Learning to Match the Schemas of Data Sources: A Multistrategy Approach

AnHai Doan, Pedro Domingos, Alon Halevy
{anhai, pedrod, alon}@cs.washington.edu
Department of Computer Science and Engineering
University of Washington, Seattle, WA 98195, USA

Abstract

The problem of integrating data from multiple data sources – either on the Internet or within enterprises – has received much attention in the database and AI communities. The focus has been on building data integration systems that provide a *uniform* query interface to the sources. A key bottleneck in building such systems has been the laborious manual construction of *semantic mappings* between the query interface and the source schemas. Examples of mappings are "element location maps to address" and "price maps to listed-price". We propose a multistrategy learning approach to automatically find such mappings. The approach applies multiple learner modules, where each module exploits a different type of information either in the schemas of the sources or in their data, then combines the predictions of the modules using a meta-learner. Learner modules employ a variety of techniques, ranging from Naive Bayes and nearest-neighbor classification to entity recognition and information retrieval. We describe the LSD system, which employs this approach to find semantic mappings. To further improve matching accuracy, LSD exploits domain integrity constraints, user feedback, and nested structures in XML data. We test LSD experimentally on several real-world domains. The experiments validate the utility of multistrategy learning for data integration and show that LSD proposes semantic mappings with a high degree of accuracy.

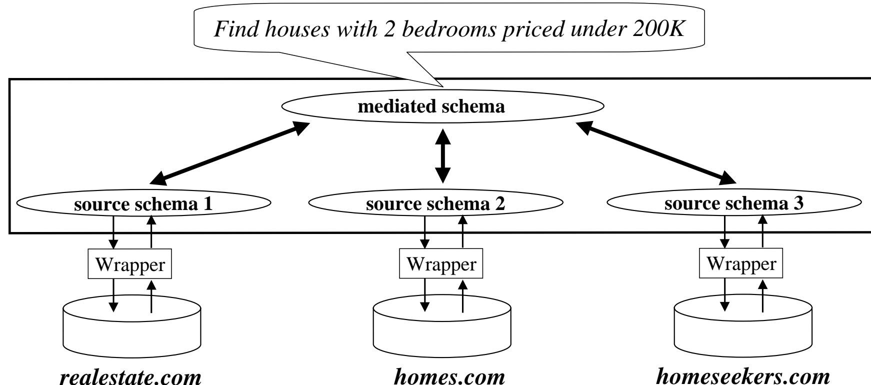


Figure 1: A data integration system in the real-estate domain.

1 Introduction

The number of structured data sources available online is growing rapidly. Integrating data across such sources holds the potential to better many aspects of our lives, ranging from everyday activities such as reading news and choosing a movie to important tasks such as buying a house and making a business decision. Manually integrating data from multiple sources, however, is extremely labor intensive. Hence, researchers in both the database and AI communities have proposed building *data integration systems* (e.g., [GMPQ+97, LRO96, IFF+99, LKG99, FW97, KMA+98]). Such a system provides a *uniform* query interface to a multitude of data sources, thereby freeing the user from the tedious job of accessing the individual sources, querying them, and manually combining the answers.

Consider for example a data integration system that helps users find houses on the real-estate market (Figure 1). The system provides the uniform interface in terms of a *mediated schema* that captures the relevant aspects of the real-estate domain. The mediated schema may contain elements such as ADDRESS, PRICE, and DESCRIPTION, listing the house address, price, and description, respectively. The system maintains for each data source a *source schema* that describes the content of the source. *Wrapper programs*, attached to each data source, handle data formatting transformations between the local data model and the data model in the integration system.

Given a user query formulated in the mediated schema, such as "find houses with 2 bedrooms priced under $200K", the system translates the query into queries in the source schemas, executes these queries with the help of the wrappers, then combines the data returned by the sources to produce the final answers.

To translate user queries, a data integration system uses *semantic mappings* between the mediated schema and the local schemas of the data sources. Examples of mappings are "element ADDRESS of the mediated schema maps to element location of the schema of *realestate.com*" and "PRICE maps to listed-price". Today, such semantic mappings are specified *manually* by the system builder, in a very labor-intensive and error-prone process. The acquisition of semantic mappings therefore has become a key bottleneck in deploying data integration systems. The emergence of XML [XML98] as a standard *syntax* for sharing data among sources further fuels data sharing applications and underscores the need to develop methods for acquiring semantic mappings. Hence, the development of techniques to automate the mapping process has now become crucial to truly

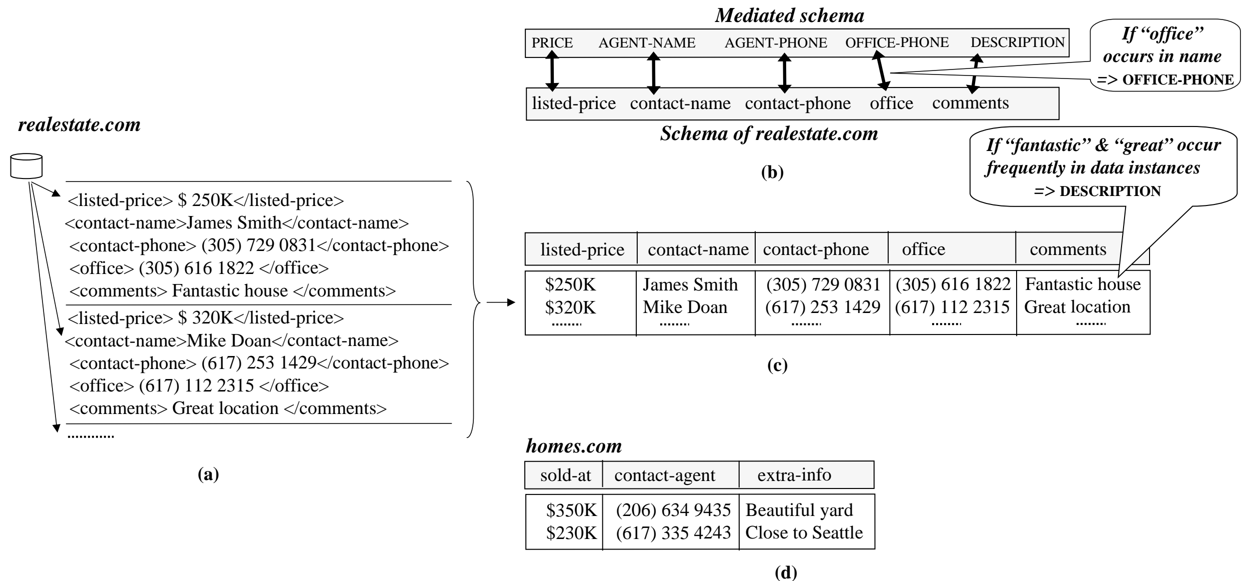


Figure 2: Source *realestate.com* provides data in XML format, as shown in (a). Once we have manually provided the mappings between the source schema and the mediated schema (the five arrows in (b)), a system can learn many different matching hypotheses from schemas in (b) and source data in (c), then apply them to predict mappings for a new source *homes.com* in (d).

achieve large-scale data integration.

In this paper we describe a machine learning approach to automatically create semantic mappings. Throughout the discussion, we shall assume that the sources present their data in XML and that the mediated- and source schemas are represented with XML DTDs (see Section 2 for more details on XML). The schema-matching problem is then to find semantic correspondences between the elements of the mediated schema and the source DTDs.

The key idea underlying our approach is that after a small set of data sources have been manually mapped to the mediated schema, a schema-matching system should be able to automatically learn from these mappings to successfully propose mappings for subsequent data sources.

Example 1 Consider the data-integration system in Figure 1. Suppose we have source *realestate.com* in Figure 2.a, for which we have manually specified the semantic mappings. This amounts to specifying the five arrows in Figure 2.b. The first arrow, for example, states that source-schema element listed-price matches mediated-schema element PRICE.

