Discovering Direct and Indirect Matches for Schema Elements

Li Xu[*] and David W. Embley[*]
Department of Computer Science
Brigham Young University
Provo, Utah 84602, U.S.A.
{lx, embley}@cs.byu.edu

Abstract

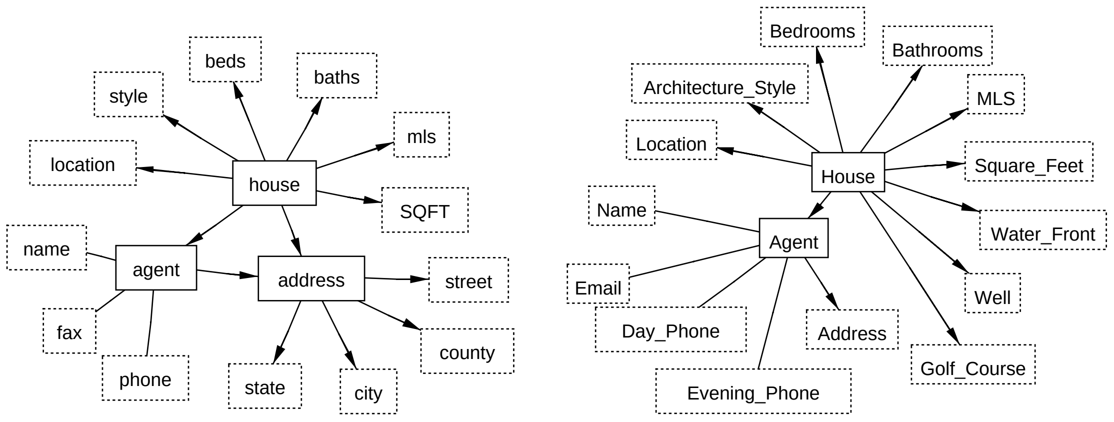
Automating schema matching is challenging. Previous approaches (e.g. [MBR01, DDH01]) to automating schema matching focus on computing direct element matches between two schemas. Schemas, however, rarely match directly. Thus, to complete the task of schema matching, we must also compute indirect element matches. In this paper, we present a framework for generating direct as well as many indirect element matches between a target schema and a source schema. Recognizing expected data values associated with schema elements and applying schema-structure heuristics are the key ideas to computing indirect matches. Experiments we have conducted over several real-world application domains show encouraging results.
Keyword: Schema matching, data integration, schema integration, data exchange.

1 Introduction

In this paper, we focus on the long-standing and challenging problem of automating schema matching [MBR01]. Schema matching is a key operation for many applications including data integration, schema integration, message mapping in E-commerce, and semantic query processing [RB01]. Schema matching takes two schemas as input and produces a semantic correspondence between the schema elements in the two input schemas [RB01]. In this paper, we assume that we wish to map schema elements from a *source* schema into a *target* schema. In its simplest form, the semantic correspondence is a set of *direct element matches* each of which binds a source schema element to a target schema element if the two schema elements are semantically equivalent. To date, most research [DDH01, EJX01, MBR01, LC00, MZ98, PTU00, BCV99] has focused on computing direct element matches. Such simplicity, however, is rarely sufficient, and researchers have thus proposed the use of queries over source schemas to form virtual schema elements to bind with target schema elements [MHH00, BE02]. In this more complicated form, the semantic correspondence is a set of *indirect element matches* each of which binds a virtual source schema element to a target schema element through appropriate *manipulation operations* over a source schema.

We assume that all source and target schemas are described using rooted conceptual-model graphs (a conceptual generalization of XML). Element nodes either have associated data values or associated object identifiers, which we respectively call *value schema elements* and *object schema elements*. We augment schemas with a variety of ontological information. For this paper the augmentations we discuss are WordNet [Mil95], sample data, and regular-expression recognizers. For each application, we construct a lightweight domain ontology [ECJ$^+$99], which declares the regular-expression recognizers. We use the regular-expression recognizers to discover both direct and indirect matches between two arbitrary schemas. Based on the graph structure and these

[*]This material is based upon work supported by the National Science Foundation under grant IIS-0083127.



