Annotating Structured Data of the Deep Web

Yiyao Lu, Hai He, Hongkun Zhao, Weiyi Meng,
State University of New York at Binghamton
Binghamton, NY, 13902, U.S.A.
{ylu0, haihe, hkzhao, meng}@cs.binghamton.edu

Clement Yu
University of Illinois at Chicago
Chicago, IL 60607, U.S.A.
yu@cs.uic.edu

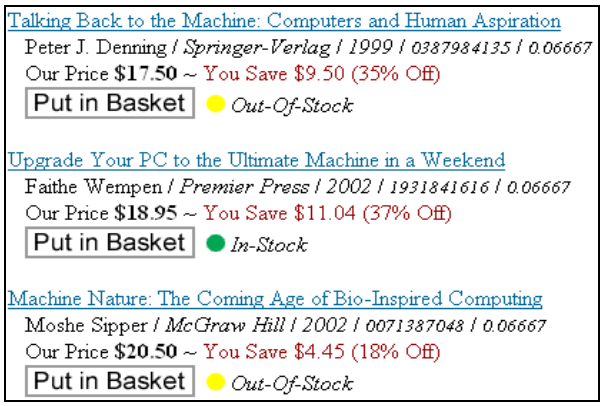
Abstract

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 comparison shopping, they need to be extracted out and assigned meaningful labels. In this paper, we present a multi-annotator approach that first aligns the data units into different groups such that the data in the same group have the same semantics. Then for each group, we annotate it from different aspects and aggregate the different annotations to predict a final annotation label. An annotation wrapper for the search site is automatically constructed and can be used to annotate new result pages from the same site. Our experiments indicate that the proposed approach is highly effective.

1. Introduction

A large portion of the deep web is database-based, i.e., the data encoded in the result pages returned by search engines come from the underlying structured databases (e.g., relational databases). Such type of search engines is usually referred to as *Web databases*. A typical result page returned from a Web database consists of multiple search result records (SRRs). Usually, one SRR contains multiple data units each of which describes one aspect of the corresponding entity. For example, Figure 1 depicts a portion of a sample result page returned from a book search engine with three records on it. Each record represents one book and consists of several data units (title, author, etc.).

There is a high demand for collecting data of interest from multiple Web databases. For example, a comparison-shopping system needs to collect the price, availability, and other information of the same product from multiple providers. Such kind of applications requires that the collected data be semantically labeled so that they can be appropriately organized/stored for subsequent analysis. However, in most cases, such labels are not provided when the data units are encoded in the returned result page. Like in Figure 1, human users can easily tell that the first line of each SRR is the book title although it is not explicitly labeled as such. Early applications require huge human efforts to annotate the data units manually, which severely limits the scalability of such applications. In this paper, we want to address the problem of how to automatically assign meaningful labels to the data units embedded in the SRRs returned from Web databases.



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 text of the first record

Figure 1. Example results from bookpool.com

Our solution consists of two phases. In the first phase (the *alignment phase*), for a given result page returned from a Web database, all data units in the SRRs are first identified and then clustered into different groups. Each group contains the data units

semantically belonging to the same attribute/concept (e.g., all titles are aligned into the same group). By grouping the data units with the same semantics together, data units with the same semantics can be holistically and robustly annotated by a single label, avoiding possibly assigning different labels to these data units. The alignment also makes it easier to identify the common patterns or features among the data units with the same semantics. In the second phase (the *annotation phase*), we employ several basic annotators with each exploiting one type of features. Then for each aligned group, every basic annotator is used to annotate the units within this group holistically. A probability model is adopted to combine the results of different basic annotators and to determine an appropriate label for each group. Based on the annotated data units, an *annotation wrapper* is constructed which can be used to efficiently annotate data units on new result pages retrieved from the same Web database in response to new queries.

This paper has the following contributions: (1) We propose a hierarchical clustering based approach to align data units into different groups. Instead of using only the DOM tree or other HTML tag tree structures of the result records 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, text contents, presentation formats, and adjacency information. (2) We utilize the integrated schema of the search interfaces of multiple Web databases in the same domain to enhance the label assignment process. The integrated schema can be automatically obtained and our experiments show the added benefits of using it. (3) We propose a multi-annotator approach to tackle the annotation problem with each basic annotator exploiting a different type of features. This approach is highly flexible as existing basic annotators may be modified and new annotators can be added easily without affecting the operation of others. (4) Our approach constructs an annotation wrapper for any given Web database. The wrapper can be applied to efficiently annotating the SRRs retrieved from the same Web database with new queries.