Once we have specified the mappings, a machine learning system can exploit different types of information in the source schema and data, for the purpose of finding future semantic mappings. It can exploit the *names* of schema elements: knowing that office matches OFFICE-PHONE, it could construct a hypothesis that if an element name contains the word “office”, then that element is likely to be OFFICE-PHONE. The system can also look at example phone numbers in the source data (Figure 2.c), and learn the *format* of phone numbers. It could also learn from *word frequencies*: it could discover that words such as “fantastic” and “great” appear frequently in house descriptions. Hence, it may hypothesize that if these words appear frequently in the data instances of an element, then that element is likely to be DESCRIPTION. As yet another example, the system could also

learn from the *characteristics of value distributions*: it can look at the average value of an element, and learn that if that value is in the thousands, then the element is more likely to be price than the number of bathrooms.

Once the system has learned from the manually provided mappings, it is then able to find semantic mappings for new data sources. Consider source *homes.com* in Figure 2.d. First, the system extracts data from this source to populate a table where each column consists of data instances for a single element of the source schema. Next, the system examines the name and the data instances of each column, and applies the learned hypotheses to predict the matching mediated-schema element. For example, when examining column extra-info, the system could predict that it matches DESCRIPTION, because the data instances in the column frequently contain the words "fantastic" and "great", and therefore are likely to be house descriptions.□

In general, there are *many* different types of information that a schema-matching system can exploit, such as names, formats, word frequencies, positions, and characteristics of value distributions. It is important that the system exploit all such types of information, in order to maximize matching accuracy. Consider element contact-agent of source *homes.com* (Figure 2.d). If the system exploits only the *name* "contact-agent", it can only infer that the element relates to the agent, and hence matches either AGENT-NAME or AGENT-PHONE. If the system exploits only the *data values* of the element, it can recognize only that they are phone numbers, and hence the element matches either AGENT-PHONE or OFFICE-PHONE. Only when combining both types of information can the system correctly infer that the element matches AGENT-PHONE.

Clearly, no single learning technique will be able to exploit effectively all types of information. Hence, our work takes a *multistrategy learning* approach [MT94]. To predict mappings for a new data source, we propose applying a multitude of learning modules, called *base learners*, where each base learner exploits well a certain type of information, then combining the base learners' predictions using a *meta-learner*. The meta-learner uses the manually mapped sources to learn a set of weights for each base learner. The weights indicate the relative importance of the base learners and can be different for each mediated-schema element, reflecting that different learners may be most appropriate in different cases.

We have developed the LSD (Learning Source Descriptions) system, which employs the above multistrategy learning approach to semi-automatically create semantic mappings. To improve matching accuracy, LSD extends multistrategy learning to exploit domain integrity constraints that appear frequently in database schemas, and to incorporate user feedback on the proposed mappings. It also employs a novel learning technique that utilizes the nested structure inherent in XML data.

In this paper, we focus on describing multistrategy learning, the central component of the LSD system. We discuss additional techniques employed by LSD briefly in Section 4 and in detail in [DDH01]. Specifically, we make the following contributions:

- We propose the use of multistrategy learning for finding semantic mappings. In addition to providing accuracy superior to that of any single learner, the technique has the advantage of being highly extensible when new learners are developed.

- We describe the implementation of multistrategy learning in the LSD system. LSD extends multistrategy learning to exploit domain constraints, user feedback, and XML structure, in order to propose semantic mappings with high accuracy.

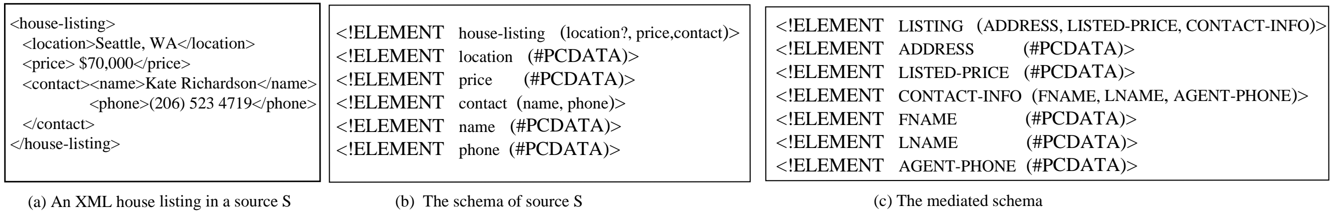


Figure 3: Examples of XML listing, source schema (DTD), and mediated schema.

- We describe experiments on several real-world data integration domains. The results validate the utility of multistrategy learning, and show that with the current set of learners, LSD already obtains high predictive accuracy of 71-92% across all tested domains.

The paper is organized as follows. The next section defines the schema-matching problem. Section 3 describes our multistrategy learning approach. Section 4 describes the current LSD implementation. Section 5 presents our experiments. Section 6 discusses the limitations of the current LSD system. Section 7 reviews related work. Section 8 discusses future work and concludes.

2 The Schema-Matching Problem

The goal of schema matching is to produce semantic mappings that enable transforming data instances of one schema into instances of the other [MHH00, RB01]. In many common cases, the mappings are *one-to-one (1-1)* between elements in the two schemas (e.g., "location maps to ADDRESS"), while in others, the mappings may be more complex (e.g., "num-baths maps to HALF-BATHS + FULL-BATHS"). In this paper we focus on computing 1-1 mappings between the schemas. In a separate paper [DDH02] we discuss extending multistrategy learning to compute more complex mappings.

We emphasize that the input to the schema matching problem considered here is already *structured* data. In many data-integration applications, it is necessary to precede schema matching by a *data extraction* phase. In this phase, unstructured data (e.g., text, HTML) is mapped to some structured form (e.g., tuples, XML). Data extraction has been the focus of intensive research on wrappers, information extraction, and segmentation (e.g., [Kus00a, AK97, HGMN$^+$98, Fre98]), often also benefiting from machine learning techniques.

2.1 Schema Matching for XML DTDs

The eXtensible Markup Language [XML98] is increasingly being used as a protocol for the dissemination and exchange of information between data sources. Hence, in this paper we consider the problem of discovering semantic mappings in the context of XML data. In addition to encoding relational data, XML can encode object-oriented, hierarchical, and semi-structured data. An XML document consists of pairs of matching *open-* and *close-tags*, enclosing *elements*. Each element may also enclose additional sub-elements. A document contains a unique root element, in which all others are nested. (In general, an XML element may also have a set of *attributes*. For the purpose of this paper we treat its attributes and sub-elements in the same fashion.)

Figure 3.a shows a house listing stored as an XML document. Here "⟨location⟩Seattle, WA⟨/location⟩" is an XML element. The *name* of this element is "location", and the *data value* is "Seattle, WA".

We consider XML documents with associated DTDs (Document Type Descriptors). A DTD is a BNF-style grammar that defines legal elements and relationships between the elements. We assume that the mediated schema is a DTD over which users pose queries, and that each data source is associated with a source DTD. Data may be supplied by the source directly in this DTD, or processed through a wrapper that converts the data from a less structured format. Figures 3.b-c show sample source- and mediated-DTDs in the real-estate domain. Throughout the paper, we shall use this font to denote source-schema elements, and THIS FONT for mediated-schema elements.

Given a mediated DTD and a source DTD, the *schema-matching problem* is to find semantic mappings between the two DTDs. In this paper we start by considering the case of finding *one-to-one (1-1) mappings* between tag names of the source DTD and those of the mediated DTD. For example, in Figures 3.b-c, tag location matches ADDRESS, and contact matches CONTACT-INFO. Intuitively, two tags (i.e., schema elements) match if they refer to semantically equivalent concepts. The notion of semantic equivalence is subjective and heavily dependent on the particular domain and context for which data integration is performed. However, as long as this notion is interpreted by the user *consistently* across all data sources in the domain, the equivalence relations LSD learns from the training sources provided by the user should still apply to subsequent, unseen sources.

2.2 Schema Matching as Classification

Our approach rephrases the problem of finding 1-1 mappings as a *classification* problem: given the mediated-DTD tag names as distinct *labels* $c_1, \ldots, c_n$, we attempt to assign to each source-schema tag a matching label. (If no label matches the source-schema tag, then the unique label OTHER is assigned.)

Classification proceeds by training a learner L on a set of *training examples* $\{(x_1, c_{i1}), \ldots, (x_m, c_{im})\}$, where each x_j is an object and c_{ij} is the observed label of that object. During the *training phase*, the learner inspects the training examples and builds an *internal classification model* (e.g., the hypotheses in Figure 2.a) on how to classify objects.