(a) Schema 1 (b) Schema 2

Figure 1: Schema Graphs for Schema 1 and Schema 2

augmentations, we exploit a broad set of techniques together to settle direct and indirect element matches between a target schema and a source schema. As will be seen, regular-expression recognition and schema structure are the key ways to detect indirect element matches.

In this paper, we offer the following contributions: (1) a way to discover many indirect semantic correspondences between a target schema T and a source schema S as well as the direct correspondences and (2) experimental results of our implementation to show the performance of our approach. We present the details of our contribution as follows. Section 2 explains what we mean by direct and indirect matches between T and S. Section 3 describes a set of basic matching techniques to find potential element matches between elements in T and elements in S, and to provide confidence measures between 0 (lowest confidence) and 1 (highest confidence) for each potential match. Section 4 presents an algorithm to settle direct and indirect matches between T and S. Section 5 gives experimental results for a data set used in [DDH01] to demonstrate the success of our approach. In Section 6 we review related work, and in Section 7 we summarize, consider future work, and draw conclusions.

2 Source-to-Target Mappings

We represent all source and target schemas using rooted conceptual-model graphs. Nodes of the graph denote object and value schema elements, and edges of the graph denote relationships among object and value schema elements. The root node is a designated object of primary interest. Figure 1, for example, shows two schema graphs, each partially describing two real-estate applications. In a schema graph we denote value schema elements as dotted boxes, object schema elements as solid boxes, functional relationship as lines with an arrow from domain to range, and nonfunctional relationship as lines without arrowheads.

The output of schema matching is a set of element mappings that match actual or virtual source schema elements with fixed target schema elements. Our source-to-target mappings allow for a variety of source derived data, including missing generalizations and specializations, merged and split values, and transformation of attributes with Boolean indicators into values.

We say that a match (t, s) is *direct* when a target schema element t and a source schema element s denote the same set of values or objects. To detect direct matches, researchers typically look for synonym matches between names of schema elements. Sometimes, however, the identification of synonyms is not enough. For example, *location* in Figure 1(a) is the lot description for a listed property, and *Location* in Figure 1(b) is the location address of a selling house. Our approach considers both schema information and data instances to help settle direct element matches, and thus largely avoids this problem of being misled by polysemy.

Although a source may not have a schema element that directly matches a target element, target facts may nevertheless be derivable from source facts. We call these correspondences *indirect* matches. When trying to detect indirect matches, we consider the following problems, which we illustrate using the schemas in Figure 1.

1. *Generalization* and *Specialization.* Two elements, *Day_Phone* and *Evening_Phone* in Figure 1(b) are both specializations of *phone* values in Figure 1(a). Thus, if Figure 1(a) is the target, we need the union of *Day_Phone* and *Evening_Phone*, and if Figure 1(b) is the target, we should find a way to separate the day phones from the evening phones.

2. *Merged and Split Values.* Four elements, *street*, *county*, *city*, and *state* are separate in Figure 1(a) and merged as *Location* of houses or *Address* of agents in Figure 1(b). Thus, we need to split the values if Figure 1(a) is the target and merge the values if Figure 1(b) is the target.

3. *Schema Element Name as Value.* In Figure 1(b), the features $Water\_Front$ and $Golf\_Course$ are all schema element names rather than values. The Boolean values "Yes" and "No" associated with them are not the values but indicate whether the values $Water\_Front$ and $Golf\_Course$ should be included as description values for *location* in Figure 1(a).

Currently, we use four operations over source schemas to resolve these problems.

1. *Selection.* The data values associated with a target schema element are a subset of the values associated with a source schema element.

2. *Union.* The data values associated with a target schema element are a superset of the values associated with a source schema element (usually several source schema elements). *Union* is the inverse of *Selection.*

3. *Composition.* The values associated with a target schema element match a concatenation of values from two or more source schema elements.