The rest of this paper is organized as follows. Section 2 reviews previous related work. Section 3 describes our data alignment algorithm. Section 4 introduces our multi-annotator approach. In Section 5, we present our wrapper generation method. Section 6 reports our experimental results and Section 7 concludes the paper.

2. Related work

Wrapper induction (e.g., [17, 18]) is a semi-automatic technique to extract the desired information from Web pages. It needs users to label desired data to extract, and then the wrapper induction system induces the rules to construct the wrapper for extracting the corresponding data. [1, 5] are two efforts to automatically construct (extraction) wrappers, but they do not annotate the extracted data. Embley et al. [7] utilize ontologies together with several heuristics to automatically extract data in multi-record documents and label them. However, ontologies for different domains must be constructed manually. [19] exploits 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 has to be hand-labelled.

We are aware of only two recent works [2, 22] aiming at automatically assigning meaningful labels to the data units in the search results without domain limitations. Arlotta et al. [2] basically annotate data units with the closest labels on result pages. However, this method has limited applicability because many sites do not encode data units with their labels on result pages. DeLa [22] is the one most similar to our work. It first uses HTML tag information to align data units by filling data units into a table through a regular expression based data tree algorithm and then it annotates the data units. Some of the basic annotators we use are also used by DeLa. However, our work differs significantly from DeLa. First, our data alignment method is not based on the HTML tag tree structure. Instead, we utilize new features that can be automatically obtained from the result page including the content and data types of the data units (see Section 3.1). Second, while our annotation approach uses both the local interface schema of a Web database and an integrated interface schema of multiple Web databases of the same domain, DeLa uses only the former. In Section 4.1, we will show that utilizing an integrated interface schema has many added benefits. Third, we use a probabilistic model to combine the results of different annotators while it is not clear how the heuristics in DeLa are combined. Finally, we construct an annotation wrapper describing the rules for identifying the data unit and their labels for any given Web database, which can be used to annotate new result pages from the same site efficiently, while DeLa needs to go through its complex annotating procedure for every new result page.

Data alignment is an effective step in achieving good annotation accuracy and it is also used in [19, 22]. However, existing automatic data alignment techniques (including the one described in [24]) are based on HTML tag tree structures only. The assumption is that the sub-trees 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 because the tag tree is very sensitive to even minor differences, which may be caused by the need to emphasize certain data unit, artificial effect of the page, or erroneous coding. Our data alignment approach differs from the previous works in two main aspects: (1) we utilize more features (see Section 3.1); (2) we employ a clustering approach to perform alignment, which has not been used by others for data alignment as far as we are aware.

To enable fully automatic annotation, the SRRs need to be automatically extracted from the result pages. We employ the ViNTs system [23] to perform this task. 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 so that no information is lost. The physical position information of each node on the rendered page, including its coordinates and area size, can also be obtained through this system.

3. Data Alignment

3.1. Features of data units

Data units belonging to the same concept from different SRRs usually share many common features. Five features are utilized in our approach.

1. ***Data Content (DC)***. The data units with the same concept often share certain keywords. This is true for several 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 Figure 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, the web designers like to put some leading labels in front of certain data units to make it easier for users to understand the data. Such data units of the same concept usually have the same leading label. For example, in Figure 1, the price of every book has the leading words **"Our Price"**.

2. ***Presentation Style (PS).*** This feature describes how a data unit is displayed on a web page. It consists of 6 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 Figure 1, all the availability information is displayed in the same font and in italic.

3. ***Data Type (DT).*** Each data unit has its own semantic type although it is just a text string in the HTML code. Seven basic data types are considered in our approach: *Date*, *Time*, *Currency*, *Integer*, *Decimal*, *Percentage*, and *Ordinary String*. Each type except *Ordinary String* has certain pattern(s) so that it can be easily identified. Text not one of the first 6 types is treated as an *Ordinary String*. Usually the data units of the same concept have the same data type.

4. ***Tag Path (TP).*** A tag path of a data unit is a sequence of tags traversing from the root of the record to the corresponding node in the tag tree. An observation is made that the tag paths of the data units with the same concept have very similar tag paths, though in many cases, not exactly the same.

5. ***Adjacency (AD).*** Consider two data units d_1 and d_2 from two different SRRs r_1 and r_2, respectively. Let p_i and s_i be the data units that precede and succeed d_i in r_i, respectively, i = 1, 2. It can be observed that if p_1 and p_2 belong to the same concept and/or s_1 and s_2 belong to the same concept, then it is more likely that d_1 and d_2 also belong to the same concept.