In the *matching phase*, given an object x the learner L uses its internal classification model to predict a label for x. In this paper we assume the prediction is of the form

$$\langle (c_1, s_1), (c_2, s_2), \ldots, (c_n, s_n) \rangle,$$

where $\sum_{i=1}^{n} s_i = 1$ and a pair (c_i, s_i) means that learner L predicts x matches label c_i with *confidence score* s_i. The higher a confidence score, the more certain the learner is in its prediction. (In machine learning, some learners output a *hard* prediction, which is a single label. However, most such learners can be easily modified to produce confidence-score predictions.)

For example, consider the *Name Learner* which assigns a label to an XML element based on its *name* (see Section 4.1 for more detail). Given an XML element, such as "⟨phone⟩ (235) 143 2726⟨/phone⟩", the Name Learner inspects the name, which is "phone", and may issue a prediction such as ⟨(ADDRESS,0.1), (DESCRIPTION,0.2), (AGENT-PHONE,0.7)⟩.

3 Solution: Multistrategy Learning

We propose a multistrategy learning approach to the schema-matching problem. Our approach operates in two phases: training and matching. In the training phase we manually mark up the schemas of a few data sources, then use them to train the learners. In the matching phase we apply

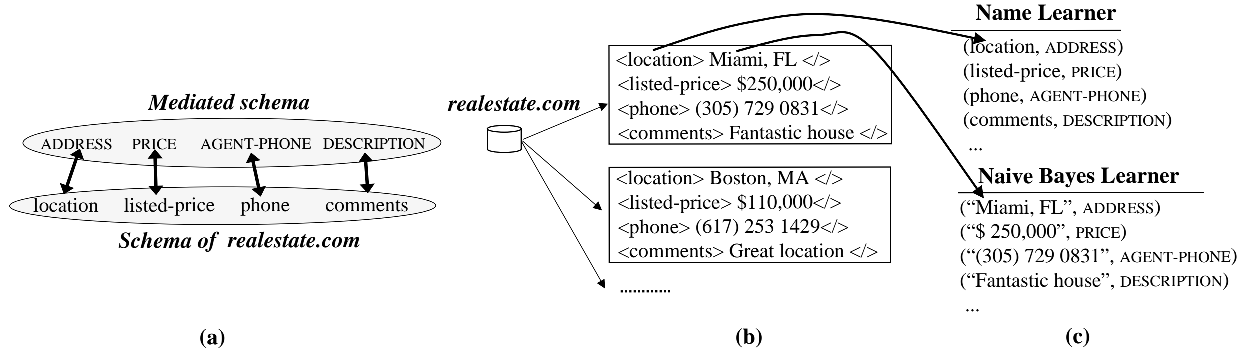


Figure 4: The training phase of our multistrategy learning approach.

the trained learners to predict mappings for new sources. We now describe the two phases in detail. In Section 4 we describe the implementation of the two phases in the LSD system.

3.1 The Training Phase

Suppose we start with the mediated schema shown in Figure 4.a. To create data for training, we take a source *realestate.com* and match its schema with the mediated schema. Figure 4.a shows that source element location has been matched with mediated-schema element ADDRESS, listed-price with PRICE, and so on.

Next, we extract from *realestate.com* a set of house listings. Figure 4.b shows several such listings in XML. (For ease of exposition, we abbreviate all XML closing tags to ⟨/⟩.) We train the base learners using the extracted listings. Each base learner tries to learn to distinguish among the mediated-schema elements, so that when presented with an XML element from a new source, the learner can predict whether it matches any mediated-schema element, or none.

Even though the goal of all base learners is the same, each of them learns from a different type of information available in the extracted data or the source schema. Thus, each learner processes the extracted data into a different set of training examples. For example, consider the Name Learner and the Naive Bayes Learner (both are described in detail in Section 4.1). The Name Learner matches an XML element based on its *tag name*. Therefore, it extracts the tag names from the data and pairs them with their true labels to form the training examples shown in Figure 4.c (under the heading "Name Learner"). Example (location,ADDRESS) states that if an XML element has the name "location", then it matches ADDRESS.

The Naive Bayes Learner matches an XML element based on its *data value*. Therefore, it extracts the data values from the house listings and pairs them with their true labels to form the training examples shown in Figure 4.c (under the heading "Naive Bayes Learner"). Example ("Miami, FL",ADDRESS) states that "Miami, FL" is an ADDRESS. The learner can form this example because "Miami, FL" is an instance of source element location, and we have told it that location matches ADDRESS.

If we have more sources for training, we simply add the training data obtained from those sources to the training data from *realestate.com*. Finally, we train each base learner on its respective set of training examples.

In addition to training the base learners, we may also have to train the meta-learner, depending

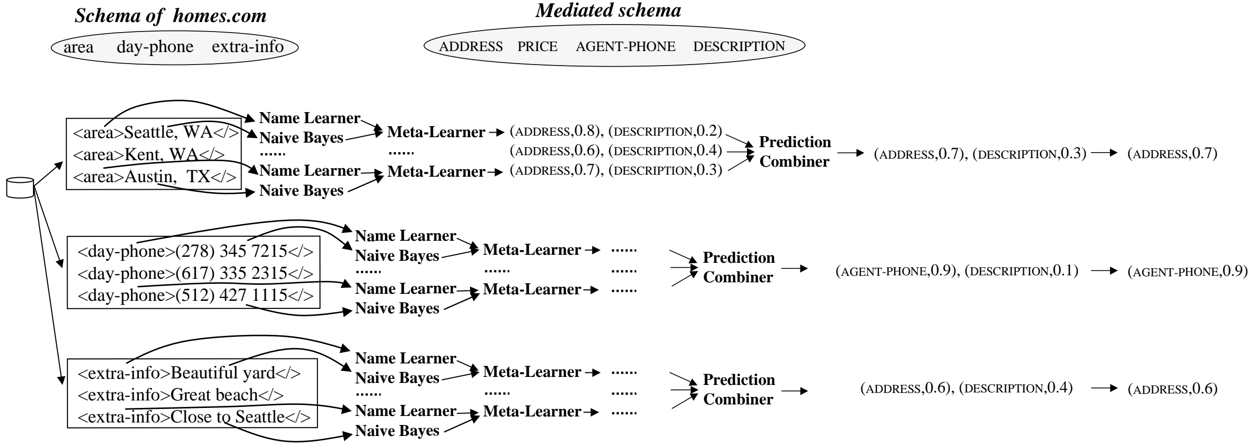


Figure 5: The matching phase of our multistrategy learning approach.

on what meta-learning technique we use. In Section 4.2 we propose a meta-learning technique based on stacking and describe training the meta-learner in detail.

3.2 The Matching Phase

Once the learners have been trained, we are ready to perform schema matching on new sources. Figure 5 illustrates the matching process on source *homes.com*. We begin by extracting a set of house listings from *homes.com*. Next, for each source-schema element we collect all the XML instances of that element (from the extracted listings). Figure 5 shows that for each of the three elements area, day-phone, and extra-info we have collected three XML instances.

We now consider matching each source-schema element in turn. Consider the first element: area. We begin by matching *each XML instance* of the element. Consider the first XML instance: "⟨area⟩Seattle, WA⟨/⟩". To match this instance, we apply the base learners. The Name Learner examines the *name* of the instance, which is "area", and issues a prediction. The Naive Bayes Learner examines the *data value* of the instance, which is "Seattle, WA", and issues another prediction. The meta-learner then combines the two predictions into a single prediction for the first XML instance. This prediction is $\langle(\mathsf{ADDRESS}, 0.8), (\mathsf{DESCRIPTION}, 0.2)\rangle$ (Figure 5). There are many possible ways to combine the base learners' predictions, depending on the particular meta-learning technique being employed. In Section 4.2 we propose and describe in detail a specific meta-learning approach that uses stacking [Wol92, TW99].

After making a prediction for the first XML instance of area, we proceed similarly to for the remaining two instances (note that in all cases the input to the Name Learner is "area"), and obtain two more predictions as shown in Figure 5.