4. *Decomposition.* The values associated with target schema elements match a decomposition of values of a source schema element. *Decomposition* is the inverse of *Composition.*

The recognition and specification of these operations depend on the matching techniques we describe in Sections 3 and 4. Generating operations for *Merged* and *Split Values* and for *Subsets* and *Supersets* is straightforward if we can recognize the types of matches required. For *Schema Element Name as Value*, the resolution depends on being able to recognize the element name as a potential target value. Then, in harmony with the source values (e.g. "Yes"/"No"), we can determine the mapping—either as a direct mapping or an indirect mapping.

3 Matching Techniques

In this section we explain our four basic techniques for matching: (1) terminological relationships (e.g. synonyms and hypernyms), (2) data-value characteristics (e.g. string lengths and alphanumeric ratios), (3) domain-specific, regular-expression matches (i.e. the appearance of expected

```
f3 <= 0: NO (222.0/26.0)
f3 > 0
|   f2 <= 2: YES (181.0/3.0)
|   f2 > 2
|   |   f4 <= 11
|   |   |   f2 <= 5: YES (15.0/5.0)
|   |   |   f2 > 5: NO (14.0/6.0)
|   |   f4 > 11: NO (17.0/2.0)
```

Figure 2: Generated WordNet Rule

strings), and (4) structure (e.g. structural similarities). For the first two techniques we obtain vectors of measures for the features of interest and then apply machine learning over these feature vectors to generate a decision rule and a measure of confidence for each generated decision. We use C4.5 [Qui93] as our decision-rule and confidence-measure generator.

3.1 Terminological Relationships

The meaning of element names provides a clue about which elements match. To match element names, we use WordNet [Mil95, Fel98] which organizes English words into synonym and hypernym sets. Other researchers have also suggested using WordNet to match attributes (e.g. [BCV99, CA99]), but have given few, if any, details.

Initially we investigated the possibility of using 27 available features of WordNet in an attempt to match a token A appearing in the name of a source schema element s with a token B appearing in the name of an target schema element t. The C4.5-generated decision tree, however, was not intuitive.[1] We therefore introduced some bias by selecting only those features we believed would contribute to a human's decision to declare a potential attribute match, namely (f0) same word (1 if $A = B$ and 0 otherwise), (f1) synonym (1 if "yes" and 0 if "no"), (f2) sum of the distances of A and B to a common hypernym ("is kind of") root (if A and B have no common hypernym root, the distance is defined as a maximum number in the algorithm), (f3) the number of different common hypernym roots of A and B, and (f4) the sum of the number of senses of A and B. For our training data we used 222 positive and 227 negative A-B pairs selected from attribute names found in database schemas, which were readily available to us, along with synonym names found in dictionaries. Figure 2 shows the resulting decision tree. Surprisingly, neither f0 (same word) nor f1 (synonym) became part of the decision rule. Feature f3 dominates—when WordNet cannot find a common hypernym root, the words are not related. After f3, f2 makes the most difference—if two words are closely related to the same hypernym root, they are a good potential match. (Note that f2 covers f0 and f1 because both identical words and direct synonyms have zero distance to a common root; this helps mitigate the surprise about f0 and f1.) Lastly, if the number of senses is too high ($f4 > 11$), a pair of words tends to match almost randomly; thus the C4.5-generated rule rejects these pairs and accepts fewer senses only if pairs are reasonably close ($f2 <= 5$) to a common root.

The parenthetical numbers (x/y) following "YES" and "NO" for a decision-tree leaf L give the total number of training instances x classified for L and the number of incorrect training instances

[1] An advantage of decision-tree learners over other machine learning (such as neural nets) is that they generate results whose reasonableness can be validated by a human.

y classified for L. Based on the trained decision rule in Figure 2, we compute a confidence value, denoted $conf_1(t,s)$, where t is a target schema element and s is a source schema element. However, we want the feature f0 (same word) to dominate the others and assign a perfect confidence value (1.0) for two tokens if f0 holds. When schema element names are abbreviations, we expand them so that WordNet can recognize them. If the names of both t and s are single-word tokens, the computation of $conf_1(t,s)$ is straightforward based on the decision rule when f0 does not hold. For a "YES" leaf L, we compute confidence factors by the formula $(x\text{-}y)/x$ where x is the total number of training instances classified for L and y is the number of incorrect training instances classified for L. For a "NO" leaf, the confidence factor is 1-$(x\text{-}y)/x$, which converts "NO's" into "YES's" with inverted confidence values. If a schema element name is a phrase instead of a single-word token, we select nouns from the phrase. Then if either t or s has a name consisting of multiple noun tokens, we use an injective greedy assignment algorithm to locate the potential matching tokens between the name phrases of t and s. We compute $conf_1(t,s)$ as the average of the confidence values collected from the potential matching tokens obtained from the injective greedy algorithm.