3.2. Text nodes and data units

Each SRR extracted by ViNTs has a tag structure that determines how its contents are displayed on the Web browser. Each node in such a tag structure is either a *tag node* or a *text node*. It is obvious that all data units are located in the text nodes. However, there does not exist a 1:1 correspondence between text nodes and data units. There are two cases. First, a text node contains exactly one data unit. In other words, the text of this node contains the value of a single attribute. We refer such kind of text nodes as "*atomic text nodes*". An *atomic text node* is the same as a data unit. The second case is that multiple data units are encoded in one text node. For example, in Figure 1, part of the second line of each SRR (e.g., **"Springer-Verlag / 1999 / 0387984135"** in the first record) is a single text node. It is clear that it consists of three semantic data units: the publisher, the publication date, and ISBN. Since the text of such kind of nodes can be considered as a composition of the texts of multiple data units, we call them "*composite text nodes*". Before aligning data units into different semantic groups, we need to deal with the problem of identifying *composite text nodes* and extracting data units from such nodes.

By analyzing a large number of result pages in real applications, the following observations can be made: if the data units of attributes $A_1 \ldots A_k$ in one SRR are encoded as a *composite text node*, then (1) it is highly likely that the data units of the same attributes in other SRRs are also encoded as *composite text nodes*; and (2) they are usually located at the same place in the SRRs. These observations are valid because generally the SRRs are generated by template programs.

3.3. Data alignment algorithm

It is not difficult to see that all the features described in Section 3.1 are applicable to text nodes, including composite text nodes. Our data alignment algorithm consists of three steps:

1. *Align text nodes*. This step places text nodes with the same concept (for atomic nodes) or the same set of concepts (for composite nodes) into the same group.

2. *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" are split is called a *composite group*.

3. *Align data units*. For each composite group, this step places the data units corresponding to the same concept into the same group.

The same algorithm is used in the first and the third steps above, with the only difference being that for the former text nodes are considered while for the latter data units are considered. To ease discussion, we refer both text nodes and data units as *data units* from now on when there is no confusion. A clustering algorithm is utilized to place similar *data units* into the same group and dissimilar *data units* into different groups.

In this paper, the similarity between two *data units* is defined as follows. If they are from the same SRR, their similarity is 0. Otherwise the similarity is defined as the aggregated sum of the similarities of the 5 features between the two *data units*. More specifically, the similarity between *data units d1* and *d2* is:

$$Sim(d1, d2) = w1*SimC(d1, d2) + w2*SimP(d1, d2) + w3*SimD(d1, d2) + w4*SimT(d1, d2) + w5*SimA(d1, d2).$$

The feature similarities are defined as follows:

Data content similarity (SimC): It is the Cosine similarity [26] between texts of the two *data units*.

Presentation style similarity (SimP): It is the ratio of the number of style features the two *data units* match over the six style features used in our approach.

Data type similarity (SimD): If two *data units* have the same data type, the similarity is 1; otherwise, 0.

Tag path similarity (SimT): This is the edit distance between the tag paths of *d1* and *d2*. The edit distance ***(EDT)*** refers to the number of insertions and deletions needed to transform one tag path into the other. Obviously, the maximum number of operations needed is the total number of tags in the two tag paths. Let *t1* and *t2* be the tag paths of *d1* and *d2* respectively,

$$SimT(d1,d2) = 1 - EDT(t1, t2) / (Len(t1)+Len(t2)),$$

where $Len(t)$ denotes the number of tags in tag path t.

Adjacency similarity (SimA): This is the average similarity between the preceding *data units* and between the succeeding units of *d1* and *d2*. Only the first 4 features are used in this computation.

We apply the agglomerative clustering algorithm [16] to cluster the *data units*. Initially, each *data unit* forms a separate group of its own. We then repeatedly merge two groups that have the highest similarity value until no two groups have similarity above a threshold T. Each remaining group then contains the data units of the same concept. The similarity between two groups *C1* and *C2* is defined to be the average of the similarities between every *data unit* in *C1* and every *data unit* in *C2*.