The prediction combiner then combines the three predictions into a single prediction for area. The current prediction combiner simply computes the average score of each label from the given predictions. Thus, in this case it returns (ADDRESS,0.7), (DESCRIPTION,0.3).

We proceed similarly with the other two source-schema elements: day-phone and extra-info. The output from the prediction combiner in these cases are the two additional predictions shown in Figure 5.

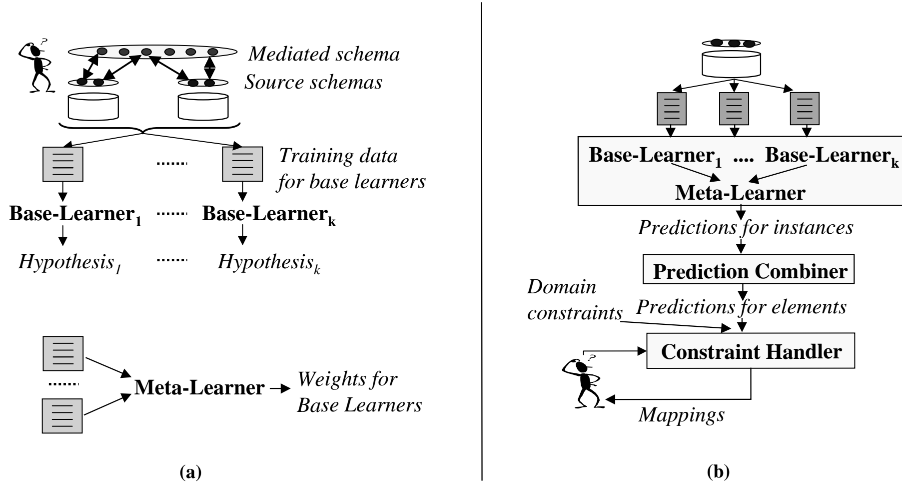


Figure 6: The two phases of LSD.

Finally, we examine the three predictions output by the prediction combiner for the three source-schema elements, and return the labels with the highest confidence scores as the matching labels. For the example in Figure 5, we would predict that area matches ADDRESS, day-phone matches AGENT-PHONE, and extra-info matches ADDRESS. In this particular case, multistrategy learning achieves a matching accuracy of 66%, because out of three predictions, it makes an incorrect one: matching extra-info to ADDRESS. The next section briefly discusses how multistrategy learning is augmented with a variety of techniques, in order to improve matching accuracy.

Remark 1: Figure 5 shows that to match a source-schema element, say area, each base learner takes as input a *row* in the data column for area. One may ask why not apply each base learner directly to the *entire data column*, because we are ultimately interested in matching *columns*, not *rows*. Obviously, for learners that only use schema information (e.g., the name matcher), there is no difference. For learners that use row data, the difference is between having few examples with a long description for each example, and having many examples with a short description for each on. Machine learning algorithms typically work much better in the latter case. Notice that longer example descriptions imply a potentially larger instance space. (For example, if an example is described by n Boolean attributes, the size of the instance space is 2^n.) Learning is easier when the instance space is small and there are many examples. When the instance space is large and there are few examples, it becomes very hard to tell where the frontiers between different classes are. Thus it is preferable to consider each row as a separate example.□

4 The LSD Implementation

We have developed the LSD system that semi-automatically create semantic mappings. LSD implements the proposed multistrategy learning approach. To further improve matching accuracy, LSD extends multistrategy learning to incorporate domain integrity constraints and user feedback, and employs a novel learning technique that exploits the nested structure inherent in XML data.

Figure 6 shows the architecture of LSD. The system consists of four major components: *base learners*, *meta-learner*, *prediction combiner*, and *constraint handler*. It operates in two phases:

training and matching. In the *training phase* LSD first asks the user to manually specify the mappings for several sources. Second, it extracts some data from each source. Third, it creates training examples for the base learners from the extracted data. As we saw in Section 3.1, different base learners require different sets of training examples. Fourth, it trains each base learner on the training examples. Finally, it trains the meta-learner using the training examples of the base learners. The output of the training phase is the internal classification models of the base-learner and the meta-learner.

In the *matching phase* the trained learners are used to match new source schemas. Matching a target source proceeds in three steps. First, LSD extracts some data from the source, and creates for each source-schema element a column of XML elements that belong to it. Second, it applies the base learners to the XML elements in the column, then combines the learners' predictions using the meta-learner and the prediction combiner. Finally, the constraint handler takes the predictions, together with the available domain constraints, and outputs 1-1 mappings for the target schema. The user can either accept the mappings, or provide some feedback and ask the constraint handler to come up with a new set of mappings.

We now describe the base learners and the meta-learner employed by LSD in detail. Then we briefly describe domain constraints and the constraint handler.

4.1 The Base Learners

LSD uses the following base learners:

The Name Learner matches an XML element using its tag name (expanded with synonyms and all tag names leading to this element from the root element). The synonyms are domain specific and are manually created by examining 10-20 randomly selected data listings in the domain. The Name Learner uses Whirl, the nearest-neighbor classification model developed by Cohen and Hirsh [CH98]. This learner stores all training examples of the form (expanded tag-name,label) that it has seen so far. Then given an XML element t, it computes the label for t based on the labels of all examples in its store that are within a distance δ from t. The similarity distance between any two examples is the TF/IDF distance (commonly employed in information retrieval) between the *expanded tag names* of the examples. See [CH98] for more details.

This learner works well on specific and descriptive names, such as price or house_location. It is not good at names that do not share synonyms (e.g., comments and DESCRIPTION), that are partial (e.g., office to indicate office phone), or vacuous (e.g., item, listing).

The Content Learner also uses Whirl. However, this learner matches an XML element using its *data value*, instead of its *tag name* as with the name learner. Therefore, here each example is a pair (data-value,label), and the TF/IDF distance between any two examples is the distance between their data values.

This learner works well on long textual elements, such as house description, or elements with very distinct and descriptive values, such as color (red, blue, green, etc.). It is not good at short, numeric elements such as number of bathrooms and number of bedrooms.

The Naive Bayes Learner is one of the most popular and effective text classifiers [DH74, DP97, MN98]. This learner treats each input instance as a *bag of tokens*, which is generated by parsing

and stemming the words and symbols in the instance. Let $d = \{w_1, \ldots, w_k\}$ be an input instance, where the w_j are tokens. Then the Naive Bayes learner assigns d to the class c_i that maximizes $P(c_i|d)$. This is equivalent to finding c_i that maximizes $P(d|c_i)P(c_i)$. $P(c_i)$ is estimated as the fraction of training instances with label c_i. Assuming that the tokens w_j appear in d *independently* of each other given c_i, we can compute $P(d|c_i)$ as $P(w_1|c_i)P(w_2|c_i)\cdots P(w_k|c_i)$, where $P(w_j|c_i)$ is estimated as $n(w_j, c_i)/n(c_i)$. $n(c_i)$ is the total number of token positions of all training instances with label c_i, and $n(w_j, c_i)$ is the number of times token w_j appears in all training instances with label c_i. Even though the independence assumption is typically not valid, the Naive Bayes learner still performs surprisingly well in many domains (see [DP97] for an explanation).

The Naive Bayes learner works best when there are tokens that are strongly indicative of the correct label, by virtue of their frequencies (e.g., "beautiful" and "great" in house descriptions). It also works well when there are only weakly suggestive tokens, but many of them. It does not work well for short or numeric fields, such as color and zip code.

The County-Name Recognizer searches a database (extracted from the Web) to verify if an XML element is a county name. LSD uses this module in conjunction with the base learners when working on the real-estate domain. This module illustrates how recognizers with a narrow and specific area of expertise can be incorporated into our system.

The XML Learner: As we built LSD, we realized that none of our learners can handle the hierarchical structure of XML data very well. For example, the Naive Bayes learner frequently confused instances of classes HOUSE, CONTACT-INFO, OFFICE-INFO, and AGENT-INFO. This is because the learner "flattens" out all structures in each input instance and uses as tokens only the *words* in the instance. Since the above classes share many words, it cannot distinguish them very well. The content learner faces the same problems.