Assuming Schema 1 in Figure 1(a) is a target schema, and Schema 2 in Figure 1(b) is a source schema, when we apply the test for terminological relationships of schema element names, the confidence value $conf_1(t,s)$ is high for the matches such as {*house*, *House*}, {*beds*, *Bedrooms*}, {*baths*, *Bathrooms*}, {*phone*, *Day_Phone*}, and {*phone*, *Evening_Phone*}, as it should be. Also, the confidence of {*location*, *Location*} is high, even though the meaning is entirely different; but, as we shall see, other techniques can sort our this anomaly.

3.2 Data-Value Characteristics

Whether two sets of data have similar value characteristics provides another a clue about which elements match. Previous work in [LC00] shows that this technique can successfully help match elements by considering such characteristics as string-lengths and alphabetic/non-alphabetic ratios of alphanumeric data and means and variances of numerical data. We use features similar to those in [LC00], but generate a C4.5 decision rule rather than a neural-net decision rule. Based on the decision rule, which turns out to be lengthy but has a form similar to the decision tree in Figure 2, we generate a confidence value, denoted $conf_2(t,s)$, for each element pair (t, s) of value schema elements.

Testing the decision rule using data values associated with Schema 1 in Figure 1(a) as a target schema and Schema 2 in Figure 1(b) as a source schema, the confidence value $conf_2(t,s)$ is high for the matches such as {*beds*, *Bedrooms*}, {*baths*, *Bathrooms*}, {*phone*, *Day_Phone*}, and {*fax*, *Day_Phone*} as expected. However, *mls* in the target and *Location* in the source tend to look alike according to the value characteristics measured, a surprise which needs other techniques to find the difference. Interestingly, the lot features in *location* of the target schema and the house locations in *Location* of the source schema do not have similar value characteristics; this is because their alphabetic/non-alphabetic ratios are vastly different, as they should be.

3.3 Expected Data Values

Whether expected values appear in a set of data provides yet another clue about which elements match. For a specific application, we can specify a lightweight domain ontology [ECJ+99], which includes a set of concepts and relationship sets among the concepts, and associates with each concept a set of regular expressions that matches values and keywords expected to appear for the value concept. Then, using techniques described in [ECJ+99], we can extract values from sets of data associated with source and target value elements and categorize their data-value patterns

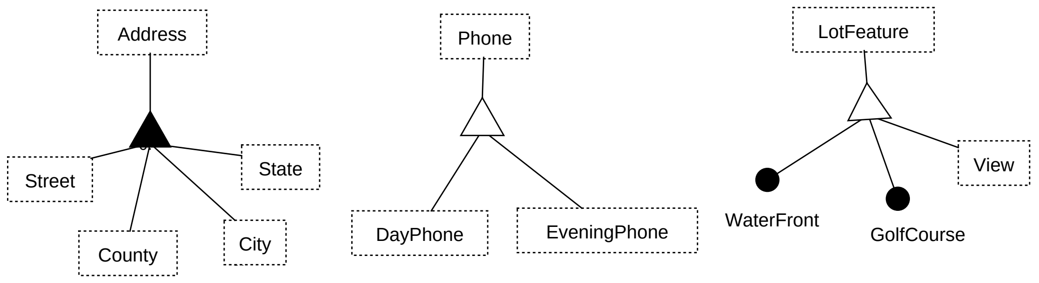


Figure 3: Application Domain Ontology (Partial)

based on the regular expressions declared for application concepts. The derived data-value patterns and the declared relationship sets among concepts in the domain ontology can help discover both direct and indirect matches for schema elements.