The data unit feature weights and the clustering threshold T form an *Alignment Performance Vector* $APV = \{w1, w2, w3, w4, w5, T\}$. A genetic algorithm based method [27] is used to obtain the best *APV* that leads to the highest performance over a set of training data. The algorithm starts with a randomly generated initial population containing 30 *APV*s. The evolution of *APV*s from one generation to the next consists of three processes: crossover, mutation and reproduction. Crossover is done for each consecutive pair of *APV*s with probability 0.75. When crossover is to be done, a randomly generated binary mask with the same number of components as the *APV* is created. Each mask is used to generate a child from a pair of parents. The binary values, zero or one, in each mask are used to select the value of an *APV* component from either the first or the second parent, respectively. Mutation is performed on each child *APV* with probability 0.1. When mutation is performed, each *APV* component is either decreased or increased by 0.01 with equal probability (0.5) subject to the constraint of the boundaries of each range. Reproduction uses a *wheel of fortune* scheme to select *APV*s in the current generation to be in the next generation. It selects fit *APV*s with high probability and unfit *APV*s with low probability. The exact probability that an *APV* is selected is the fitness of the *APV*, which is computed as follows. If the performance returned by an *APV* is above 85%, the initial fitness value is the returned performance value; otherwise, 0. The final fitness of an *APV* in a generation is its initial fitness value divided by the sum of the initial fitness values of all the *APV*s in this generation. The genetic algorithm terminates after the *APV*s converge, and the fittest *APV* of all generations is selected as the "optimal" *APV*.

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. In our approach, the *split text nodes step* is carried out as follows. First, a group of text nodes is not "split-able" when one of the following constraints is satisfied: (a) a node is a hyperlink (a hyperlink is

assumed to already represent a single semantic unit); (b) the texts of all nodes in the group are the same; (c) all the nodes in the group have the same non-string data type. For example, if all nodes in the group are of *date* type, each node is already a single semantic unit. Next, the group is split if none of the above constraints is satisfied. To do the splitting correctly, we need to identify the right *separators*. We observe that (1) the same separator is likely to be used to separate different semantic units within the same text node, and (2) the same separator is likely to be used to separate the texts in the data units of the same concept across multiple SRRs. Based on these observations, in this step, we first scan the text strings of every text node in the group and select the symbol (non-letter and non-digit) with the most occurrences as the separator. Second, for each text node in the group, its text is split into several small pieces using the separator, each of which represents a real data unit.

After the splitting is completed for a *composite group*, the data units in this group are not aligned yet. Using the separators to generate the alignment directly may be problematic because the "values" in the composite text nodes often do not have a uniform format, for example, some data units may be missing in some of these "values" if the corresponding SRRs do not have information for some attributes. Our solution to the data unit alignment problem is to apply the above agglomerative clustering method to the data units in each composite group.

4. Assigning labels

4.1. Local vs. integrated interface schemas

For a Web database, its local search interface often contains some attributes of the underlying data. We denote a local search interface schema (**LIS**) as $S_i = \{A_1, A_2, ..., A_k\}$, 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 $S_e = \{a_1, a_2, ..., a_n\}$, where each a_j ($j = 1,...,n$) is an attribute to be discovered. The schema of the retrieved data and the interface schema usually share a significant number of attributes [21]. This observation provides the basis for some of our basic annotators (see Section 4.2). If an attribute a_t in the search results does have 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 for specifying query conditions. This phenomenon raises a so-called *local interface schema inadequacy problem*. Given a "hidden" attribute a_t discovered in the search result schema S_e, there is no matching attribute A_t in the local interface schema 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 LIS for annotation is the *inconsistent label problem*, i.e., different labels are assigned to semantically identical data units returned from different sites because these sites may give different names to the same attribute (e.g., **"Format"** vs. **"Binding Type"**). This will cause difficulty when using the annotated data collected from different sites, for example, for data integration applications.

In our approach, for each used domain, we use WISE-Integrator [12, 13] as the basic tool to automatically build an integrated interface schema (**IIS**) over multiple Web databases in that domain. The generated integrated interface combines all the attributes of the local interface schemas. For the attributes of the same concept, their values are also merged. Each global attribute has a unique global name and an *attribute-mapping table* is created to establish the mapping between the name of each local interface schema attribute and its corresponding name in the integrated interface schema.

For each Web database in a given domain, our approach uses both the LIS of the database and the IIS of the domain to annotate the retrieved data units. Using IIS has two major advantages. First, it has the potential to increase the annotation recall. Since the integrated interface contains the attributes on *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 on the local interface. Second, when an annotator discovers a label for a group of data units, the label will be replaced with its corresponding global attribute name (if any) on the IIS by looking up the attribute-mapping table so that the data units of the same concept across different Web databases will have the same label.

We should point out that even though using the integrated interface schema can significantly alleviate the *local interface schema inadequacy problem* and the *inconsistent label problem*, it does not 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. As to the second problem, for example, in Figure 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

integrated interface schema cannot solve the problem and new techniques are needed.