To address this problem we developed a novel learner that exploits the hierarchical structure of XML data. Our XML learner is similar to the Naive Bayes learner in that it also represents each input instance as a bag of tokens, assumes the tokens are independent of each other given the class, and multiplies the token probabilities to obtain the class probabilities. However, it differs from Naive Bayes in one crucial aspect: it considers not only *text* tokens, but also *structure* tokens that take into account the hierarchical structure of the input instance. For more detail on the XML learner, see [DDH01].

4.2 The Meta-Learner

The meta-learner employed by LSD uses *stacking* [Wol92, TW99] to combine the predictions of the base learners. For ease of exposition, we shall explain the workings of the meta-learner using only two base learners: the Name Learner and the Naive Bayes Learner.

In the variation of stacking that we use [TW99], the goal of training the meta-learner is to compute for each pair of base learner and label a *learner weight* which reflects the importance of the learner with respect to predictions regarding that label.

For example, after training, the meta-learner may assign to the pair of Name Learner and label ADDRESS the weight 0.1 and to the pair of Naive Bayes learner and ADDRESS the weight 0.9 (see the lower part of Figure 7.a). This means that on the basis of training, the meta-learner *trusts* the Naive Bayes learner much more than the name learner in predictions regarding ADDRESS.

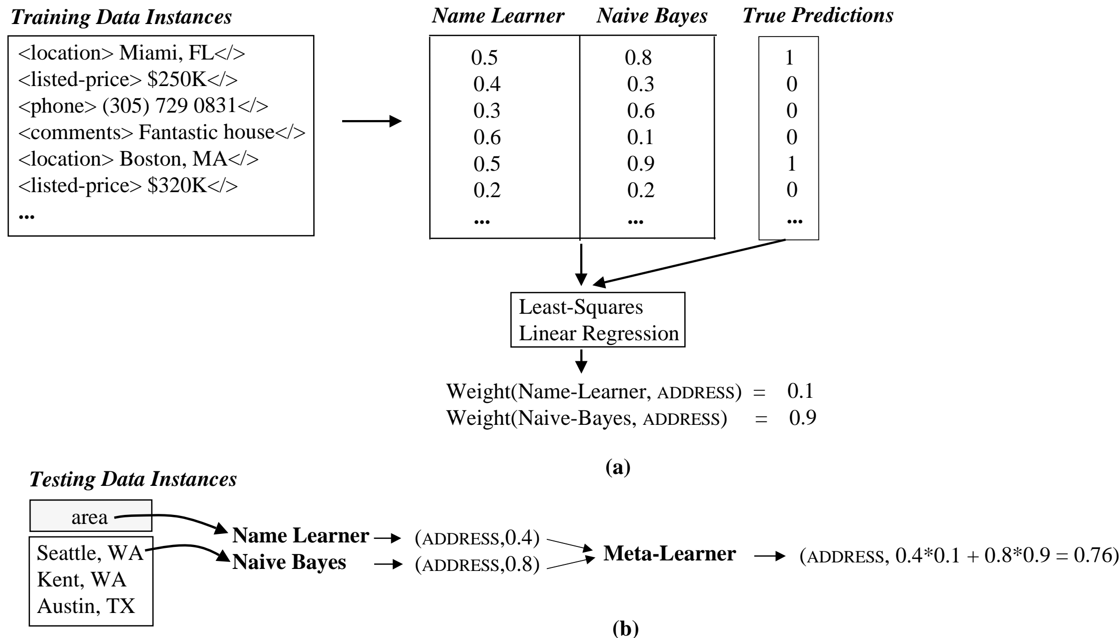


Figure 7: (a) Training the meta-learner to compute the weights for label ADDRESS, and (b) applying the meta-learner.

Once trained, the meta-learner combines the base learners' predictions by returning the average of the predictions, weighted by the learner weights. For example, consider column area in Figure 7.b. This column has three data instances. Consider applying the name learner and the Naive Bayes learner to the first instance: "Seattle, WA". The name learner will examine the instance's name, which is "area", and issue the prediction (ADDRESS,0.4). The Naive Bayes learner will examine the instance's value, which is "Seattle, WA", and issue the prediction (ADDRESS,0.8). The meta-learner then computes the weighted average of the confidence scores: $0.4{*}0.1 + 0.8{*}0.9 = 0.76$ and issues the prediction (ADDRESS,0.76).

Training the Meta-Learner: We now describe how the meta-learner computes the learner weights, one for each pair of base learner and label.

Consider a specific label: ADDRESS. The meta-learner first creates the list of all training data instances. Figure 7.a shows the training data instances, as extracted from source *realestate.com* in Figure 4.

The meta-learner then asks each base learner to predict how well each training data instance match ADDRESS. In Figure 7.a, the column under heading "Name Learner" lists the confidence scores that name learner issues for the training data instances. The first score, 0.5, says the name learner predicts that the first data instance matches ADDRESS with confidence 0.5. Similarly, the column under heading "Naive Bayes" lists the confidence scores that the Naive Bayes learner issues for the training data instances.

A naive approach to generate a column such as the one for the name learner is to have the name learner trained on *all* training data instances, then applied to each data instance. However, this approach biases the name learner because when applied to any instance t, it has already been

trained on t. *Cross validation* is a technique commonly employed in machine learning to prevent such bias. To apply cross validation, the training instances are randomly divided into d equal parts $T_1, T_2, \ldots, T_d$ (we use $d = 5$ in our experiments). Next, for each part T_i, $i \in [1, d]$, the name learner is trained on the remaining $(d-1)$ parts, then applied to the instances in T_i.

Once the columns of confidence scores have been generated, the meta-learner creates a column of "true predictions", which contains a value 1 for each data instance that indeed matches ADDRESS, and 0 otherwise. The meta-learner knows if a data instance matches ADDRESS or not, because it has the manual mappings provided by the user.

Next, the meta-learner performs a least-squares linear regression over the three columns, with "Name Learner" and "Naive Bayes Learner" as predictors and "True Predictions" as the response variable, to compute weights for the name learner and Naive Bayes learner (with respect to ADDRESS). The regression process has the effect that if a base learner tends to output a high confidence that an instance matches ADDRESS when it does and a low confidence when it does not, it will be assigned a high weight, and vice-versa.

Formally, let the confidence scores issued by the name learner be $s_1, s_2, \ldots, s_k$, the scores issued by the Naive Bayes learner be $t_1, t_2, \ldots, t_k$, and the true predictions be $u_1, u_2, \ldots, u_k$. Then the regression finds the learner weights $w^{ADDRESS}_{NameLearner}$ and $w^{ADDRESS}_{NaiveBayes}$ that minimize the sum of square errors $\sum_{i=1}^{k}[u_i - (s_i * w^{ADDRESS}_{NameLearner} + t_i * w^{ADDRESS}_{NaiveBayes})]^2$.

4.3 The Constraint Handler

Consider again the predictions output by the prediction combiner in Figure 5. As we saw in Section 3.2, multistrategy learning returns the labels with the highest confidence scores as the matching labels, and so will report in this specific case that area matches ADDRESS, day-phone matches AGENT-PHONE, and extra-info matches ADDRESS. This result is good but not perfect because two source-schema elements, area and extra-info, have been mapped to ADDRESS.

We know that a house listing should have at most one address. The key motivation behind the constraint handler is to exploit such commonsense domain constraints, as well as other constraints specified on the database schemas (e.g., key and functional dependency constraints), in order to improve matching accuracy.

Note that domain constraints are specified only once, at the beginning, as a part of creating the mediated schema, and independently of any actual source schema. Thus, exploiting domain constraints does not require any subsequent work from the user. Of course, as the user gets to know the domain better, constraints can be added or modified as needed.

The constraint handler takes the domain constraints, together with the predictions produced by the prediction converter, and outputs the 1-1 mappings. Conceptually, it searches through the space of possible mapping combinations (a mapping combination assigns a label to each source-schema element), to find the one with the lowest cost, where cost is defined based on the likelihood of the mapping combination and the degree to which the combination satisfies the domain constraints. For a detailed description of the constraint handler, see [DDH01].

5 Empirical Evaluation

We have evaluated LSD on several real-world domains. Our goals were to evaluate the matching accuracy of LSD and the contribution of different system components.

