyao Lu, Hai He, Hongkun Zhao, Weiyi Meng, *Member*, *IEEE*, and Clement Yu, *Senior Member*, *IEEE*

Abstract—An increasing number of databases have become web accessible through HTML form-based search interfaces. The data units returned from the underlying database are usually encoded into the result pages dynamically for human browsing. For the encoded data units to be machine processable, which is essential for many applications such as deep web data collection and Internet comparison shopping, they need to be extracted out and assigned meaningful labels. In this paper, we present an automatic annotation approach that first aligns the data units on a result page into different groups such that the data in the same group have the same semantic. Then, for each group we annotate it from different aspects and aggregate the different annotations to predict a final annotation label for it. An annotation wrapper for the search site is automatically constructed and can be used to annotate new result pages from the same web database. Our experiments indicate that the proposed approach is highly effective.

Index Terms—Data alignment, data annotation, web database, wrapper generation

1 Introduction

A large portion of the deep web is database based, i.e., for many search engines, data encoded in the returned result pages come from the underlying structured databases. Such type of search engines is often referred as *Web databases* (WDB). A typical result page returned from a WDB has multiple *search result records* (SRRs). Each SRR contains multiple *data units* each of which describes one aspect of a real-world entity. Fig. 1 shows three SRRs on a result page from a book WDB. Each SRR represents one book with several data units, e.g., the first book record in Fig. 1 has data units "Talking Back to the Machine: Computers and Human Aspiration," "Peter J. Denning," etc.

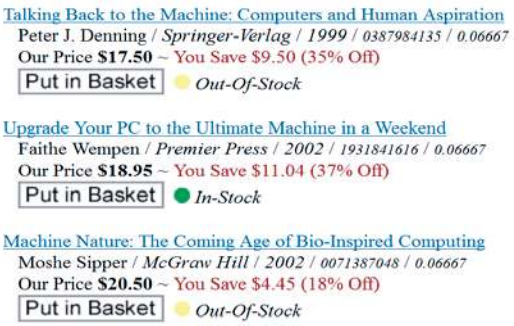
In this paper, a *data unit* is a piece of text that semantically represents one concept of an entity. It corresponds to the value of a record under an attribute. It is different from a *text node* which refers to a sequence of text surrounded by a pair of HTML tags. Section 3.1 describes the relationships between text nodes and data units in detail. In this paper, we perform data unit level annotation.

There is a high demand for collecting data of interest from multiple WDBs. For example, once a book comparison shopping system collects multiple result records from different book sites, it needs to determine whether any two SRRs refer to the same book. The ISBNs can be compared to achieve this. If ISBNs are not available, their titles and authors could be compared. The system also needs to list the prices offered by each site. Thus, the system needs to know the semantic of each data unit. Unfortunately, the semantic labels of data units are often not provided in result pages. For instance, in Fig. 1, no semantic labels for the values of title, author, publisher, etc., are given. Having semantic labels for data units is not only important for the above *record linkage* task, but also for storing collected SRRs into a database table (e.g., Deep web crawlers [23]) for later analysis. Early applications require tremendous human efforts to annotate data units manually, which severely limit their scalability. In this paper, we consider how to automatically assign labels to the data units within the SRRs returned from WDBs.

Given a set of SRRs that have been extracted from a result page returned from a WDB, our automatic annotation solution consists of three phases as illustrated in Fig. 2. Let d_i^j denote the data unit belonging to the ith SRR of concept j. The SRRs on a result page can be represented in a table format (Fig. 2a) with each row representing an SRR. Phase 1 is the *alignment phase*. In this phase, we first identify all data units in the SRRs and then organize them into different groups with each group corresponding to a different concept (e.g., all titles are grouped together). Fig. 2b shows the result of this phase with each column containing data units of the same concept across all SRRs. Grouping data units of the same semantic can help identify the common patterns and features among these data units. These common features are the basis of our annotators. In Phase 2 (the *annotation phase*), we introduce multiple basic annotators with each exploiting one type of features. Every basic annotator is used to produce a label for the units within their group holistically, and a probability model is adopted to determine the most appropriate label for each group. Fig. 2c shows that at the end of this phase, a semantic label L^j is assigned to each column. In Phase 3 (the *annotation wrapper generation phase*), as shown in Fig. 2d, for each identified

- *Y. Lu and W. Meng are with the Department of Computer Science, Binghamton University, Binghamton, NY 13902. E-mail: luyiyao@gmail.com, meng@cs.binghamton.edu.*
- *H. He is with Morningstar, Inc., Chicago, IL 60602. E-mail: hai.he@morningstar.com.*
- *H. Zhao is with Bloomberg L.P., Princeton, NJ 08858. E-mail: hkzhao@gmail.com.*
- *C. Yu is with the Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607. E-mail: cyu@uic.edu.*

Manuscript received 3 Aug. 2010; revised 25 July 2011; accepted 31 July 2011; published online 5 Aug. 2011.
Recommended for acceptance by S. Sudarshan.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2010-08-0428.
Digital Object Identifier no. 10.1109/TKDE.2011.175.



(a). Original HTML page.

```
<FORM><A>Talking Back to the Machine: Computers and
Human Aspiration</A><BR> Peter J. Denning / <FONT>
<I>Springer-Verlag / 1999 / 0387984135/0.06667</I>
</FONT> <BR>Our Price <B>$17.50</B> ~ <FONT>You
Save $9.50 (35% Off)</FONT><BR> <I>Out-Of-
Stock</I></FORM>
```

(b). Simplified HTML source for the first SRR.

Fig. 1. Example search results from Bookpool.com.

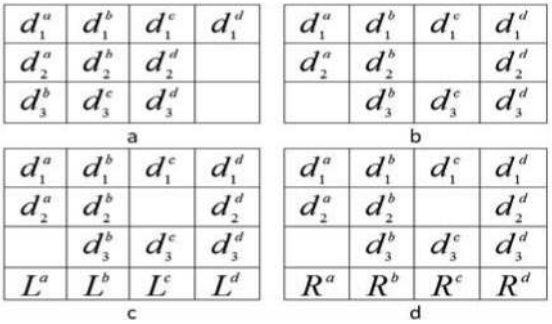


Fig. 2. Illustration of our three-phase annotation solution.

concept, we generate an annotation rule R^j that describes how to extract the data units of this concept in the result page and what the appropriate semantic label should be. The rules for all aligned groups, collectively, form the annotation wrapper for the corresponding WDB, which can be used to directly annotate the data retrieved from the same WDB in response to new queries without the need to perform the alignment and annotation phases again. As such, annotation wrappers can perform annotation quickly, which is essential for online applications.

This paper has the following contributions:

1. While most existing approaches simply assign labels to each HTML text node, we thoroughly analyze the relationships between text nodes and data units. We perform data unit level annotation.
2. We propose a clustering-based shifting technique to align data units into different groups so that the data units inside the same group have the same semantic. Instead of using only the DOM tree or other HTML tag tree structures of the SRRs to align the data units (like most current methods do), our approach also considers other important features shared among data units, such as their data types (DT), data contents (DC), presentation styles (PS), and adjacency (AD) information.
3. We utilize the integrated interface schema (IIS) over multiple WDBs in the same domain to enhance data unit annotation. To the best of our knowledge, we are the first to utilize IIS for annotating SRRs.
4. We employ six basic annotators; each annotator can independently assign labels to data units based on certain features of the data units. We also employ a probabilistic model to combine the results from different annotators into a single label. This model is highly flexible so that the existing basic annotators may be modified and new annotators may be added easily without affecting the operation of other annotators.
5. We construct an annotation wrapper for any given WDB. The wrapper can be applied to efficiently annotating the SRRs retrieved from the same WDB with new queries.

The rest of this paper is organized as follows: Section 2 reviews related works. Section 3 analyzes the relationships between data units and text nodes, and describes the data unit features we use. Section 4 introduces our data alignment algorithm. Our multiannotator approach is presented in Section 5. In Section 6, we propose our wrapper generation method. Section 7 reports our experimental results and Section 8 concludes the paper.

2 Related Work

Web information extraction and annotation has been an active research area in recent years. Many systems [18], [20] rely on human users to mark the desired information on sample pages and label the marked data at the same time, and then the system can induce a series of rules (wrapper) to extract the same set of information on webpages from the same source. These systems are often referred as a *wrapper induction system*. Because of the supervised training and learning process, these systems can usually achieve high extraction accuracy. However, they suffer from poor scalability and are not suitable for applications [24], [31] that need to extract information from a large number of web sources.

Embley et al. [8] utilize ontologies together with several heuristics to automatically extract data in multirecord documents and label them. However, ontologies for different domains must be constructed manually. Mukherjee et al. [25] exploit the presentation styles and the spatial locality of semantically related items, but its learning process for annotation is domain dependent. Moreover, a seed of instances of semantic concepts in a set of HTML documents needs to be hand labeled. These methods are not fully automatic.