4.2. Basic annotators

The data units belonging to the same concept (attribute) often share special common features, which are usually displayed in certain patterns. Based on this observation, we define 6 basic annotators to label data units, with each of them considering a special type of patterns/features.

Table Annotator (TA)

Many Web databases use a table to organize the returned SRRs, which visually has multiple rows and columns with each row representing an SRR. The table header, which indicates the meaning of each column, is usually located at the top of the table. Figure 2 is an example of such a Web database. 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.

Job Title	Company Name	Location	Salary
Interactive Advertising Project Manager	EXPERTseeker.com	Boonton, NJ	$55K-$75K
Director of Quality Assurance	Quigo	New York, NY	--
Google: Software Engineer	Google Inc.	New York, NY	--

Figure 2. Result page with table format

Our Table Annotator uses the physical position information of each data unit obtained at the extraction step and it 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. The remaining data units are processed similarly. In case that the table header is not provided or is not successfully extracted by the ViNTs extractor [23], the Table Annotator will not be applied.

Query-based Annotator (QA)

The basic idea of this annotator is that the returned SRRs from a Web database are always related to the specified query. Specifically, the query terms entered through the search attributes on the local search interface of the Web database will most likely appear in some retrieved SRRs. For example, in Figure 1, query term "machine" is submitted through the "Title" field and all three titles of the returned SRRs contain this query term. Thus, the search attribute name "Title" can be used to annotate the title values of these SRRs.

Given a query with a set of query terms submitted against an attribute A_j on the local search interface, the query-based annotator finds the group that has the largest total occurrences of these query terms and assigns the name of A_j to the group. As mentioned in Section 4.1, the LIS 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. Even so, our experimental results show that this annotator is the most effective one – it can correctly annotate about 40% of the attributes.

Schema value Annotator (SA)

Many attributes on a search interface have pre-defined values on the interface. For example, the attribute "**Publishers"** may have a set of pre-defined values in its selection list. Our schema value annotator is to utilize the combined value set on the integrated interface to perform annotation.

Given a group of data units $G_i = \{u_1, u_2, ..., u_n\}$, the schema value annotator is to discover the best matched attribute to the group from the integrated interface schema. Let A_j be an attribute containing a list of values $\{v_1, v_2, ..., v_m\}$ on the IIS. For each data unit u_k, this annotator first computes the Cosine similarities between u_k and all values in A_j to find the value (say v_t) with the highest similarity. Then the annotator sums up the similarities for all the data units and multiplies the sum by the number of non-zero similarities. This final value is treated as the matching score between G_i and A_j. The schema value annotator uses the name of the attribute A_j that has the highest matching score among all attributes to annotate the group G_i. Note that multiplying the above sum by the number of non-zero similarities is to give preference to attributes that have more matches (i.e., having non-zero 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].

Frequency-based Annotator (FA)

In Figure 1, "**Our Price**" appears in the three records and the followed price values are all different. 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 fields, 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 Web databases 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 should be concatenated to form the label for the group G_i. For example, in Figure 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.

In-text prefix Annotator (IA)

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

Given a group, the in-text prefix annotator checks whether all data units share the same prefix. 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 to annotate values following it. In the above example, the label "You save" will be assigned to the group. Any group whose data units are completely identical is not considered by this annotator.

Common knowledge Annotator (CA)

Some data units on the result page are self-explanatory because of the common knowledge shared by the human beings. For example, "in stock" or "out of stock" occurs in many SRRs from e-commerce sites. It is the common sense that it is about the availability information of the product. 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. 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.

It should be pointed out that our common concepts are different from the ontologies that are widely used in semantic Web (e.g., [6, 9, 10, 15, 20]). First, our *common concepts* are domain independent. Second, they can be obtained from existing information resources with little additional human effort. The DeLa approach [22] also employed table annotator, query-based annotator, in-text prefix annotator and common knowledge annotator as its basic annotators and its table annotator is implemented differently from the one proposed in this paper.

4.3. Combining annotators

The *applicability* of an annotator is defined as the percentage of the attributes to which the annotator can be applied. Say out of 10 attributes, if 4 appear in tables, then the applicability of the Table Annotator is 40%. Our analysis shows that on the average, no single annotator can annotate more than 50% of the attributes on the result pages collected in our dataset. This indicates that the results of different annotators need to be combined to achieve higher applicability. Moreover, different annotators may produce different labels for a given group. Thus, we need strategies to select the most appropriate one for the group.

Our annotators are fairly independent from each other since each exploits an independent feature. Based on this characteristic, we propose a simple probability-based 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. $P(L)$ 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, $i = 1, ..., k$, identify the same label for a group of data units, then the combined probability that at least one of the annotators is correct is .