Domains	Mediated Schema			Sources	Downloaded Listings	Source Schemas			
	Tags	Non-leaf Tags	Depth			Tags	Non-leaf Tags	Depth	Matchable Tags
Real Estate I	20	4	3	5	502 – 3002	19 – 21	1 – 4	1 – 3	84 – 100 %
Time Schedule	23	6	4	5	704 – 3925	15 – 19	3 – 5	1 – 4	95 – 100 %
Faculty Listings	14	4	3	5	32 – 73	13 – 14	4	3	100 %
Real Estate II	66	13	4	5	502 – 3002	33 – 48	11 – 13	4	100 %

Table 1: Domains and data sources for our experiments.

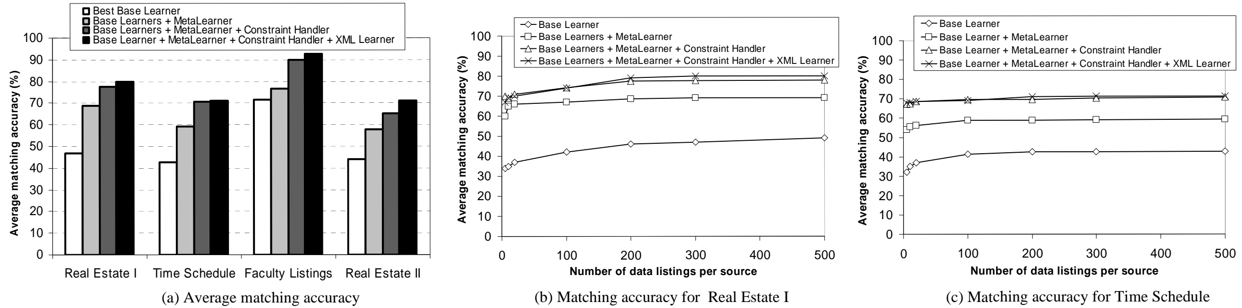
Domains and Data Sources: We report the evaluation of LSD on four domains, whose characteristics are shown in Table 1. Both Real Estate I and Real Estate II integrate sources that list houses for sale, but the mediated schema of Real Estate II is much larger than that of Real Estate I (66 vs. 20 distinct tags). Time Schedule integrates course offerings across universities, and Faculty Listing integrates faculty profiles across CS departments in the U.S.

We began by creating a mediated DTD for each domain. Then we chose five sources on the WWW. We tried to choose sources that had fairly complex structure. Next, since sources on the WWW are not yet accompanied by DTDs, we created a DTD for each source. In doing so, we were careful to mirror the structure of the data in the source, and to use the terms from the source. Then we downloaded data listings from each source. Where possible, we downloaded the entire data set; otherwise, we downloaded a representative data sample by querying the source with random input values. Finally, we converted each data listing into an XML document that conforms to the source schema.

In preparing the data, we performed only trivial data cleaning operations such as removing "unknown", "unk", and splitting "$70000" into "$" and "70000". Our assumption is that the learners LSD employs should be robust enough to handle dirty data. Next we specified integrity constraints for each domain. This task takes approximately 30 min - 2 hours per domain. See [DDH01] for more detail on specifying domain constraints.

Table 1 shows the characteristics of the mediated DTDs, the sources, and source DTDs. The table shows the number of tags (leaf and non-leaf) and maximum depth of the DTD tree for the mediated DTDs. For the source DTDs the table shows the range of values for each of these parameters. The rightmost column shows the percentage of source-DTD tags that have a 1-1 matching with the mediated DTD.

Experiments: For each domain we performed three sets of experiments. First, we measured LSD's accuracy and investigated the utility of multistrategy learning. Second, we examined the sensitivity of accuracy with respect to the amount of data available from each source. Finally, we conducted lesion studies to measure the contribution of each base learner and the constraint handler to the overall performance.



(a) Average matching accuracy (b) Matching accuracy for Real Estate I (c) Matching accuracy for Time Schedule

Figure 8: (a) Average matching accuracy; experiments were run with 300 data listings from each source; for sources from which fewer than 300 listings were extracted, all listings were used. (b)-(c) The average domain accuracy as a function of the amount of data available per source.

Experimental Methodology: To generate the data points shown in the next three sections, we ran each experiment three times, each time taking a new sample of data from each source. In each experiment on a domain we carried out all ten runs in each of which we chose three sources for training and used the remaining two sources for testing. We trained LSD on the training sources, then applied it to match the schemas of the testing sources. The *matching accuracy of a source* is then defined as the percentage of matchable source-schema tags that are matched correctly by LSD. The *average matching accuracy of a source* is its accuracy averaged over all settings in which the source is tested. The *average matching accuracy of a domain* is the accuracy averaged over all five sources in the domain.

5.1 Results

Matching Accuracy: Figure 8.a shows the average matching accuracy for different domains and LSD configurations. For each domain, the four bars (from left to right) represent the average accuracy produced respectively by the best single base learner (excluding the XML learner), the meta-learner using the base learners, the domain constraint handler on top of the meta-learner, and all the previous components together with the XML learner (i.e., the complete LSD system).

The results show that LSD achieves 71 - 92% accuracy across the four domains. Notice that in contrast, the best matching results of the base learners (achieved by either the Naive Bayes or the Name Learner, depending on the domain) are only 42 - 72%.

As expected, adding the meta-learner improves accuracy substantially, by 5 - 22%. Adding the domain-constraint handler further improves accuracy by 7 - 13%. Adding the XML learner improves accuracy by 0.8 - 6.0%. In all of our experiments, the XML learner outperformed the Naive Bayes learner by 3-10%, confirming that the XML learner is able to exploit the hierarchical structure in the data. The results also show that the gains with the XML learner depend on the amount of structure in the domain. For the first three domains, the gains are only 0.8 - 2.8%. In these domains, sources have relatively few tags with structure (4 - 6 non-leaf tags), most of which have been correctly matched by the other base learners. In contrast, sources in the last domain (Real Estate II) have many non-leaf tags (13), giving the XML learner more room to produce improvements (6%).

In Section 6 we identify the reasons that prevent LSD from correctly matching the remaining

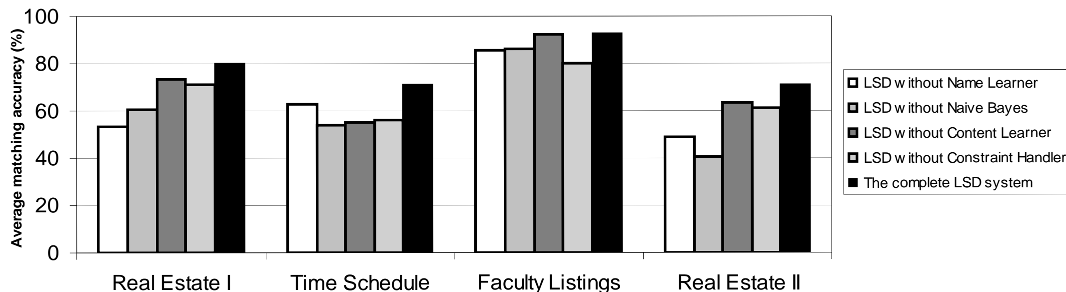


Figure 9: The average matching accuracy of LSD versions with each component being left out versus that of the complete LSD system.

10 - 30% of the tags.

Performance Sensitivity: Figures 8.b-c show the variation of the average domain accuracy as a function of the number of data listings available from each source, for the Real Estate I and Time Schedule domains, respectively. The results show that on these domains the performance of LSD stabilizes fairly quickly: it climbs steeply in the range 5 - 20, minimally from 20 to 200, and levels off after 200. Experiments with other domains show the similar behavior. LSD thus appears to be robust, and can work well with relatively little data. One of the reasons this observation is important is that we can reduce the running time of LSD if we run it on fewer examples.

Lesion Studies: Figure 9 shows the contribution of each base learner and the constraint handler to the overall performance. For each domain, the first four bars (from left to right) represent the average accuracy produced by LSD when one of the components is removed. (The contribution of the XML learner is already shown in Figure 8.a). The fifth bar represents the accuracy of the complete LSD system, for comparison purposes. The results show that each component contributes to the overall performance, and there appears to be no clearly dominant component.

6 Discussion

We now address the limitations of the current LSD system, as well as issues related to system evaluation.