We declare the concepts and relationship sets in our lightweight domain ontologies independently of any target and source schemas. We call them lightweight for two reasons. (1) The construction of concepts and relationships is not the same as the construction of a conceptual schema in global-as-view approaches [Ull97] for integrating heterogeneous information sources. A global-as-view information-integration system maintains a global schema, and the system needs to update the global schema when new information sources enter the system. Thus, the global-as-view approach requires that the global schema should be complete in the sense that it embodies all the contents in the underlying information sources. We neither require nor expect that the knowledge declared in an application domain ontology is complete for the application. Moreover, (2) the objective of the regular expressions declaring expected values for application concepts is to discover corresponding concepts, not to extract items of interest [ECJ$^+$99]. Since the domain ontology need not be as complete nor as exact as the declarations for a data-extraction ontology, we see our domain ontologies as being lightweight.

Figure 3 shows three components in our real-estate domain ontology, which we used to automate matching of the two schemas in Figure 1 and also for matching real-world schemas in the real-estate domain. The three components include an address component specifying *Address* as potentially consisting of *State*, *City*, *County*, and *Street*;[2] a phone component specifying *Phone* as a possible superset of *DayPhone*, and *EveningPhone*;[3] and a lot-feature component specifying *LotFeature* as a possible superset of $View$ values and individual values $WaterFront$ and $GolfCourse$.[4] Behind a dotted box (or individual value), a regular-expression recognizer [ECJ$^+$99] describes the expected data values for a potential application concept. The ontology explicitly declares that (1) the expected values associated with *Address* match with a concatenation of the expected values for *Street*, *County*, *City* and *State*; (2) the set of values associated with *Phone* is a superset of the values associated with concepts *DayPhone* and *EveningPhone*; and (3) the set of values associated with *LotFeature* is a superset of the values associated with the set $View$ and the singleton-sets $WaterFront$ and $GolfCourse$.

Provided with the domain ontology just described and a set of data values associated with value elements in Schema 1 in Figure 1(a) and Schema 2 in Figure 1(b), we can discover indirect matches as follows. (We first explain the idea with examples and then more formally explain how this works

[2]Filled-in (black) triangles denote aggregation ("part-of" relationships).
[3]Open (white) triangles denote generalization/specialization ("ISA" supersets and subsets).
[4]Large black dots denote individual objects or values.

in general.)

1. *Composition* and *Decomposition.* Based on the *Address* declared in the ontology in Figure 3, the recognition-of-expected-values technique [ECJ+99] can help detect that (1) the values of both *Address* and *Location* in Schema 2 match with the ontology concept *Address*, and (2) the values of *street*, *county*, *city*, and *state* in Schema 1 match with the ontology concepts *Street*, *County*, *City*, and *State* respectively. Thus, if Schema 1 is the target and Schema 2 is the source, we can use *Decomposition* over *Address* and *Location* in the source to indirectly match with *street*, *county*, *city*, and *state* in the target. If we switch and let Schema 1 be the source and Schema 2 be the target, based on the same information, we can identify the same set of indirect matching element pairs except that the manipulation operation becomes *Composition.*

2. *Union* and *Selection.* Based on the specification of the regular expression matched for *Phone*, the schema elements *Day_Phone* and *Evening_Phone* in Schema 2 match with the concepts *DayPhone* and *EveningPhone* respectively, and *phone* in Schema 1 also matches with the concept *Phone*. *Phone* in the ontology explicitly declares that the set of expected values of *Phone* is a superset of the expected values of *DayPhone* and *EveningPhone*. Thus, we are able to identify the indirect matching schema elements between *phone* in Schema 1 and *Day_Phone* and *Evening_Phone* in Schema 2. If Schema 1 is the target and Schema 2 is the source, we can apply a *Union* operation over Schema 2 to derive a virtual schema element $Phone'$, which can directly match with *phone* in Schema 1. If Schema 1 is the source and Schema 2 is the target, we may be able to recognize keywords such as *day-time*, *day*, *work phone*, *evening*, and *home* associated with each listed phone in the source. If so, we can use a *Selection* operation to sort out which phones belong in which specialization (if not, a human expert may not be able to sort these out either).