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. For example, suppose for a given group of data units G, one annotator is not applicable to G, two annotators assign label L1 to G with combined probability P1, and three annotators assign label L2 to G with combined probability P2. If P1 > P2, then L1 will be the final label chosen for G. Ties are broken by selecting the label identified by more annotators. Thus, if P1 = P2, then L2 will be chosen for G.

The advantage of this method is that it provides high flexibility. If an existing annotator is modified or a new annotator is added in, all we need to do is to obtain the success rate of this new/revised annotator while keeping all remaining annotators unchanged.

5. Annotation wrapper

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, each group corresponds to an attribute in the SRRs. They are organized based on the order of its data units in the original SRR. Consider the i^{th} group G_i. Every SRR has a tag-node sequence like Figure 1(b) 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 Figure 1(b) has **"<FORM><A>"** as its prefix and **"</A>
"** as its suffix. If a data unit is generated by splitting from a parent unit, then its prefix and suffix are the same as those of its parent data unit. At the same time, the separators used for splitting the parent unit as well as its position index in the split unit vector are recorded. Thus, the annotation rule for each attribute is 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 exist 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 assigned the 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 $unitindex_i$.

6. Experiments

Our experiments are based on 91 Web databases randomly selected from 6 domains: *book*, *movie*, *music*, *game*, *job*, and *auto*. For each Web database, its local interface schema is extracted automatically using WISE-*i*Extractor [14]. For each domain, WISE-Integrator [12, 13] is used to build the integrated interface schema automatically. These collected sites are randomly divided into two disjoint groups. The first group contains 26 sites to be used for training, and the second group has 65 sites for testing. For each training site, one sample result page is obtained and two data sets, **DS1** and **DS2**, are formed. DS1 contains 14 pages, which is used for training the weights of the data unit features used in the alignment step, including the threshold T (See Section 3.3). DS2 contains all 26 pages and it is used to determine the success rate of each annotator (See Section 4.3). For each testing site, we collect 2 sample result pages using different queries to form data sets, **DS3** and **DS4**. Each of them contains one page from every site. DS3 is used to test the performance of our alignment and annotation methods based on the parameter values and statistics obtained from DS1 and DS2. At the same time, the annotation wrapper for each site will be generated. DS4 is used to test the quality of the generated wrappers. For each result page in the four data sets, the data units are manually extracted, aligned in groups, and assigned labels by a human expert, and the information will be used as the ground truth for comparison purpose.

We adopt the *precision* and *recall* to measure the performance of our methods. For *alignment*, the *precision* is defined as the percentage of the correctly aligned data units over all the units aligned by the system; *recall* is the percentage of the data units correctly aligned by the system over all data units aligned by the expert. For annotation, *precision* is the percentage of the correctly annotated units over all the data units annotated by the system; *recall* is the percentage of the data units correctly annotated by the system over all the manually annotated units.

The optimal *APV* obtained through our genetic training method is {0.30, 0.67, 1.08, 0.49, 0.14, 0.86}. The alignment precision and recall yielded using this *APV* are both **97.6%** on average over the 14 pages in DS1. This result indicates that the data type and the presentation style are the most important features.

Each time, one annotator is used to annotate all 26 pages in DS2 to obtain its success rate. The success rate for table annotator is 1.0, 0.95 for query-based annotator, 0.93 for frequency-based annotator, 0.45 for schema value annotator, 0.85 for common knowledge based annotator, and 0.64 for in-text prefix annotator. The table annotator is 100% correct when applicable. The frequency-based and the query-based annotator also have very high success rate while the schema value annotator is the least. None of the annotators can be applied to more than 50% of the attributes.

The collected pages in DS3 are used to evaluate our alignment and annotation methods. The alignment performance is evaluated by comparing the system generated aligned groups with the manually identified groups. Table 1 shows the average precision and recall for each domain and the overall average precision and recall for all 65 pages. Our method achieves very high accuracy in all domains in terms of both precision and recall. The overall average performances are nearly the same as those obtained from training, which shows that our alignment method is robust. The errors usually happen when an attribute has multiple data units in the same SRR (e.g., multiple authors for a book). We will address this issue in the future.

Table 1. Performance of data alignment

Domain	Precision	Recall
Auto	100%	100%
Book	97.3%	97.3%
Job	99.3%	99.3%
Movie	99.3%	99.3%
Music	96.8%	96.8%
Game	96.2%	96.2%
average	98.0%	98.0%

After the aligned groups for the pages in DS3 are annotated, we compare the system assigned label and the manually assigned label of each data unit to see if they match. From Table 2, we can see that both precision and recall are very high, which shows that our combined annotation method is very effective.