Accuracy: The first issue to address is whether we can increase the accuracy of LSD beyond the current range of 71 - 92%. There are several reasons that prevent LSD from correctly matching the remaining 10 - 30% of the tags. First, some tags (e.g., suburb) cannot be matched because none of the training sources has matching tags that would provide training data. This problem can be handled by adding domain-specific recognizers or importing data from sources outside the domain.

Second, some tags simply require different types of learners. For example, course codes are short alpha-numeric strings that consist of department code followed by course number. As such, a format learner would presumably match them better than any of LSD's current base learners.

Finally, some tags cannot be matched because they are simply ambiguous. For example, given the following source data fragment "course-code: CSE142 section: 2 credits: 3", it is not clear if

"credits" refers to the course- or the section credits. Here, the challenge is to provide the user with a possible *partial* mapping. If our mediated DTD contains a label hierarchy, in which each label (e.g., credit) refers to a concept more general than those of its descendent labels (e.g., course-credit and section-credit) then we can match a tag with the most specific unambiguous label in the hierarchy (in this case, credit), and leave it to the user to choose the appropriate child label.

Efficiency: The training phase of LSD can be carried out offline, so training time is not an issue. In the matching phase, LSD spends most of its time in the constraint handler (typically in the range of seconds to 5 minutes, but sometimes up to 20 minutes in our experiments), though we should note that we did not spend any time on optimizing the code. Since we would like the process of match prediction and incorporation of user feedback to be interactive, we need to ensure that the constraint handler's performance does not become a bottleneck. The most obvious solution is to incorporate some constraints within some early phases to substantially reduce the search space. There are many fairly simple constraints that can be pre-processed, such as constraints on an element being textual or numeric. Another solution is to consider more efficient search techniques (than the A* search technique [HNR72] that we currently use), and to tailor them to our context.

Schema Overlap: In our experiments source schemas overlap substantially with the mediated schema (84-100% of source-schema tags are matchable). This is typically the case for "aggregator" domains, where the data-integration system provides access to sources that offer essentially the same service. We plan to examine other types of domains, where the schema overlap is much smaller. The performance of LSD on these domains will depend largely on its ability to recognize that a certain source-schema tag matches *none* of the mediated-schema tags, even if superficial resemblances are present.

Performance Evaluation: We have reported LSD's performance in terms of the predictive matching accuracy. Predictive accuracy is an important performance measure because (a) the higher the accuracy, the more reduction in human labor the system can achieve, and (b) the measure facilitates comparison and development of schema matching techniques. The next step is to actually *quantify* the reduction in human labor that the system achieves. This step is known to be difficult, due to widely varying assumptions on how a semi-automatic tool such as LSD is used, and has just recently been investigated [MMGR02, DMR02].

Examining Other Meta-Learning Techniques: The meta-learning technique employed by LSD is conceptually easy to understand and appears empirically to work well. Nevertheless, it is important to evaluate the technique further. Moreover, a wealth of meta learning techniques have been developed in the literature (e.g., see [MT94] and other papers in this special issue). Examining the suitability of these meta-learning techniques for schema matching is an important future research problem.

Computing Complex Mappings: LSD focuses on computing 1-1 semantic mappings. Mappings between two schemas, however, often specify non 1-1 relationships, such as name = concat(first-name,last-name) and price = listed-price * (1 + tax-rate). Discovering such complex mappings is therefore an essential feature of any practical schema matching tool. We have begun to extend multistrategy learning to compute such mappings (see [DDH02] for more detail).

Mapping Maintenance: In dynamic and autonomous environments, such as the Internet, sources often undergo changes in their schemas and data. This poses difficult mapping validation problems: how do we know if the semantic mappings computed by LSD would still be valid in, say, three months? We plan to examine such issues, building upon related work in wrapper maintenance [Kus00b].

7 Related Work

We describe work related to LSD from several perspectives.

Schema Matching: Work on schema matching can be classified into rule- and learner-based approaches. (For a comprehensive survey of schema matching, see [RB01].) Rule-based approach includes [MZ98, PSU98, CA99]. The Transcm system [MZ98] performs matching based on the *name* and *structure* of schema elements. The Artemis system [CA99] uses names, structures, as well as domain types of schema elements to match schemas. In general, rule-based systems utilize only schema information in a hard-coded fashion, whereas our approach exploits both schema and data information, and does so automatically, in an extensible fashion.

In the learner-based approach, the Semint system [LC00] uses a neural-network learner. It matches schema elements using properties such as field specifications (e.g., data types and scale) and statistics of data content (e.g., maximum, minimum, and average). Unlike LSD, it does not exploit other types of data information such as word frequencies and field formats. The ILA system [PE95] matches schemas of two sources based on comparing objects that it knows to be the same in both sources. Both Semint and ILA employ a single type of learner, and therefore have limited applicability. For example, the neural net of Semint does not deal well with textual information, and in many domains it is not possible to find overlaps in the sources, making ILA's method inapplicable.

Clifton et al. [CHR97] describe DELTA, which associates with each attribute a text string that consists of all meta-data on the attribute, then matches attributes based on the similarity of the text strings. The authors also describe a case study using DELTA and Semint and note the complementary nature of the two methods. With LSD, both Semint and DELTA could be plugged in as new base learners, and their predictions would be combined by the meta-learner.

The Clio system [MHH00] introduces *value correspondences*, which specify functional relationships among related elements (e.g., hotel-tax = room-rate $*$ state-tax-rate). Given a set of such correspondences, Clio produces the SQL queries that translate data from one source to the other. A key challenge in creating the queries is to find semantically meaningful ways to relate the data elements in a source, in order to produce data for the other source. For example, given the above value correspondence, Clio must discover that room-rate belongs to a hotel that is located in the state where the tax rate is state-tax-rate. To this end, Clio explores joining elements along foreign-key relationship paths. There can be many foreign keys, thus many possible ways to join. Hence, the problem boils down to searching for the most likely join path. Clio uses several heuristics and user feedback to arrive at the best join path, and thus the best query candidate. Clio is therefore complementary to the current work of ours. It can take the mappings produced by LSD as the value correspondences.

Ontology Matching: Many works have addressed ontology matching in the context of ontology design and integration (e.g., [Cha00, MFRW00, NM00, MWJ98]). They use a variety of heuristics to match ontology elements. They do not use machine learning and do not exploit information in the data instances. However, most of them [Cha00, MFRW00, NM00] have powerful features that allow for efficient user interaction. Such features are important components of a comprehensive solution to matching ontologies and schemas, and hence should be added to LSD in the future.

Several recent works have attempted to further automate the ontology matching process. The Anchor-PROMPT system [NM01] exploits the general heuristic that paths (in the taxonomies or ontology graphs) between matching elements tend to contain other matching elements. The HICAL system [RHS01] exploits the data instances in the overlap between the two taxonomies to infer mappings. [LG01] computes the similarity between two taxonomic nodes based on their signature TF/IDF vectors, which are computed from the data instances.

Combining Multiple Learners: multistrategy learning has been researched extensively [MT94], and applied to several other domains (e.g., information extraction [Fre98], solving crossword puzzles [KSL+99], and identifying phrase structure in NLP [PR00]). In our context, our main innovations are the three-level architecture (base learners, meta-learner and prediction combiner) that allows learning from both schema and data information, the use of integrity constraints to further refine the learner, and the XML learner that exploits the structure of XML documents.

Exploiting Domain Constraints: Incorporating domain constraints into learners has been considered in several works (e.g., [DR96]), but most consider only certain types of learners and constraints. In contrast, our framework allows arbitrary constraints (as long as they can be verified using the schema and data), and works with any type of learner. This is made possible by using the constraints during the matching phase, to restrict the learner predictions, instead of the usual approach of using constraints during the training phase, to restrict the search space of learned hypotheses.

8 Conclusions & Future Work

We have described an approach to schema matching that employs and extends machine learning techniques. Our approach utilizes both schema and data from the sources. To match a source-schema element, the system applies a set of learners, each of which looks at the problem from a different perspective, then combines the learners' predictions using a meta-learner. The meta-learner's predictions are further improved using domain constraints and user feedback. We also developed a novel XML learner that exploits the hierarchical structure in XML data to improve matching accuracy. Our experiments show that we can accurately match 71-92% of the tags on several domains.