3. *Schema Element Name as Value.* Because regular-expression recognizers can recognize schema element names as well as values, the recognizer for $LotFeature$ will recognize names such as $Water\_Front$ and $Golf\_Course$ in Schema 2 as values. Moreover, the recognizer for $LotFeature$ can also recognize data values associated with *location* in Schema 1 such as *Mountain View*, *City Overlook*, and *Water-Front Property*. Thus, when Schema 2 is the target and Schema 1 is the source, whenever we match a source-schema-element name with a target *location* value, we can declare "Yes" as the value for the matching target concept. If, on the other hand, Schema 1 is the target and Schema 2 is the source, we can declare that the schema element name should be a value for *location* for each "Yes" associated with the matching source element.

More formally, let c_i be an application concept, such as *Street*, and consider a concatenation of concepts such as *Address* components. Suppose the regular expression for concept c_i matches the first part of a value v for a value schema element and the regular expression for concept c_j matches the last part of v, then we say that the concatenation $c_i \circ c_j$ matches v. In general, we may have a set of concatenated concepts C_t match a target element t and a set of concatenated concepts C_s match a source element s. For each concept in C_t or in C_s, we have an associated hit ratio. The hit ratios give the percentage of t or s values that match (or are included in at least some match) with the values of the concepts in C_t or C_s respectively. We also have a hit ratio r_t associated with C_t, which gives the percentage of t values that match the concatenation of concepts in C_t, and a hit ratio r_s associated with C_s, which gives the percentage of s values that match the concatenation of concepts in C_s. To obtain hit ratios for Boolean fields recognized as schema-element names, we distribute the schema-element names over all the Boolean fields.

We decide if s matches with t directly or indirectly by comparing C_t and C_s. If C_t equals C_s, we declare a *direct* match (t, s). Otherwise, if $C_t \subset C_s$, we derive an *indirect* match (t, s) through a *Decomposition* operation. If both C_t and C_s contain one individual concept c_t and c_s respectively, and if the values of concept c_t are declared as a subset of the values of concept c_s, we derive an *indirect* match (t, s) through a *Selection* operation. Similarly, we can detect indirect matches associated with *Composition* and *Union* operations. When we have schema-element names as values, distribution of the name over the Boolean value fields converts these schema elements into standard schema elements with conventional value-populated fields. Thus no additional comparisons are needed to detect direct and indirect matches when schema-element names are values.[5] We compute the confidence value for (t, s), which we denoted as $conf_3(t,s)$, as follows. If we can declare a direct match or derive an indirect match through manipulating *Union*, *Selection*, *Composition*, and *Decomposition* for (t, s), and the hit ratios r_t and r_s are above an accepted threshold, we output the highest confidence value 1.0 for $conf_3(t,s)$. Otherwise, we construct two vectors v_t and v_s whose coefficients are hit ratios associated with concepts in C_t and C_s. We calculate a VSM [BYRN99] cosine measure $cos(v_t, v_s)$ between v_t and v_s, and let $conf_3(t,s)$ be $(cos(v_t, v_s) \times (r_t + r_s)/2)$.

3.4 Structure

We consider structure matching as one more technique that provides a clue about which elements to match. As an example of how structure helps resolve schema matching, and especially how it helps identify indirect element matches, consider *address* in Schema 1 (Figure 1(a)), which represents address objects for both house locations and agent contact addresses. Note that address objects functionally determine the value schema elements *street*, *county*, *city*, and *state*. In Schema 2 (Figure 1(b)), there are two kinds of addresses: *Location*, which is a value element that contains house location addresses, and *Address*, which is a value element that contains agent contact addresses. Assume that Schema 2 is the source and Schema 1 is the target. Observe that both *Location* and *Address* in Schema 2 match with *street*, *county*, *city* and *state* in Schema 1 indirectly through the *Decomposition* operation with a confidence factor, $conf_3$. Based on this observation and on structural observations, we can declare two sets of indirect element matches. One set includes {*street*, *Location*}, {*county*, *Location*}, {*city*, *Location*}, and {*street*, *Location*}. The other set includes {*street*, *Address*}, {*county*, *Address*}, {*city*, *Address*}, and {*street*, *Address*}. For each matching element pair, we add a *Union* operation in conjunction with the *Decomposition* operation to show that both *Location* and *Address* in Schema 2 match with the concatenation of *street*, *county*, *city*, and *state* in Schema 1. (We formalize these ideas in the matching algorithm, which we now present.)