The efforts to automatically construct wrappers are [1], [5], [21], but the wrappers are used for data extraction only (not for annotation). We are aware of several works [2], [28], [30], [36] which aim at automatically assigning meaningful labels to the data units in SRRs. Arlotta et al. [2] basically annotate data units with the closest labels on result pages. This method has limited applicability because many WDBs do not encode data units with their labels on result pages. In ODE system [28], ontologies are first constructed using query interfaces and result pages from WDBs in the same

domain. The domain ontology is then used to assign labels to each data unit on result page. After labeling, the data values with the same label are naturally aligned. This method is sensitive to the quality and completeness of the ontologies generated. DeLa [30] first uses HTML tags to align data units by filling them into a table through a regular expression-based data tree algorithm. Then, it employs four heuristics to select a label for each aligned table column. The approach in [36] performs attributes extraction and labeling simultaneously. However, the label set is predefined and contains only a small number of values.

We align data units and annotate the ones within the same semantic group holistically. Data alignment is an important step in achieving accurate annotation and it is also used in [25] and [30]. Most existing automatic data alignment techniques are based on one or very few features. The most frequently used feature is HTML tag paths (TP) [33]. The assumption is that the subtrees corresponding to two data units in different SRRs but with the same concept usually have the same tag structure. However, this assumption is not always correct as the tag tree is very sensitive to even minor differences, which may be caused by the need to emphasize certain data units or erroneous coding. ViDIE [21] uses visual features on result pages to perform alignment and it also generates an alignment wrapper. But its alignment is only at text node level, not data unit level. The method in [7] first splits each SRR into text segments. The most common number of segments is determined to be the number of aligned columns (attributes). The SRR with more segments are then resplit using the common number. For each SRR with fewer segments than the common number, each segment is assigned to the most similar aligned column.

Our data alignment approach differs from the previous works in the following aspects. First, our approach handles all types of relationships between text nodes and data units (see Section 3.1), while existing approaches consider only some of the types (i.e., one-to-one [25] or one-to-many [7], [30]). Second, we use a variety of features together, including the ones used in existing approaches, while existing approaches use significantly fewer features (e.g., HTML tag in [30] and [33], visual features in [21]). All the features that we use can be automatically obtained from the result page and do not need any domain specific ontology or knowledge. Third, we introduce a new clustering-based shifting algorithm to perform alignment.

Among all existing researches, DeLa [30] is the most similar to our work. But our approach is significantly different from DeLa's approach. First, DeLa's alignment method is purely based on HTML tags, while ours uses other important features such as data type, text content, and adjacency information. Second, our method handles all types of relationships between text nodes and data units, whereas DeLa deals with only two of them (i.e., one-to-one and one-to-many). Third, DeLa and our approach utilize different search interfaces of WDBs for annotation. Ours uses an IIS of multiple WDBs in the same domain, whereas DeLa uses only the local interface schema (LIS) of each individual WDB. Our analysis shows that utilizing IISs has several benefits, including significantly alleviating the *local interface schema inadequacy problem* and *the inconsistent label problem* (see Section 5.1 for more details). Fourth, we significantly enhanced DeLa's annotation method. Specifically, among the six basic annotators in our method, two (i.e., schema value annotator (SA) and frequency-based annotator (FA)) are new (i.e., not used is DeLa), three (table annotator (TA), query-based annotator (QA) and common knowledge annotator (CA)) have better implementations than the corresponding annotation heuristics in DeLa, and one (in-text prefix/suffix annotator (IA)) is the same as a heuristic in DeLa. For each of the three annotators that have different implementations, the specific difference and the motivation for using a different implementation will be given in Section 5.2. Furthermore, it is not clear how DeLa combines its annotation heuristics to produce a single label for an attribute, while we employ a probabilistic model to combine the results of different annotators. Finally, DeLa builds wrapper for each WDB just for data unit extraction. In our approach, we construct an annotation wrapper describing the rules not only for extraction but also for assigning labels.

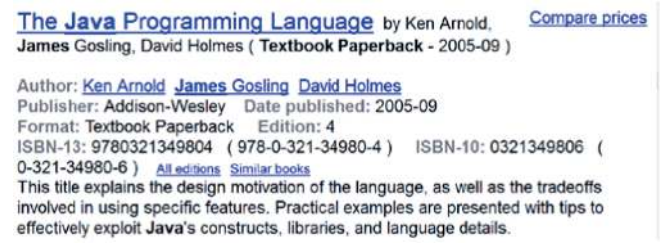
To enable fully automatic annotation, the result pages have to be automatically obtained and the SRRs need to be automatically extracted. In a metasearch context, result pages are retrieved by queries submitted by users (some reformatting may be needed when the queries are dispatched to individual WDBs). In the deep web crawling context, result pages are retrieved by queries automatically generated by the Deep Web Crawler. We employ ViNTs [34] to extract SRRs from result pages in this work. Each SRR is stored in a tree structure with a single root and each node in the tree corresponds to an HTML tag or a piece of text in the original page. With this structure, it becomes easy to locate each node in the original HTML page. The physical position information of each node on the rendered page, including its coordinates and area size, can also be obtained using ViNTs.

This paper is an extension of our previous work [22]. The following summarizes the main improvements of this paper over [22]. First, a significantly more comprehensive discussion about the relationships between text nodes and data units is provided. Specifically, this paper identifies four relationship types and provides analysis of each type, while only two of the four types (i.e., one-to-one and one-to-many) were very briefly mentioned in [22]. Second, the alignment algorithm is significantly improved. A new step is added to handle the many-to-one relationship between text nodes and data units. In addition, a clustering-shift algorithm is introduced in this paper to explicitly handle the one-to-nothing relationship between text nodes and data units while the previous version has a pure clustering algorithm. With these two improvements, the new alignment algorithm takes all four types of relationships into consideration. Third, the experiment section (Section 7) is significantly different from the previous version. The data set used for experiments has been expanded by one domain (from six to seven) and by 22 WDBs (from 91 to 112). Moreover, the experiments on alignment and annotation have been redone based on the new data set and the improved alignment algorithm. Fourth, several related papers that were published recently have been reviewed and compared in this paper.

3 Fundamentals

3.1 Data Unit and Text Node

Each SRR extracted by ViNTs has a tag structure that determines how the contents of the SRRs are displayed on a web browser. Each node in such a tag structure is either a *tag*


The Java Programming Language by Ken Arnold, James Gosling, David Holmes (Textbook Paperback - 2005-09) Compare prices

Author: Ken Arnold James Gosling David Holmes
Publisher: Addison-Wesley Date published: 2005-09
Format: Textbook Paperback Edition: 4
ISBN-13: 9780321349804 (978-0-321-34980-4) ISBN-10: 0321349806 (0-321-34980-6) All editions Similar books
This title explains the design motivation of the language, as well as the tradeoffs involved in using specific features. Practical examples are presented with tips to effectively exploit Java's constructs, libraries, and language details.

Fig. 3. An example illustrating the Many-to-One relationship.

node or a *text node*. A tag node corresponds to an HTML tag surrounded by "<" and ">" in HTML source, while a text node is the text outside the "<" and ">." Text nodes are the visible elements on the webpage and data units are located in the text nodes. However, as we can see from Fig. 1, text nodes are not always identical to data units. Since our annotation is at the data unit level, we need to identify data units from text nodes.

Depending on how many data units a text node may contain, we identify the following four types of relationships between data unit (U) and text node (T):

- *One-to-One Relationship* (denoted as $T = U$). In this type, each text node contains exactly one data unit, i.e., the text of this node contains the value of a single attribute. This is the most frequently seen case. For example, in Fig. 1, each text node surrounded by the pair of tags <A> and </A> is a value of the *Title* attribute. We refer to such kind of text nodes as *atomic text nodes*. An atomic text node is equivalent to a data unit.
- *One-to-Many Relationship* (denoted as $T \supset U$). In this type, multiple data units are encoded in one text node. For example, in Fig. 1, part of the second line of each SRR (e.g., "Springer-Verlag/1999/0387984135/0.06667" in the first record) is a single text node. It consists of four semantic data units: *Publisher*, *Publication Date*, *ISBN*, and *Relevance Score*. Since the text of such kind of nodes can be considered as a composition of the texts of multiple data units, we call it a *composite text node*. An important observation that can be made is: if the data units of attributes $A_1 \ldots A_k$ in one SRR are encoded as a composite text node, it is usually true that the data units of the same attributes in other SRRs returned by the same WDB are also encoded as composite text nodes, and those embedded data units always appear in the same order. This observation is valid in general because SRRs are generated by template programs. We need to split each composite text node to obtain real data units and annotate them. Section 4 describes our splitting algorithm.
- *Many-to-One Relationship* (denoted as $T \subset U$). In this case, multiple text nodes together form a data unit. Fig. 3 shows an example of this case. The value of the *Author* attribute is contained in multiple text nodes with each embedded inside a separate pair of (<A>, </A>) HTML tags. As another example, the tags <B> and </B> surrounding the keyword "**Java**" split the title string into three text nodes. It is a general practice that webpage designers use special HTML tags to embellish certain information. Zhao et al. [35] call this kind of tags as *decorative tags* because they are used mainly for changing the appearance of part of the text nodes. For the purpose of extraction and annotation, we need to identify and remove these tags inside SRRs so that the wholeness of each split data unit can be restored. The first step of our alignment algorithm handles this case specifically (see Section 4 for details).
- *One-To-Nothing Relationship* (denoted as $T \neq U$). The text nodes belonging to this category are not part of any data unit inside SRRs. For example, in Fig. 3, text nodes like "Author" and "Publisher" are not data units, but are instead the semantic labels describing the meanings of the corresponding data units. Further observations indicate that these text nodes are usually displayed in a certain pattern across all SRRs. Thus, we call them *template text nodes*. We employ a frequency-based annotator (see Section 5.2) to identify template text nodes.