More broadly, we believe that our approach contributes an important aspect to the development of schema-matching solutions. Given that schema matching is a fundamental step in numerous data management applications [RB01], it is desirable to develop a generic solution that is robust and applicable across domains. In order to be robust, such a solution must have several important properties. First, it must *improve over time* – knowledge gleaned from previous instances of the schema-matching problem should contribute to solving subsequent instances. Second, it must allow knowledge to be incorporated in an *incremental* fashion, so that as the user gets to know the domain

better, knowledge can be easily modified or added. Third, it must allow *multiple types* of knowledge to be used to maximize the matching accuracy. Machine learning techniques, and in particular, multistrategy learning, provide a basis for supporting these properties. Hence, while we do not argue that our techniques provide a complete solution to the schema-matching problem, we do believe that we provide an indispensable component of any robust solution to the problem.

We have been extending our multistrategy learning approach to compute non 1-1 mappings [DDH02], to match ontologies [DMDH02], a crucial task in constructing large-scale knowledge bases and processing information on the Semantic Web, and to develop a formal semantics for mapping [MHDB02]. We are continuing our work on these issues, as well as issues outlined in the discussion section. The most up-to-date information on LSD can be found on its website [lsd]. The site also stores a public repository of data intended to be used as benchmarks in evaluating schema matching algorithms.

Acknowledgments: We thank Phil Bernstein, Oren Etzioni, David Hsu, Geoff Hulten, Zack Ives, Jayant Madhavan, Erhard Rahm, Igor Tatarinov, and the reviewers for invaluable comments. This work was supported by NSF Grants 9523649, 9983932, IIS-9978567, and IIS-9985114. The second author was also supported by an IBM Faculty Partnership Award, and the third author was supported by a Sloan Fellowship and gifts from Microsoft Research, NEC and NTT.

References

[AK97] Naveen Ashish and Craig A. Knoblock. Wrapper generation for semi-structured internet sources. *SIGMOD Record*, 26(4):8–15, 1997.

[CA99] S. Castano and V. De Antonellis. A schema analysis and reconciliation tool environment for heterogeneous databases. In *Proc. of the Int. Database Engineering and Applications Symposium (IDEAS-99)*, pages 53–62, 1999.

[CH98] W. Cohen and H. Hirsh. Joins that generalize: Text classification using WHIRL. In *Proc. of the Fourth Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 1998.

[Cha00] H. Chalupsky. Ontomorph: A translation system for symbolic knowledge. In *Principles of Knowledge Representation and Reasoning*, 2000.

[CHR97] C. Clifton, E. Housman, and A. Rosenthal. Experience with a combined approach to attribute-matching across heterogeneous databases. In *Proc. of the IFIP Working Conference on Data Semantics (DS-7)*, 1997.

[DDH01] A. Doan, P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. In *Proceedings of the ACM SIGMOD Conference*, 2001.

[DDH02] A. Doan, P. Domingos, and A. Halevy. Learning complex mapping between structured representations. Technical Report UW-CSE-2002, University of Washington, 2002.

[DH74] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1974.

[DMDH02] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map ontologies on the Semantic Web. In *Proceedings of the World-Wide Web Conference (WWW-02)*, 2002.

[DMR02] H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the 2nd Int. Workshop on Web Databases (German Informatics Society)*, 2002.

[DP97] P. Domingos and M. Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29:103–130, 1997.

[DR96] S. Donoho and L. Rendell. Constructive induction using fragmentary knowledge. In *Proc. of the 13th Int. Conf. on Machine Learning*, pages 113–121, 1996.

[Fre98] Dayne Freitag. Machine learning for information extraction in informal domains. *Ph.D. Thesis*, 1998. Dept. of Computer Science, Carnegie Mellon University.

[FW97] M. Friedman and D. Weld. Efficiently executing information-gathering plans. In *Proc. of the Int. Joint Conf. of AI (IJCAI)*, 1997.

[GMPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Inf. Systems*, 8(2), 1997.

[HGMN+98] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, and V. Vassalos. Template-based wrappers in the TSIMMIS system (system demonstration). In *ACM Sigmod Record*, Tucson, Arizona, 1998.

[HNR72] P. Hart, N. Nilsson, and B. Raphael. Correction to "a formal basis for the heuristic determination of minimum cost paths". *SIGART Newsletter*, 37:28–29, 1972.

[IFF+99] Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld. An adaptive query execution system for data integration. In *Proc. of SIGMOD*, 1999.

[KMA+98] C. Knoblock, S. Minton, J. Ambite, N. Ashish, P. Modi, I. Muslea, A. Philpot, and S. Tejada. Modeling web sources for information integration. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 1998.

[KSL+99] G. Keim, N. Shazeer, M. Littman, S. Agarwal, C. Cheves, J. Fitzgerald, J. Grosland, F. Jiang, S. Pollard, and K. Weinmeister. PROVERB: The probabilistic cruciverbalist. In *Proc. of the 6th National Conf. on Artificial Intelligence (AAAI-99)*, pages 710–717, 1999.

[Kus00a] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1–2):15–68, 2000.

[Kus00b] N. Kushmerick. Wrapper verification. *World Wide Web Journal*, 3(2):79–94, 2000.

[LC00] W. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondence in heterogeneous databases using neural networks. *Data and Knowledge Engineering*, 33:49–84, 2000.

[LG01] M. Lacher and G. Groh. Facilitating the exchange of explicit knowledge through ontology mappings. In *Proceedings of the 14th Int. FLAIRS conference*, 2001.

[LKG99] E. Lambrecht, S. Kambhampati, and S. Gnanaprakasam. Optimizing recursive information gathering plans. In *Proc. of the Int. Joint Conf. on AI (IJCAI)*, 1999.

[LRO96] A. Y. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, 1996.

[lsd] LSD's website: cs.washington.edu/homes/anhai/lsd.html.

[MFRW00] D. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera Ontology Environment. In *Proceedings of the 17th National Conference on Artificial Intelligence*, 2000.

[MHDB02] J. Madhavan, A. Halevy, P. Domingos, and P. Bernstein. Representing and reasoning about mappings between domain models. In *Proceedings of the National AI Conference (AAAI-02)*, 2002.

[MHH00] R. Miller, L. Haas, and M. Hernandez. Schema mapping as query discovery. In *Proc. of VLDB*, 2000.

[MMGR02] S. Melnik, H. Molina-Garcia, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2002.

[MN98] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[MT94] R. Michalski and G. Tecuci, editors. *Machine Learning: A Multistrategy Approach.* Morgan Kaufmann, 1994.

[MWJ98] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic Integration of Knowledge Sources. In *Proceedings of Fusion'99*, 1998.

[MZ98] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. of VLDB*, 1998.

[NM00] N.F. Noy and M.A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2000.

[NM01] N.F. Noy and M.A. Musen. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. In *Proceedings of the Workshop on Ontologies and Information Sharing at the International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

[PE95] M. Perkowitz and O. Etzioni. Category translation: Learning to understand information on the Internet. In *Proc. of Int. Joint Conf. on AI (IJCAI)*, 1995.

[PR00] V. Punyakanok and D. Roth. The use of classifiers in sequential inference. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS-00)*, 2000.

[PSU98] L. Palopoli, D. Sacca, and D. Ursino. Semi-automatic, semantic discovery of properties from database schemes. In *Proc. of the Int. Database Engineering and Applications Symposium (IDEAS-98)*, pages 244–253, 1998.

[RB01] E. Rahm and P.A. Bernstein. On matching schemas automatically. *Tech. report MSR-TR-2001-17*, 2001. Microsoft Research, Redmon, WA.

[RHS01] I. Ryutaro, T. Hideaki, and H. Shinichi. Rule Induction for Concept Hierarchy Alignment. In *Proceedings of the 2nd Workshop on Ontology Learning at the* 17th *Int. Joint Conf. on AI (IJCAI)*, 2001.

[TW99] K. M. Ting and I. H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[Wol92] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[XML98] Extensible markup language (XML) 1.0. www.w3.org/TR/1998/REC-xml-19980210, 1998. W3C Recommendation.