4 Matching Algorithm

We have implemented an algorithm using our matching techniques that produces both direct and indirect matches between a target schema T and a source schema S. Figure 4 gives the algorithm, which we informally explain as follows.

[5]Clearly, the system would take different actions when transferring the data between schemas, but this is beyond the scope of this paper, which focuses only on discovering direct and indirect matches among schema elements.

```
Input: target schema $T$ and source schema $S$
Output: a set of element matches with manipulation operations
Step 1: Compute $conf$ measures between $T$ and $S$
          collect the object elements in $T$ into $T_1$, and collect the value elements in $T$ into $T_2$
          collect the object elements in $S$ into $S_1$, and collect the value elements in $S$ into $S_2$
          for each $(t,\ s)$ in $(T_1 \times S_1) \cup (T_2 \times S_2)$
            compute $conf_1(t,s)$ based on terminological relationships
          for each $(t,\ s)$ in $T_2 \times S_2$
            compute $conf_2(t,s)$ based on data-value characteristics
            compute $conf_3(t,s)$ based on expected data values
          for each $(t,\ s)$ in $T_1 \times S_1$
            $conf(t,s) = conf_1(t,s)$
          for each $(t,\ s)$ in $T_2 \times S_2$
            if $conf_3(t,s) = 1.0$ then $conf(t,s) = conf_3(t,s)$
            else
              $c_s(t,s) = conf_1(t,s)$
              $c_v(t,s) = (conf_2(t,s) + conf_3(t,s))/2$
              $conf(t,s) = c_s(t,s) \times w_s + c_v(t,s) \times w_v$
Step 2: Settle object element matches
          for each $t$ in $T_1$ and each $s$ in $S_1$
            collect $atoms_{direct}(s)$, $atoms(s)$, $atoms_{direct}(t)$ and $atoms(t)$
          for each $(t,\ s)$ in $T_1 \times S_1$
            compute $sim_{vicinity}(t,s)$ and $sim_{importance}(t,s)$
            if $sim_{vicinity}(t,s) > th_{vicinity}$ and $sim_{importance}(t,s) > th_{importance}$
              and $conf(t,s) > th_{conf}$ then
              mark $(t,s)$ as selected, mark $t$ in $T_1$, and mark $s$ in $S_1$
          Adjust $atoms_{direct}$ sets in $T$ and $S$ as follows
               for each unmarked $t$ in $T_1$
                 if $max_{s_i \in S_1}(sim_{vicinity}(t, s_i)) > th_{vicinity}$ then
                   adjust every $atoms_{direct}(t') = atoms_{direct}(t') \bigcup atoms_{direct}(t)$
                      where $t'$ is a parent object schema element of $t$ on which $t$ is functionally dependent
               for each unmarked $s$ in $S_1$
                 if $max_{t_i \in T_1}(sim_{vicinity}(t_i, s)) > th_{vicinity}$ then
                   adjust every $atoms_{direct}(s') = atoms_{direct}(s') \bigcup atoms_{direct}(s)$
                      where $s'$ is a parent object schema element of $s$ on which $s$ is functionally dependent
          assign appropriate operations with object element matches
Step 3: Settle value element matches
          for each selected $(t,\ s)$, which is a settled object element match
            for each $(t',\ s')$ in $atoms_{direct}(t) \times atoms_{direct}(s)$
              if $conf(t', s') = 1.0$ then
                mark settled element match($t'$, $s'$)
                mark $t'$ and $s'$ in $atoms_{direct}(t)$ and $atoms_{direct}(s)$ respectively
            combine $conf$ measures into a single $conf$ matrix $M$ for each pair $(t'',\ s'')$,
              where $t'' \in atoms_{direct}(t)$ and $t''$ is not marked, and $s'' \in atoms_{direct}(s)$ and $s''$ is not marked
            while there is an unsettled $conf$ measure in $M$ greater than $th_{conf}$
               find the largest unsettled $conf$ measure in $M$
               settle $conf$ by setting it to 1, and mark $conf$ as being settled
               for each unsettled $conf'$ in the rows and columns of $conf$
                 settle $conf'$ by setting it to 0, and mark $conf'$ as being settled
            mark settled element matches based on the settled $conf$ measures
            assign appropriate operations with value element matches
Step 4: Output element matches with manipulation operations
```