3.2 Data Unit and Text Node Features

We identify and use five common features shared by the data units belonging to the same concept across all SRRs, and all of them can be automatically obtained. It is not difficult to see that all these features are applicable to text nodes, including composite text nodes involving the same set of concepts, and template text nodes.

3.2.1 Data Content (DC)

The data units or text nodes with the same concept often share certain keywords. This is true for two reasons. First, the data units corresponding to the search field where the user enters a search condition usually contain the search keywords. For example, in Fig. 1, the sample result page is returned for the search on the title field with keyword "machine." We can see that all the titles have this keyword. Second, web designers sometimes put some leading label in front of certain data unit within the same text node to make it easier for users to understand the data. Text nodes that contain data units of the same concept usually have the same leading label. For example, in Fig. 1, the price of every book has leading words "Our Price" in the same text node.

3.2.2 Presentation Style (PS)

This feature describes how a data unit is displayed on a webpage. It consists of six style features: *font face, font size, font color, font weight, text decoration* (underline, strike, etc.), and whether it is *italic*. Data units of the same concept in different SRRs are usually displayed in the same style. For example, in Fig. 1, all the availability information is displayed in the exactly same presentation style.

3.2.3 Data Type (DT)

Each data unit has its own semantic type although it is just a text string in the HTML code. The following basic data types are currently considered in our approach: *Date, Time, Currency, Integer, Decimal, Percentage, Symbol,* and *String*. String type is further defined in *All-Capitalized-String, First-Letter-Capitalized-String*, and *Ordinary String*. The data type of

a composite text node is the concatenation of the data types of all its data units. For example, the data type of the text node "Premier Press/2002/1931841616/0.06667" in Fig. 1 is <First-Letter-Capitalized-String> <Symbol> <Integer> <Symbol> <Integer> <Symbol> <Decimal>. Consecutive terms with the same data type are treated as a single term and only one of them will be kept. Each type except Ordinary String has certain pattern(s) so that it can be easily identified. The data units of the same concept or text nodes involving the same set of concepts usually have the same data type.

3.2.4 Tag Path (TP)

A tag path of a text node is a sequence of tags traversing from the root of the SRR to the corresponding node in the tag tree. Since we use ViNTs for SRR extraction, we adopt the same tag path expression as in [34]. Each node in the expression contains two parts, one is the *tag name*, and the other is the *direction* indicating whether the next node is the next sibling (denoted as "S") or the first child (denoted as "C"). Text node is simply represented as <#TEXT>. For example, in Fig. 1b, the tag path of the text node "Springer-Verlag/1999/0387984135/0.06667" is *<FORM>C<A>C
S<#TEXT>S<FONT>C<T>C*. An observation is that the tag paths of the text nodes with the same set of concepts have very similar tag paths, though in many cases, not exactly the same.

3.2.5 Adjacency (AD)

For a given data unit d in an SRR, let d^p and d^s denote the data units immediately before and after d in the SRR, respectively. We refer d^p and d^s as the preceding and succeeding data units of d, respectively. Consider two data units d_1 and d_2 from two separate SRRs. It can be observed that if d_1^p and d_2^p belong to the same concept and/or d_1^s and d_2^s belong to the same concept, then it is more likely that d_1 and d_2 also belong to the same concept.

We note that none of the above five features is guaranteed to be true for any particular pair of data units (or text nodes) with the same concept. For example, in Fig. 1, "*Springer-Verlag*" and "*McGraw-Hill*" are both publishers but they do not share any content words. However, such data units usually share some other features. As a result, our alignment algorithm (Section 4) can still work well even in the presence of some violation of these features. This is confirmed by our experimental results in Section 7.

4 Data Alignment

4.1 Data Unit Similarity

The purpose of data alignment is to put the data units of the same concept into one group so that they can be annotated holistically. Whether two data units belong to the same concept is determined by how similar they are based on the features described in Section 3.2. In this paper, the similarity between two data units (or two text nodes) d_1 and d_2 is a weighted sum of the similarities of the five features between them, i.e.:

$$\begin{aligned} Sim(d_1,d_2) = {} & w_1 * SimC(d_1,d_2) + w_2 * SimP(d_1,d_2) \\ & + w_3 * SimD(d_1,d_2) + w_4 * SimT(d_1,d_2) \\ & + w_5 * SimA(d_1,d_2). \end{aligned} \quad (1)$$

The weights in the above formula are obtained using a genetic algorithm based method [10] and the trained weights are given in Section 7.2. The similarity for each individual feature is defined as follows:

- **Data content similarity (*SimC*).** It is the *Cosine similarity* [27] between the term frequency vectors of d_1 and d_2:

$$SimC(d_1,d_2) = \frac{V_{d_1} \bullet V_{d_2}}{\|V_{d_1}\| * \|V_{d_2}\|}, \quad (2)$$

  where V_d is the frequency vector of the terms inside data unit d, $||V_d||$ is the length of V_d, and the numerator is the inner product of two vectors.
- **Presentation style similarity (*SimP*).** It is the average of the style feature scores (FS) over all six presentation style features (F) between d_1 and d_2:

$$SimP(d_1,d_2) = \sum_{i=1}^{6} FS_i/6, \quad (3)$$

  where FS_i is the score of the ith style feature and it is defined by $FS_i = 1$ if $F_i^{d_1} = F_i^{d_2}$ and $FS_i = 0$ otherwise, and F_i^d is the ith style feature of data unit d.
- **Data type similarity (*SimD*).** It is determined by the common sequence of the component data types between two data units. The longest common sequence (LCS) cannot be longer than the number of component data types in these two data units. Thus, let t_1 and t_2 be the sequences of the data types of d_1 and d_2, respectively, and TLen(t) represent the number of component types of data type t, the data type similarity between data units d_1 and d_2 is

$$SimD(d_1,d_2) = \frac{LCS(t_1,t_2)}{Max(Tlen(t_1),Tlen(t_2))}. \quad (4)$$

- **Tag path similarity (*SimT*).** This is the edit distance (EDT) between the tag paths of two data units. The edit distance here refers to the number of insertions and deletions of tags needed to transform one tag path into the other. It can be seen that the maximum number of possible operations needed is the total number of tags in the two tag paths. Let p_1 and p_2 be the tag paths of d_1 and d_2, respectively, and PLen(p) denote the number of tags in tag path p, the tag path similarity between d_1 and d_2 is

$$SimT(d_1,d_2) = 1 - \frac{EDT(p_1,p_2)}{PLen(p_1)+PLen(p_2)}. \quad (5)$$

  Note that in our edit distance calculation, a substitution is considered as a deletion followed by an insertion, requiring two operations. The rationale is that two attributes of the same concept tend to be encoded in the same subtree in DOM (relative to the root of their SRRs) even though some decorative tags may appear in one SRR but not in the other. For example, consider two pairs of tag paths (<T1><T2><T3>, <T1> <T3>) and (<T1><T2><T3>, <T1><T4><T3>). The two tag

paths in the first pair are more likely to point to the attributes of the same concept as T2 might be a decorative tag. Based on our method, the first pair has edit distance 1 (insertion of T2) while the second pair has edit distance 2 (deletion of T2 plus insertion of T4). In other words, the first pair has a higher similarity.

- **Adjacency similarity (*SimA*).** The adjacency similarity between two data units d_1 and d_2 is the average of the similarity between d_1^p and d_2^p and the similarity between d_1^s and d_2^s, that is

$$SimA(d_1, d_2) = \big(Sim'(d_1^p, d_2^p) + Sim'(d_1^s, d_2^s)\big)/2. \quad (6)$$

When computing the similarities (Sim') between the preceding/succeeding units, only the first four features are used. The weight for adjacency feature (w_5) is proportionally distributed to other four weights.

Our alignment algorithm also needs the similarity between two data unit groups where each group is a collection of data units. We define the similarity between groups G_1 and G_2 to be the average of the similarities between every data unit in G_1 and every data unit in G_2.