Table 2. Performance of annotation

Domain	DS3		DS4	
	Precision	Recall	Precision	Recall
Auto	100%	95.8%	94.5%	91.7%
Book	97.2%	96.2%	96.7%	91.9%
Job	95.3%	92.6%	100 %	92.1%
Movie	99.7%	97.7%	99.7%	97.7%
Music	93.9%	93.9%	97.6%	93.9%
Game	98.9%	98.8%	97.3%	92.0%
average	97.2%	95.9%	96.8%	92.3%

For each web page in DS3, once its SRRs have been annotated, an annotation wrapper is built and applied to the corresponding page in DS4. From Table 2, we can see that the precision is nearly the same as that when wrappers are not used while the recall drops more than 3 percentage points. The reason is that in this experiment, for each site in DS3, only one page is used to build the wrapper. As a result, it is possible that some attributes that appear on the page in DS4 do not appear on the training page in DS3 (so they do not appear in the wrapper expression). For example, many Web databases only allow user to send one query based on only one attribute each time. If the query term is based on "Title", then the data units for attribute "Author" may not be correctly annotated because the *query-based annotator* is the main technique used for attributes that have a textbox on the search interface. One solution to remedy this problem is to combine multiple result pages based on different queries to build a more robust wrapper. However, there may still exist a problem that some sites provide different layout format for the queries based on different attributes. We would like to further investigate this issue in the future.

We also conducted experiments to evaluate the significance of each feature on the performance of our alignment algorithm. 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 with the new parameters is run on DS3 again. The performance is then compared with that when all the features are used. Figure 3 shows the results. When any one of these features is not used, both the precision and recall decrease, which shows that all the features used in our approach are valid and useful. We can also see that the data type and the presentation style are the most important features. When they are not used, the precision and recall drop around 25 percentage points. The adjacency feature is the least significant one, but without it, the precision and recall drop about 3 percentage points.

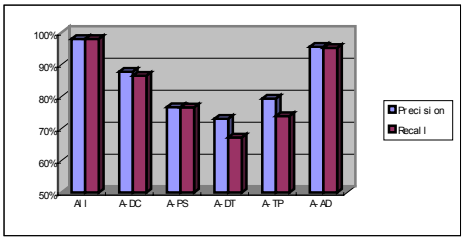


Figure 3. Evaluation of alignment parameters

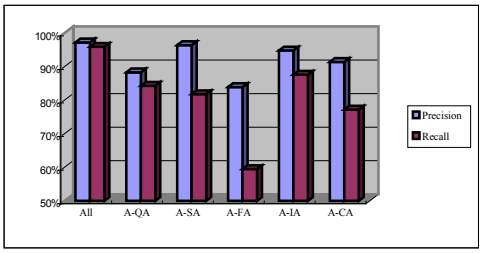


Figure 4. Evaluation of annotators

We use a similar method to evaluate the significance of each annotator. Each time, one annotator is removed and the remaining annotators are used to annotate the pages in DS3. From Figure 4, we can see that first, omitting any annotator causes both precision and recall to drop, i.e., every annotator contributes positively to the overall performance. Among the 6 annotators, the query based annotator and the frequency based annotator are the most significant. Moreover, when an annotator is removed, the recall decreases more severely than precision. This indicates that each of our annotators is fairly independent in describing one aspect of the attribute which, to a large degree, is not applicable to other annotators.

Finally, we conducted experiments to study the effect of using LIS versus IIS in annotation. We run the annotation process on DS3 again but this time, instead of using the integrated interface built for each domain, we use the local interface of each Web database. From the result depicted in Table 3, we can see that using the LIS has little effect on precision, but significant effect on recall (the overall average recall is reduced by almost 7 percentage points) because of the

local interface schema inadequacy problem (see Section 4.1). And it also proves that using IIS can indeed increase the annotation performance.