4.2 Alignment Algorithm

Our data alignment algorithm is based on the assumption that attributes appear in the same order across all SRRs on the same result page, although the SRRs may contain different sets of attributes (due to missing values). This is true in general because the SRRs from the same WDB are normally generated by the same template program. Thus, we can conceptually consider the SRRs on a result page in a table format where each row represents one SRR and each cell holds a data unit (or empty if the data unit is not available). Each table column, in our work, is referred to as an *alignment group*, containing at most one data unit from each SRR. If an alignment group contains all the data units of one concept and no data unit from other concepts, we call this group *well-aligned*. The goal of alignment is to move the data units in the table so that every alignment group is well aligned, while the order of the data units within every SRR is preserved.

Our data alignment method consists of the following four steps. The detail of each step will be provided later.

Step 1: Merge text nodes. This step detects and removes decorative tags from each SRR to allow the text nodes corresponding to the same attribute (separated by decorative tags) to be merged into a single text node.

Step 2: Align text nodes. This step aligns text nodes into groups so that eventually each group contains the text nodes with the same concept (for atomic nodes) or the same set of concepts (for composite nodes).

Step 3: Split (composite) text nodes. This step aims to split the "values" in composite text nodes into individual data units. This step is carried out based on the text nodes in the same group holistically. A group whose "values" need to be split is called a *composite group*.

Step 4: Align data units. This step is to separate each composite group into multiple aligned groups with each containing the data units of the same concept.

As we discussed in Section 3.1, the Many-to-One relationship between text nodes and data units usually occurs because of the decorative tags. We need to remove them to restore the integrity of data unit. In Step 1, we use a modified method in [35] to detect the decorative tags. For every HTML tag, its statistical scores of a set of predefined features are collected across all SRRs, including the distance to its leaf descendants, the number of occurrences, and the first and last occurring positions in every SRRs, etc. Each individual feature score is normalized between 0 and 1, and all normalized feature scores are then averaged into a single score s. A tag is identified as a decorative tag if $s \geq \mathrm{P}(\mathrm{P} = 0.5$ is used in this work, following [35]). To remove decorative tags, we do the breadth-first traversal over the DOM tree of each SRR. If the traversed node is identified as a decorative tag, its immediate child nodes are moved up as the right siblings of this node, and then the node is deleted from the tree.

In Step 2, as shown in ALIGN in Fig. 4, text nodes are initially aligned into alignment groups based on their

```
ALIGN(SRRs)
1.  j ← 1;
2.  while true
      //create alignment groups
3.    for i ← 1 to number of SRRs
4.      G_j ← SRR[i][j];    //j^th element in SRR[i]
5.    if G_j is empty
6.      exit; //break the loop
7.    V ← CLUSTERING(G);
8.    if |V| > 1
        //collect all data units in groups following j
9.      S ← Ø;
10.     for x ← 1 to number of SRRs
11.       for y ← j+1 to SRR[i].length
12.         S ← SRR[x][y];
        //find cluster c least similar to following groups
13.     V[c] = min_{k=1 to |V|} (sim(V[k], S));
        //shifting
14.     for k ← 1 to |V| and k ≠ c
15.       foreach SRR[x][j] in V[k]
16.         insert NIL at position j in SRR[x];
17.   j ←j+1;        //move to next group

CLUSTERING(G)
1.  V ←all data units in G;
2.  while |V| > 1
3.    best ← 0;
4.    L ←NIL; R ←NIL;
5.    foreach A in V
6.      foreach B in V
7.        if ((A != B) and (sim(A, B) > best))
8.          best ← sim(A,B);
9.          L ←A;
10.         R ←B;
11.   If best > T
12.     remove L from V;
13.     remove R from V;
14.     add L ∪ R to V;
15.   else break loop;
16. return V;
```

Fig. 4. Alignment algorithm.

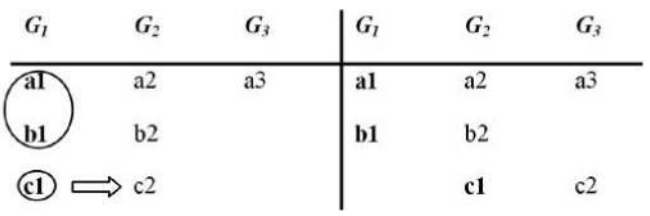


Fig. 5. An example illustrating Step 2 of the alignment algorithm.

positions within SRRs so that group G_j contains the jth text node from each SRR (lines 3-4). Since a particular SRR may have no value(s) for certain attribute(s) (e.g., a book would not have *discount price* if it is not on sale), G_j may contain the elements of different concepts. We apply the agglomerative clustering algorithm [17] to cluster the text nodes inside this group (line 7 and CLUSTERING). Initially, each text node forms a separate cluster of its own. We then repeatedly merge two clusters that have the highest similarity value until no two clusters have similarity above a threshold T. After clustering, we obtain a set of clusters V and each cluster contains the elements of the same concept only.

Recall that attributes are assumed to be encoded in the same order across all SRRs. Suppose an attribute A_j is missing in an SRR. Since we are initially aligned by position, the element of attribute A_k, where $k > j$ is put into group G_j. This element must have certain commonalities with some other elements of the same concept in the groups following G_j, which should be reflected by higher similarity values. Thus, if we get multiple clusters from the above step, the one having the least similarity (V[c]) with the groups following G_j should belong to attribute A_j (lines 9-13). Finally in Step 2, for each element in the clusters other than V[c], shift it (and all elements after it) to the next group. Lines 14-16 show that we achieve this by inserting an NIL element at position j in the corresponding SRRs. This process is repeated for position $j + 1$ (line 17) until all the text nodes are considered (lines 5-6).

Example 1. In Fig. 5, after initial alignment, there are three alignment groups. The first group G_1 is clustered into two clusters {{a1, b1}, {c1}}. Suppose {a1, b1} is the least similar to G_2 and G_3, we then shift c1 one position to the right. The figure on the right depicts all groups after shifting.

After the text nodes are grouped using the above procedure, we need to determine whether a group needs to be further split to obtain the actual data units (Step 3). First, we identify groups whose text nodes are not "split-able." Each such a group satisfies one of the following conditions:

1. each of its text nodes is a hyperlink (a hyperlink is assumed to already represent a single semantic unit);
2. the texts of all the nodes in the group are the same;
3. all the nodes in the group have the same nonstring data type; and
4. the group is not a merged group from Step 1.

Next, we try to split each group that does not satisfy any of the above conditions. To do the splitting correctly, we need to identify the right *separators*. We observe that the same separator is usually used to separate the data units of a given pair of attributes across all SRRs retrieved from the same WDB although it is possible that different separators are used for data units of different attribute pairs. Based on this observation, in this step, we first scan the text strings of every text node in the group and select the symbol(s) (nonletter, nondigit, noncurrency, nonparenthesis, and nonhyphen) that occur in most strings in consistent positions. Second, for each text node in the group, its text is split into several small pieces using the separator(s), each of which becomes a real data unit. As an example, in "Springer-Verlag/1999/0387984135 /0.06667," "/" is the separator and it splits the composite text node into four data units.

The data units in a composite group are not always aligned after splitting because some attributes may have missing values in the composite text node. Our solution is to apply the same alignment algorithm in Step 2 here, i.e., initially align based on each data unit's natural position and then apply the clustering-based shifting method. The only difference is that, in Step 4, since all data units to be aligned are split from the same composite text node, they share the same presentation style and tag path. Thus, in this case, these two features are not used for calculating similarity for aligning data units. Their feature weights are proportionally distributed to the three features used.

DeLa [30] also detects the separators inside composite text nodes and uses them to separate the data units. However, in DeLa, the separated pieces are simply aligned by their natural order and missing attribute values are not considered.

5 Assigning Labels

5.1 Local versus Integrated Interface Schemas

For a WDB, its search interface often contains some attributes of the underlying data. We denote a LIS as $S_i = \{A_1, A_2, \ldots, A_k\}$, where each A_j is an attribute. When a query is submitted against the search interface, the entities in the returned results also have a certain *hidden* schema, denoted as $S_e = \{a_1, a_2, \ldots, a_n\}$, where each a_j $(j = 1 \ldots n)$ is an attribute to be discovered. The schema of the retrieved data and the LIS usually share a significant number of attributes [29]. This observation provides the basis for some of our basic annotators (see Section 5.2). If an attribute a_t in the search results has a matched attribute A_t in the LIS, all the data units identified with a_t can be labeled by the name of A_t.

However, it is quite often that S_e is not entirely contained in S_i because some attributes of the underlying database are not suitable or needed for specifying query conditions as determined by the developer of the WDB, and these attributes would not be included in S_i. This phenomenon raises a problem called *local interface schema inadequacy problem*. Specifically, it is possible that a *hidden* attribute discovered in the search result schema S_e does not have a matching attribute A_t in the LIS S_i. In this case, there will be no label in the search interface that can be assigned to the discovered data units of this attribute.

Another potential problem associated with using LISs for annotation is the *inconsistent label problem*, i.e., different labels are assigned to semantically identical data units returned from different WDBs because different LISs may give different names to the same attribute. This can cause problem

when using the annotated data collected from different WDBs, e.g., for data integration applications.

In our approach, for each used domain, we use WISE-Integrator [14]) to build an IIS over multiple WDBs in that domain. The generated IIS combines all the attributes of the LISs. For matched attributes from different LISs, their values in the local interfaces (e.g., values in selection list) are combined as the values of the integrated global attribute [14]. Each global attribute has a unique global name and an attribute-mapping table is created to establish the mapping between the name of each LIS attribute and its corresponding name in the IIS. In this paper, for attribute A in an LIS, we use $gn(A)$ to denote the name of A's corresponding attribute (i.e., the global attribute) in the IIS.

For each WDB in a given domain, our annotation method uses both the LIS of the WDB and the IIS of the domain to annotate the retrieved data from this WDB. Using IISs has two major advantages. First, it has the potential to increase the annotation recall. Since the IIS contains the attributes in all the LISs, it has a better chance that an attribute discovered from the returned results has a matching attribute in the IIS even though it has no matching attribute in the LIS. Second, when an annotator discovers a label for a group of data units, the label will be replaced by its corresponding global attribute name (if any) in the IIS by looking up the attribute-mapping table so that the data units of the same concept across different WDBs will have the same label.

We should point out that even though using the IIS can significantly alleviate both the *local interface schema inadequacy problem* and the *inconsistent label problem*, it cannot solve them completely. For the first problem, it is still possible that some attributes of the underlying entities do not appear in any local interface, and as a result, such attributes will not appear in the IIS. As to the second problem, for example, in Fig. 1, "$17.50" is annotated by "*Our Price*," but at another site a price may be displayed as "You Pay: $50.50" (i.e., "$50.50" is annotated by "*You Pay*"). If one or more of these annotations are not local attribute names in the attribute-mapping table for this domain, then using the IIS cannot solve the problem and new techniques are needed.

5.2 Basic Annotators

In a returned result page containing multiple SRRs, the data units corresponding to the same concept (attribute) often share special common features. And such common features are usually associated with the data units on the result page in certain patterns. Based on this observation, we define six basic annotators to label data units, with each of them considering a special type of patterns/features. Four of these annotators (i.e., table annotator, query-based annotator, in-text prefix/suffix annotator, and common knowledge annotator) are similar to the annotation heuristics used by DeLa [30] but we have different implementations for three of them (i.e., table annotator, query-based annotator, and common knowledge annotator). Details of the differences and the advantages of our implementations for these three annotators will be provided when these annotators are introduced below.

Manufacture	Model	Class	Year	City	State	Price
HONDA	accord LX	4 DOOR	1998	playa del rey	CA	$11,500
HONDA	ACCORD LX	4 DOOR	1994	Spokane	WA	$ 7,500
HONDA	Accord Lx	4 DOOR	1997	Winona ake	ID	$ 8,700
HONDA	Accord LX	4 DOOR	1994	Cave Creek	AZ	$ 5,999
HONDA	Accord	4 DOOR	1999	Pomona	CA	$17,500

Fig. 6. SRRs in table format.

5.2.1 Table Annotator (TA)

Many WDBs use a table to organize the returned SRRs. In the table, each row represents an SRR. The table header, which indicates the meaning of each column, is usually located at the top of the table. Fig. 6 shows an example of SRRs presented in a table format. Usually, the data units of the same concepts are well aligned with its corresponding column header. This special feature of the table layout can be utilized to annotate the SRRs.

Since the physical position information of each data unit is obtained during SRR extraction, we can utilize the information to associate each data unit with its corresponding header. Our Table Annotator works as follows: First, it identifies all the column headers of the table. Second, for each SRR, it takes a data unit in a cell and selects the column header whose area (determined by coordinates) has the maximum vertical overlap (i.e., based on the x-axis) with the cell. This unit is then assigned with this column header and labeled by the header text A (actually by its corresponding global name $gn(A)$ if $gn(A)$ exists). The remaining data units are processed similarly. In case that the table header is not provided or is not successfully extracted by ViNTs [34], the Table Annotator will not be applied.

DeLa [30] also searches for table header texts as the possible labels for the data units listed in table format. However, DeLa only relies on HTML tag <TH> and <THEAD> for this purpose. But many HTML tables do not use <TH> or <THEAD> to encode their headers, which limits the applicability of DeLa's approach. In the test data set we collected, 11 WDBs have SRRs in table format, but only three use <TH> or <THEAD>. In contrast, our table annotator does not have this limitation.

5.2.2 Query-Based Annotator (QA)

The basic idea of this annotator is that the returned SRRs from a WDB are always related to the specified query. Specifically, the query terms entered in the search attributes on the local search interface of the WDB will most likely appear in some retrieved SRRs. For example, in Fig. 1, query term "machine" is submitted through the *Title* field on the search interface of the WDB and all three titles of the returned SRRs contain this query term. Thus, we can use the name of search field *Title* to annotate the title values of these SRRs. In general, query terms against an attribute may be entered to a textbox or chosen from a selection list on the local search interface.

Our Query-based Annotator works as follows: Given a query with a set of query terms submitted against an attribute A on the local search interface, first find the group that has the largest total occurrences of these query terms and then assign $gn(A)$ as the label to the group.

As mentioned in Section 5.1, the LIS of a WDB usually does not have all the attributes of the underlying database.

As a result, the query-based annotator by itself cannot completely annotate the SRRs.

DeLa [30] also uses query terms to match the data unit texts and use the name of the queried form element as the label. However, DeLa uses only local schema element names, not element names in the IIS.

5.2.3 Schema Value Annotator (SA)

Many attributes on a search interface have predefined values on the interface. For example, the attribute *Publishers* may have a set of predefined values (i.e., publishers) in its selection list. More attributes in the IIS tend to have predefined values and these attributes are likely to have more such values than those in LISs, because when attributes from multiple interfaces are integrated, their values are also combined [14]. Our schema value annotator utilizes the combined value set to perform annotation.

Given a group of data units $G_i = \{d_1, \ldots, d_n\}$, the schema value annotator is to discover the best matched attribute to the group from the IIS. Let A_j be an attribute containing a list of values $\{v_1, \ldots, v_m\}$ in the IIS. For each data unit d_k, this annotator first computes the *Cosine similarities* between d_k and all values in A_j to find the value (say v_t) with the highest similarity. Then, the data fusion function *CombMNZ* [19] is applied to the similarities for all the data units. More specifically, the annotator sums up the similarities and multiplies the sum by the number of nonzero similarities. This final value is treated as the matching score between G_i and A_j.

The schema value annotator first identifies the attribute A_j that has the highest matching score among all attributes and then uses $gn(A_j)$ to annotate the group G_i. Note that multiplying the above sum by the number of nonzero similarities is to give preference to attributes that have more matches (i.e., having nonzero similarities) over those that have fewer matches. This is found to be very effective in improving the retrieval effectiveness of combination systems in information retrieval [4].

5.2.4 Frequency-Based Annotator (FA)

In Fig. 1, "Our Price" appears in the three records and the followed price values are all different in these records. In other words, the adjacent units have different occurrence frequencies. As argued in [1], the data units with the higher frequency are likely to be attribute names, as part of the template program for generating records, while the data units with the lower frequency most probably come from databases as embedded values. Following this argument, "Our Price" can be recognized as the label of the value immediately following it. The phenomenon described in this example is widely observable on result pages returned by many WDBs and our frequency-based annotator is designed to exploit this phenomenon.

Consider a group G_i whose data units have a lower frequency. The frequency-based annotator intends to find common preceding units shared by all the data units of the group G_i. This can be easily conducted by following their preceding chains recursively until the encountered data units are different. All found preceding units are concatenated to form the label for the group G_i.

Example 2. In Fig. 1, during the data alignment step, a group is formed for {"$17.50," "$18.95," "$20.50"}. Clearly the data units in this group have different values. These values share the same preceding unit "Our Price," which occurs in all SRRs. Furthermore, "Our Price" does not have preceding data units because it is the first unit in this line. Therefore, the frequency-based annotator will assign label "Our Price" to this group.

5.2.5 In-Text Prefix/Suffix Annotator (IA)

In some cases, a piece of data is encoded with its label to form a single unit without any obvious separator between the label and the value, but it contains both the label and the value. Such nodes may occur in all or multiple SRRs. After data alignment, all such nodes would be aligned together to form a group. For example, in Fig. 1, after alignment, one group may contain three data units, {"You Save $9.50," "You Save $11.04," "You Save $4.45"}.

The in-text prefix/suffix annotator checks whether all data units in the aligned group share the same prefix or suffix. If the same prefix is confirmed and it is not a delimiter, then it is removed from all the data units in the group and is used as the label to annotate values following it. If the same suffix is identified and if the number of data units having the same suffix match the number of data units inside the next group, the suffix is used to annotate the data units inside the next group. In the above example, the label "You save" will be assigned to the group of prices. Any group whose data unit texts are completely identical is not considered by this annotator.

5.2.6 Common Knowledge Annotator (CA)

Some data units on the result page are self-explanatory because of the common knowledge shared by human beings. For example, "**in stock**" and "**out of stock**" occur in many SRRs from e-commerce sites. Human users understand that it is about the availability of the product because this is common knowledge. So our common knowledge annotator tries to exploit this situation by using some predefined common concepts.