Table 3. Performance using IIS vs. LIS

Domain	IIS		LIS	
	Precision	Recall	Precision	Recall
Auto	100%	95.8%	100%	99.4%
Book	97.2%	96.2%	97.0%	85.9%
Job	95.3%	92.6%	95.3%	95.3%
Movie	99.7%	97.7%	99.7%	93.1%
Music	93.9%	93.9%	92.2%	84.7%
Game	98.9%	98.8%	98.9%	93.0%
average	97.2%	95.9%	96.7%	89.1%

7. Conclusion

In this paper, we studied the data annotation problem and proposed a multi-annotator approach to automatically constructing an annotation wrapper for annotating the search result records retrieved from any given Web database. Each of these annotators exploits one type of special features for annotation and our experimental results indicate that each proposed annotator is useful and they together are capable of generating high quality annotation wrappers. We also illustrated how the use of the integrated interface schema can help alleviate the *local interface schema inadequacy problem* and the *inconsistent label problem*. In addition, a new data alignment technique using richer yet automatically obtainable features was proposed to cluster data units into different groups/concepts in support of more robust and holistic annotation. There is still room for improvement in several areas as mentioned in Section 6. For example, we need to enhance the ability to deal with multi-valued attributes that may have more than one value for some SRR (e.g., authors for books).

Acknowledgment: This work is supported in part by the following NSF grants: IIS-0414981, IIS-0414939 and CNS-0454298.

8. References

[1] A. Arasu and H. Garcia-Molina. Extracting Structured Data from Web pages. SIGMOD Conference, 2003.

[2] L. Arlotta, V. Crescenzi, G. Mecca, and P. Merialdo. Automatic Annotation of Data Extracted from Large Web Sites. WebDB Workshop, 2003.

[3] P. Chan and S. Stolfo. Experiments on Multistrategy Learning by Meta-Learning. CIKM Conference, 1993.

[4] W. Bruce Croft. Combining approaches for information retrieval. In Advances in Inf. Retr.: Recent Research from the Center for Intel. Inf. Retr., Kluwer Academic, 2000.

[5] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRUNNER: Towards Automatic Data Extraction from Large Web Sites. VLDB Conference, 2001.

[6] S. Dill, N. Eiron, D. Gibson, and et al. SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation. WWW Conference, 2003.

[7] D. Embley, D. Campbell, Y. Jiang, S. Liddle, D. Lonsdale, Y. Ng, R. Smith. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages. Data and Knowledge Engineering, 31(3), 227-251, 1999.

[8] D. Freitag. Multistrategy Learning for Information Extraction. ICML, 1998.

[9] S. Handschuh, S. Staab, and R. Volz. On Deep Annotation. WWW Conf., 2003.

[10] S. Handschuh and S. Staab. Authoring and Annotation of Web Pages in CREAM. WWW Conference, 2003.

[11] B. He, K. Chang. Statistical Schema Matching across Web Query Interfaces. SIGMOD Conference, 2003.

[12] H. He, W. Meng, C. Yu, and Z. Wu. Automatic Integration of Web Search Interfaces with WISE-Integrator. VLDB Journal, Vol.13, No.3, pp.256-273, September 2004.

[13] H. He, W. Meng, C. Yu, Z. Wu. WISE-Integrator: A System for Extracting and Integrating Complex Web Search Interfaces of the Deep Web. VLDB, Demo, 2005.

[14] H. He, W. Meng, C. Yu, and Z. Wu. Constructing Interface Schemas for Search Interfaces of Web Databases. WISE Conference, 2005.

[15] J. Heflin and J. Hendler. Searching the Web with SHOE. AAAI Workshop, 2000.

[16] L. Kaufman, P. Rousseeuw. Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons, 1990.

[17] N. Krushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. IJCAI Conf., 1997.

[18] L. Liu, C. Pu, and W.Han. XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources. ICDE conference, 2001.

[19] S. Mukherjee, I. V. Ramakrishnan, and A. Singh. Bootstrapping Semantic Annotation for Content-Rich HTML Documents. ICDE, 2005.

[20] B. Popov, A. Kiryakov, A. Kirilov, D. Manov, D. Ognyanoff, and M. Goranov. KIM - Semantic Annotation Platform. ISWC Conf., 2003.

[21] J. Wang, J. Wen, F. Lochovsky and W. Ma. Instance-based Schema Matching for Web Databases by Domain-specific Query Probing, VLDB conference, 2004.

[22] J. Wang and F.H. Lochovsky. Data Extraction and Label Assignment for Web Databases. WWW Conference, 2003.

[23] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully Automatic Wrapper Generation for Search Engines. WWW, 2005.

[24] Y. Zhai and B. Liu. Web Data Extraction Based on Partial Tree Alignment. WWW 2005, Chiba, Japan.

[25] O. Zamir and O. Etzioni. Web Document Clustering: A Feasibility Demonstration. ACM SIGIR Conference, 1998.

[26] G. Salton and M. McGill. Introduction to Modern Information Retrieval. McGraw-Hill, 1983.

[27] D. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, 1989.