Each common concept contains a label and a set of patterns or values. For example, a country concept has a label "country" and a set of values such as "U.S.A.," "Canada," and so on. As another example, the e-mail address (assume all lower cases) concept has the pattern $[\text{a-z}0-9.\_\%+-]+@([\text{a-z}0-9-]+\.)+[\text{a-z}]\{2,4\}$. Given a group of data units from the alignment step, if all the data units match the pattern or value of a concept, the label of this concept is assigned to the data units of this group.

DeLa [30] also uses some conventions to annotate data units. However, it only considers certain patterns. Our Common knowledge annotator considers both patterns and certain value sets such as the set of countries.

It should be pointed out that our common concepts are different from the ontologies that are widely used in some works in Semantic Web (e.g., [6], [11], [12], [16], [26]). First, our common concepts are domain independent. Second, they can be obtained from existing information resources with little additional human effort.

TABLE 1
Applicabilities and Success Rates of Annotators

	Applicability	Success Rate
Table Annotator	6%	1.0
Query-based Annotator	35%	0.95
Schema value Annotator	14%	0.5
Frequency-based Annotator	41%	0.86
In-text prefix/suffix Annotator	7%	0.85
Common knowledge Annotator	25%	0.84

5.3 Combining Annotators

Our analysis indicates that no single annotator is capable of fully labeling all the data units on different result pages. The *applicability* of an annotator is the percentage of the attributes to which the annotator can be applied. For example, if out of 10 attributes, four appear in tables, then the applicability of the table annotator is 40 percent. Table 1 shows the average applicability of each basic annotator across all testing domains in our data set. This indicates that the results of different basic annotators should be combined in order to annotate a higher percentage of data units. Moreover, different annotators may produce different labels for a given group of data units. Therefore, we need a method to select the most suitable one for the group.

Our annotators are fairly independent from each other since each exploits an independent feature. Based on this characteristic, we employ a simple probabilistic method to combine different annotators. For a given annotator L, let $P(L)$ be the probability that L is correct in identifying a correct label for a group of data units when L is applicable. $P(L)$ is essentially the *success rate* of L. Specifically, suppose L is applicable to N cases and among these cases M are annotated correctly, then $P(L) = M/N$. If k independent annotators L_i, $i = 1, \ldots, k$, identify the same label for a group of data units, then the combined probability that at least one of the annotators is correct is

$$1 - \prod_{i=1}^{k} (1 - P(L_i)). \quad (7)$$

To obtain the success rate of an annotator, we use the annotator to annotate every result page in a training data set (DS1 in Section 7.1). The training result is listed in Table 1. It can be seen that the table annotator is 100 percent correct when applicable. The query-based annotator also has very high success rate while the schema value annotator is the least accurate.

An important issue DeLa did not address is what if multiple heuristics can be applied to a data unit. In our solution, if multiple labels are predicted for a group of data units by different annotators, we compute the combined probability for each label based on the annotators that identified the label, and select the label with the largest combined probability.

One advantage of this model is its high flexibility in the sense that when an existing annotator is modified or a new annotator is added in, all we need is to obtain the applicability and success rate of this new/revised annotator while keeping all remaining annotators unchanged. We also note that no domain-specific training is needed to obtain the applicability and success rate of each annotator.

6 Annotation Wrapper

Once the data units on a result page have been annotated, we use these annotated data units to construct an annotation wrapper for the WDB so that the new SRRs retrieved from the same WDB can be annotated using this wrapper quickly without reapplying the entire annotation process. We now describe our method for constructing such a wrapper below.

Each annotated group of data units corresponds to an attribute in the SRRs. The annotation wrapper is a description of the annotation rules for all the attributes on the result page. After the data unit groups are annotated, they are organized based on the order of its data units in the original SRRs. Consider the ith group G_i. Every SRR has a tag-node sequence like Fig. 1b that consists of only HTML tag names and texts. For each data unit in G_i, we scan the sequence both backward and forward to obtain the prefix and suffix of the data unit. The scan stops when an encountered unit is a valid data unit with a meaningful label assigned. Then, we compare the prefixes of all the data units in G_i to obtain the common prefix shared by these data units. Similarly, the common suffix is obtained by comparing all the suffixes of these data units. For example, the data unit for book title in Fig. 1b has "<FORM><A>" as its prefix and "</A>
" as its suffix. If a data unit is generated by splitting from a composite text node, then its prefix and suffix are the same as those of its parent data unit. This wrapper is similar to the LR wrapper in [18]. Here, we use *prefix* as the left delimiter, and *suffix* as the right delimiter to identify data units. However, the LR wrapper has difficulties to extract data units packed inside composite text nodes due to the fact that there is no HTML tag within a text node. To overcome this limitation, besides the prefix and suffix, we also record the separators used for splitting the composite text node as well as its position index in the split unit vector. Thus, the annotation rule for each attribute consists of five components, expressed as: $attribute_i = <label_i, prefix_i, suffix_i, separators_i, unitindex_i>$. The annotation wrapper for the site is simply a collection of the annotation rules for all the attributes identified on the result page with order corresponding to the ordered data unit groups.

To use the wrapper to annotate a new result page, for each data unit in an SRR, the annotation rules are applied on it one by one based on the order they appear in the wrapper. If this data unit has the same prefix and suffix as specified in the rule, the rule is matched and the unit is labeled with the given label in the rule. If the separators are specified, they are used to split the unit, and $label_i$ is assigned to the unit at the position $unitindex_i$.

TABLE 2
Performance of Data Alignment

Domain	Precision	Recall
Auto	98.6%	98.6%
Book	98.4%	97.3%
Electronics	100%	100%
Game	99.6%	99.6%
Job	95.2%	100%
Movie	98.7%	98.0%
Music	99.0%	99.1%
Overall avg.	98.5%	98.9%

TABLE 3
Performance of Annotation

Domain	Precision	Recall
Auto	97.7%	97.8%
Book	97.4%	96.4%
Electronics	98.3%	98.3%
Game	96.1%	96.1%
Job	95.2%	100%
Movie	97.2%	96.7%
Music	97.6%	97.7%
Overall avg.	97.1%	97.6%

7 Experiments

7.1 Data Sets and Performance Measure

Our experiments are based on 112 WDBs selected from seven domains: *book*, *movie*, *music*, *game*, *job*, *electronics*, and *auto*. For each WDB, its LIS is constructed automatically using WISE-iExtractor [15]. For each domain, WISE-Integrator [14] is used to build the IIS automatically. These collected WDBs are randomly divided into two disjoint groups. The first group contains 22 WDBs and is used for training, and the second group has 90 WDBs and is used for testing. Data set **DS1** is formed by obtaining one sample result page from each training site. Two testing data sets **DS2** and **DS3** are generated by collecting two sample result pages from each testing site using different queries. We note that we largely recollected the result pages from WDBs used in our previous study [22]. We have noticed that result pages have become more complex in general as web developers try to make their pages fancier.

Each testing data set contains one page from each WDB. Some general terms are manually selected as the query keywords to obtain the sample result pages. The query terms are selected in such a way that they yield result pages with many SRRs from all WDBs of the same domain, for example, "Java" for Title, "James" for Author, etc. The query terms and the form element each query is submitted to are also stored together with each LIS for use by the Query-based Annotator.

Data set DS1 is used for learning the weights of the data unit features and clustering threshold T in the alignment step (See Section 4), and determining the success rate of each basic annotator (See Section 5). For each result page in this data set, the data units are manually extracted, aligned in groups, and assigned labels by a human expert. We use a genetic algorithm based method [10] to obtain the best combination of feature weights and clustering threshold T that leads to the best performance over the training data set.

DS2 is used to test the performance of our alignment and annotation methods based on the parameter values and statistics obtained from DS1. At the same time, the annotation wrapper for each site will be generated. DS3 is used to test the quality of the generated wrappers. The correctness of data unit extraction, alignment, and annotation is again manually verified by the same human expert for performance evaluation purpose.

We adopt the *precision* and *recall* measures from information retrieval to evaluate the performance of our methods. For alignment, the *precision* is defined as the percentage of the correctly aligned data units over all the aligned units by the system; *recall* is the percentage of the data units that are correctly aligned by the system over all manually aligned data units by the expert. A result data unit is counted as "incorrect" if it is mistakenly extracted (e.g., failed to be split from composite text node). For annotation, the precision is the percentage of the correctly annotated units over all the data units annotated by the system and the recall is the percentage of the data units correctly annotated by the system over all the manually annotated units. A data unit is said to be correctly annotated if its system-assigned label has the same meaning as its manually assigned label.

7.2 Experimental Results

The optimal feature weights obtained through our genetic training method (See Section 4) over DS1 are {*0.64, 0.81, 1.0, 0.48, 0.56*} for *SimC*, *SimP*, *SimD*, *SimT*, and *SimA*, respectively, and 0.59 for clustering threshold T. The average alignment precision and recall are converged at about 97 percent. The learning result shows that the data type and the presentation style are the most important features in our alignment method. Then, we apply our annotation method on DS1 to determine the success rate of each annotator (see Section 5.3).

Table 2 shows the performance of our data alignment algorithm for all 90 pages in DS2. The precision and recall for every domain are above 95 percent, and the average precision and recall across all domains are above 98 percent. The performance is consistent with that obtained over the training set. The errors usually happen in the following cases. First, some composite text nodes failed to be split into correct data units when no explicit separators can be identified. For example, the data units in some composite text nodes are separated by blank spaces created by consecutive HTML entities like " " or some formatting HTML tags such as <SPAN>. Second, the data units of the same attribute across different SRRs may sometimes vary a lot in terms of appearance or layout. For example, the promotion price information often has color or font type different from that for the regular price information. Note that in this case, such two price data units have low similarity on content, presentation style, and the tag path. Even though they share the same data type, the overall

TABLE 4
Performance of Annotation with Wrapper

Domain	Precision	Recall
Auto	94.9%	91.0%
Book	93.7%	92.1%
Electronics	95.1%	94.5%
Job	93.7%	92.0%
Movie	93.0%	92.1%
Music	95.3%	94.7%
Game	95.6%	93.8%
Overall avg.	94.4%	93.0%

similarity between them would still be low. Finally, the decorative tag detection (Step 1 of the alignment algorithm) is not perfect (accuracy about 90 percent), which results in some tags to be falsely detected as decorative tags, leading to incorrect merging of the values of different attributes. We will address these issues in the future.

Table 3 lists the results of our annotation performance over DS2. We can see that the overall precision and recall are very high, which shows that our annotation method is very effective. Moreover, high precision and recall are achieved for every domain, which indicates that our annotation method is domain independent. We notice that the overall recall is a little bit higher than precision, mostly because some data units are not correctly separated in the alignment step. We also found that in a few cases some texts are not assigned labels by any of our basic annotators. One reason is that some texts are for cosmetic or navigating purposes. These texts do not represent any attributes of the real-world entity and they are not the labels of any data unit, which belong to our One-To-Nothing relationship type. It is also possible that some of these texts are indeed data units but none of our current basic annotators are applicable to them.

For each webpage in DS2, once its SRRs have been annotated, an annotation wrapper is built, and it is applied to the corresponding page in DS3. In terms of processing speed, the time needed to annotate one result page drops from 10-20 seconds without using wrapper to 1-2 seconds using wrapper depending on the complexity of the page. The reason is that wrapper based approach directly extracts all data units specified by the tag path(s) for each attribute and assigns the label specified in the rule to those data units. In contrast, the nonwrapper-based approach needs to go through some time-consuming steps such as result page rendering, data unit similarity matrix computation, etc., for each result page.

In terms of precision and recall, the wrapper-based approach reduces the precision by 2.7 percentage points and recall by 4.6 percentage points, as can be seen from the results in Tables 3 and 4. The reason is that our wrapper only used tag path for text node extraction and alignment, and the composite text node splitting is solely based on the data unit position. In the future, we will investigate how to incorporate other features in the wrapper to increase the performance. Another reason is that in this experiment, we used only one page for each site in DS2 to build the wrapper. As a result, it is possible that some attributes that appear on the page in DS3 do not appear on the training page in DS2 (so they do not appear in the wrapper expression). For example, some WDBs only allow queries that use only one attribute at a time (these attributes are called *exclusive attributes* in [15]). In this case, if the query term is based on *Title*, then the data units for attribute *Author* may not be correctly identified and annotated because the query-based annotator is the main technique used for attributes that have a textbox on the search interface. One possible solution to remedy this problem is to combine multiple result pages based on different queries to build a more robust wrapper.

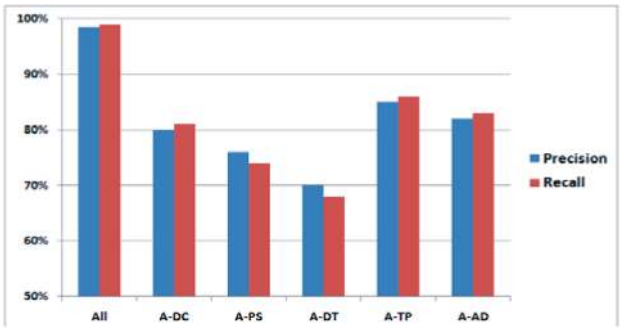


Fig. 7. Evaluation of alignment features.

We also conducted experiments to evaluate the significance of each feature on the performance of our alignment algorithm. For this purpose, we compare the performance when a feature is used with that when it is not used. Each time one feature is selected not to be used, and its weight is proportionally distributed to other features based on the ratios of their weights to the total weight of the used features. The alignment process then uses the new parameters to run on DS2. Fig. 7 shows the results. It can be seen that when any one of these features is not used, both the precision and recall decrease, indicating that all the features in our approach are valid and useful. We can also see that the data type and the presentation style are the most important features because when they are not used, the precision and recall drop the most (around 28 and 23 percentage points for precision, and 31 and 25 percentage points for recall, respectively). This result is consistent with our training result where the data type and the presentation style have the highest feature weights. The adjacency and tag path feature

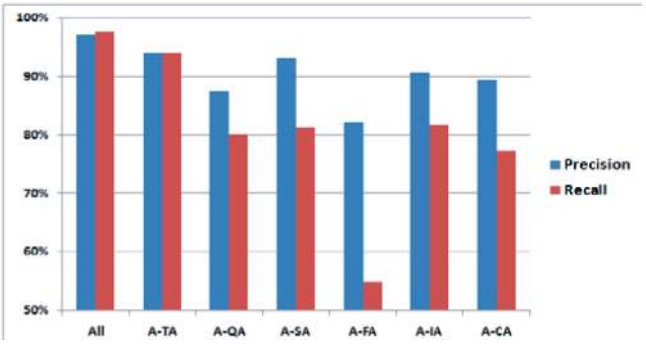


Fig. 8. Evaluation of basic annotators.

TABLE 5
Performance Using Local Interface Schema

DOMAIN	PRECISION	RECALL
AUTO	98.3%	98.3%
BOOK	96.1%	85.6%
ELECTRONICS	97.5%	91.7%
GAME	95.9%	92.2%
JOB	95.3%	95.1%
MOVIE	96.8%	90.8%
MUSIC	95.2%	83.9%
OVERALL AVG.	96.4%	91.1%

are less significant comparatively, but without either of them, the precision and recall drop more than 15 percentage points.

We use a similar method as the above to evaluate the significance of each basic annotator. Each time, one annotator is removed and the remaining annotators are used to annotate the pages in DS2. Fig. 8 shows the results. It shows that omitting any annotator causes both precision and recall to drop, i.e., every annotator contributes positively to the overall performance. Among the six annotators considered, the query-based annotator and the frequency-based annotator are the most significant. Another observation is that when an annotator is removed, the recall decreases more dramatically than precision. This indicates that each of our annotators is fairly independent in terms of describing the attributes. Each annotator describes one aspect of the attribute which, to a large extent, is not applicable to other annotators.

Finally, we conducted experiments to study the effect of using LISs versus using the IIS in annotation. We run the annotation process on DS2 again but this time, instead of using the IIS built for each domain, we use the LIS of each WDB. The results are shown in Table 5. By comparing the results in Table 5 with those in Table 3 where the IIS is used, we can see that using the LIS has a relatively small impact on precision, but significant effect on recall (the overall average recall is reduced by more than 6 percentage points) because of the local interface schema inadequacy problem as described in Section 5.1. And it also shows that using the integrated interface schema can indeed increase the annotation performance.

As reported in [30], only three domains (Book, Job, and Car) were used to evaluate DeLa and for each domain, only nine sites were selected. The overall precision and recall of DeLa's annotation method using this data set are around 80 percent. In contrast, the precision and recall of our annotation method for these three domains are well above 95 percent (see Table 3), although different sets of sites for these domains are used in the two works.

8 CONCLUSION

In this paper, we studied the data annotation problem and proposed a multiannotator approach to automatically constructing an annotation wrapper for annotating the search result records retrieved from any given web database. This approach consists of six basic annotators and a probabilistic method to combine the basic annotators. Each of these annotators exploits one type of features for annotation and our experimental results show that each of the annotators is useful and they together are capable of generating high-quality annotation. A special feature of our method is that, when annotating the results retrieved from a web database, it utilizes both the LIS of the web database and the IIS of multiple web databases in the same domain. We also explained how the use of the IIS can help alleviate the local interface schema inadequacy problem and the inconsistent label problem.

In this paper, we also studied the automatic data alignment problem. Accurate alignment is critical to achieving holistic and accurate annotation. Our method is a clustering based shifting method utilizing richer yet automatically obtainable features. This method is capable of handling a variety of relationships between HTML text nodes and data units, including one-to-one, one-to-many, many-to-one, and one-to-nothing. Our experimental results show that the precision and recall of this method are both above 98 percent. There is still room for improvement in several areas as mentioned in Section 7.2. For example, we need to enhance our method to split composite text node when there are no explicit separators. We would also like to try using different machine learning techniques and using more sample pages from each training site to obtain the feature weights so that we can identify the best technique to the data alignment problem.

ACKNOWLEDGMENTS

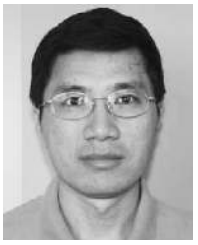
This work is supported in part by the following US National Science Foundation (NSF) grants: IIS-0414981, IIS-0414939, CNS-0454298, and CNS-0958501. The authors would also like to express their gratitude to the anonymous reviewers for providing very constructive suggestions to improve the manuscript.

REFERENCES

[1] A. Arasu and H. Garcia-Molina, "Extracting Structured Data from Web Pages," *Proc. SIGMOD Int'l Conf. Management of Data,* 2003.
[2] L. Arlotta, V. Crescenzi, G. Mecca, and P. Merialdo, "Automatic Annotation of Data Extracted from Large Web Sites," *Proc. Sixth Int'l Workshop the Web and Databases (WebDB),* 2003.
[3] P. Chan and S. Stolfo, "Experiments on Multistrategy Learning by Meta-Learning," *Proc. Second Int'l Conf. Information and Knowledge Management (CIKM),* 1993.
[4] W. Bruce Croft, "Combining Approaches for Information Retrieval," *Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval,* Kluwer Academic, 2000.
[5] V. Crescenzi, G. Mecca, and P. Merialdo, "RoadRUNNER: Towards Automatic Data Extraction from Large Web Sites," *Proc. Very Large Data Bases (VLDB) Conf.,* 2001.
[6] S. Dill et al., "SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation," *Proc. 12th Int'l Conf. World Wide Web (WWW) Conf.,* 2003.
[7] H. Elmeleegy, J. Madhavan, and A. Halevy, "Harvesting Relational Tables from Lists on the Web," *Proc. Very Large Databases (VLDB) Conf.,* 2009.
[8] D. Embley, D. Campbell, Y. Jiang, S. Liddle, D. Lonsdale, Y. Ng, and R. Smith, "Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages," *Data and Knowledge Eng.,* vol. 31, no. 3, pp. 227-251, 1999.
[9] D. Freitag, "Multistrategy Learning for Information Extraction," *Proc. 15th Int'l Conf. Machine Learning (ICML),* 1998.
[10] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, 1989.

[11] S. Handschuh, S. Staab, and R. Volz, "On Deep Annotation," *Proc. 12th Int'l Conf. World Wide Web (WWW)*, 2003.
[12] S. Handschuh and S. Staab, "Authoring and Annotation of Web Pages in CREAM," *Proc. 11th Int'l Conf. World Wide Web (WWW)*, 2003.
[13] B. He and K. Chang, "Statistical Schema Matching Across Web Query Interfaces," *Proc. SIGMOD Int'l Conf. Management of Data*, 2003.
[14] H. He, W. Meng, C. Yu, and Z. Wu, "Automatic Integration of Web Search Interfaces with WISE-Integrator," *VLDB J.*, vol. 13, no. 3, pp. 256-273, Sept. 2004.
[15] H. He, W. Meng, C. Yu, and Z. Wu, "Constructing Interface Schemas for Search Interfaces of Web Databases," *Proc. Web Information Systems Eng. (WISE) Conf.*, 2005.
[16] J. Heflin and J. Hendler, "Searching the Web with SHOE," *Proc. AAAI Workshop*, 2000.
[17] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, 1990.
[18] N. Krushmerick, D. Weld, and R. Doorenbos, "Wrapper Induction for Information Extraction," *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI)*, 1997.
[19] J. Lee, "Analyses of Multiple Evidence Combination," *Proc. 20th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, 1997.
[20] L. Liu, C. Pu, and W. Han, "XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources," *Proc. IEEE 16th Int'l Conf. Data Eng. (ICDE)*, 2001.
[21] W. Liu, X. Meng, and W. Meng, "ViDE: A Vision-Based Approach for Deep Web Data Extraction," *IEEE Trans. Knowledge and Data Eng.*, vol. 22, no. 3, pp. 447-460, Mar. 2010.
[22] Y. Lu, H. He, H. Zhao, W. Meng, and C. Yu, "Annotating Structured Data of the Deep Web," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE)*, 2007.
[23] J. Madhavan, D. Ko, L. Lot, V. Ganapathy, A. Rasmussen, and A.Y. Halevy, "Google's Deep Web Crawl," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1241-1252, 2008.
[24] W. Meng, C. Yu, and K. Liu, "Building Efficient and Effective Metasearch Engines," *ACM Computing Surveys*, vol. 34, no. 1, pp. 48-89, 2002.
[25] S. Mukherjee, I.V. Ramakrishnan, and A. Singh, "Bootstrapping Semantic Annotation for Content-Rich HTML Documents," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, 2005.
[26] B. Popov, A. Kiryakov, A. Kirilov, D. Manov, D. Ognyanoff, and M. Goranov, "KIM - Semantic Annotation Platform," *Proc. Int'l Semantic Web Conf. (ISWC)*, 2003.
[27] G. Salton and M. McGill, *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.
[28] W. Su, J. Wang, and F.H. Lochovsky, "ODE: Ontology-Assisted Data Extraction," *ACM Trans. Database Systems*, vol. 34, no. 2, article 12, June 2009.
[29] J. Wang, J. Wen, F. Lochovsky, and W. Ma, "Instance-Based Schema Matching for Web Databases by Domain-Specific Query Probing," *Proc. Very Large Databases (VLDB) Conf.*, 2004.
[30] J. Wang and F.H. Lochovsky, "Data Extraction and Label Assignment for Web Databases," *Proc. 12th Int'l Conf. World Wide Web (WWW)*, 2003.
[31] Z. Wu et al., "Towards Automatic Incorporation of Search Engines into a Large-Scale Metasearch Engine," *Proc. IEEE/WIC Int'l Conf. Web Intelligence (WI '03)*, 2003.
[32] O. Zamir and O. Etzioni, "Web Document Clustering: A Feasibility Demonstration," *Proc. ACM 21st Int'l SIGIR Conf. Research Information Retrieval*, 1998.
[33] Y. Zhai and B. Liu, "Web Data Extraction Based on Partial Tree Alignment," *Proc. 14th Int'l Conf. World Wide Web (WWW '05)*, 2005.
[34] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu, "Fully Automatic Wrapper Generation for Search Engines," *Proc. Int'l Conf. World Wide Web (WWW)*, 2005.
[35] H. Zhao, W. Meng, and C. Yu, "Mining Templates form Search Result Records of Search Engines," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, 2007.
[36] J. Zhu, Z. Nie, J. Wen, B. Zhang, and W.-Y. Ma, "Simultaneous Record Detection and Attribute Labeling in Web Data Extraction," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, 2006.

Yiyao Lu received the PhD degree in computer science from the State University of New York at Binghamton in 2011. He currently works for Bloomberg. His research interests include web information systems, metasearch engine, and information extraction.



Hai He received the PhD degree in computer science from State University of New York at Binghamton in 2005. He currently works for Morningstar. His research interests include web database integration systems, search interface extraction and integration.

Hongkun Zhao received the PhD degree in computer science from State University of New York at Binghamton in 2007. He currently works for Bloomberg. His research interests include metasearch engine, web information extraction, and automatic wrapper generation.

Weiyi Meng received the BS degree in mathematics from Sichuan University, China, in 1982, and the MS and PhD degrees in computer science from the University of Illinois at Chicago, in 1988 and 1992, respectively. He is currently a professor in the Department of Computer Science at the State University of New York at Binghamton. His research interests include web-based information retrieval, metasearch engines, and web database integration. He is a coauthor of three books *Principles of Database Query Processing for Advanced Applications*, *Advanced Metasearch Engine Technology*, and *Deep Web Query Interface Understanding and Integration*. He has published more than 120 technical papers. He is a member of the IEEE.

Clement Yu received the BS degree in applied mathematics from Columbia University in 1970 and the PhD degree in computer science from Cornell University in 1973. He is a professor in the Department of Computer Science at the University of Illinois at Chicago. His areas of interest include search engines and multimedia retrieval. He has published in various journals such as the *IEEE Transactions on Knowledge and Data Engineering*, *ACM Transactions on Database Systems*, and *Journal of the ACM* and in various conferences such as VLDB, ACM SIGMOD, ACM SIGIR, and ACM Multimedia. He previously served as chairman of the ACM Special Interest Group on Information Retrieval and as a member of the advisory committee to the US National Science Foundation (NSF). He was/is a member of the editorial board of the *IEEE Transactions on Knowledge and Data Engineering*, *the International Journal of Software Engineering and Knowledge Engineering*, *Distributed and Parallel Databases*, and *World Wide Web Journal*. He is a senior member of the IEEE.