Figure 4: Matching Algorithm

Step 1: *Compute conf measures between T and S*. For each pair of schema elements $(t,\ s)$, which are either both value elements or both object elements, the algorithm computes a confidence value, $conf(t,s)$, to combine the output confidence values of the three nonstructural matching techniques. We compute $conf(t,s)$ using the following formula.

$$conf(t,s) = \begin{cases} conf_1(t,s)\ ,\ \text{if } t \text{ and } s \text{ are object schema elements} \\ 1.0\ ,\ \text{if } conf_3(t,s) = 1.0 \text{ and } t \text{ and } s \text{ are value schema elements} \\ w_s(conf_1(t,s)) + w_v(conf_2(t,s) + conf_3(t,s))/2\ ,\ \text{otherwise} \end{cases}$$

In this formula and w_s and w_v are experimentally determined weights. When the confidence value $conf_3(t,s) = 1.0$, which is a perfect match for $(t,\ s)$, we let $conf_3$ dominate and assign $conf(t,s)$ as 1.0 and keep the detected manipulation operations (*Selection*, *Union*, *Composition*, *Decomposition*) for indirect element matches. The motivation for letting $conf_3(t,s)$ dominate is that when expected values appear in both source and target schema elements and they both match well with the values we expect, this is a strong indication that the elements should match (either directly or indirectly). Since the domain ontology is not guaranteed to be complete (and may even have some inaccuracies) for a particular application domain, the confidence values obtained from the other techniques can complement and compensate for the inadequacies of the domain knowledge. This motivates the third part of the computation for $conf(t,s)$.
Step 2: *Settle object element matches*. When comparing two object element t and s, we take three factors into account: (1) the combined confidence measure $conf(t,s)$, (2) an importance similarity measure $sim_{importance}(t,s)$, and (3) a vicinity similarity measure $sim_{vicinity}(t,s)$. We can declare a matching pair (t,s) if $conf(t,s)$, $sim_{importance}(t,s)$, and $sim_{vicinity}(t,s)$ are high. We let $atoms_{direct}(e)$ denote the set of value elements directly connected to an object schema element e and let $atoms(e) = \bigcup_{e' \in E'} atoms_{direct}(e')$ denote the value elements of e, where E' is an object schema element set including e and other object schema elements that are functional dependent on e. We denote $atoms_{value}(T)$ and $atoms_{value}(S)$ as the sets of all value elements collected from T and S respectively. Given an experimentally determined threshold, th_{conf}, we calculate $sim_{importance}(t,s)$ and $sim_{vicinity}(t,s)$ based on the following formulas.

$$sim_{vicinity}(t,s) = max(\frac{|\{x | x \in atoms(t) \wedge \exists y \in atoms(s)(conf(x,y) > th_{conf})\}|}{|atoms(t)|}, \frac{|\{x | x \in atoms(s) \wedge \exists y \in atoms(t)(conf(y,x) > th_{conf})\}|}{|atoms(s)|})$$

$$sim_{importance}(t,s) = 1.0 - |\frac{atoms(t)}{atoms_{value}(T)} - \frac{atoms(s)}{atoms_{value}(S)}|$$

Intuitively, $sim_{vicinity}$ measures the similarity of the vicinity surrounding t and the vicinity surrounding s, and $sim_{importance}$ measures the similarity of the "importance" of t and the "importance" of s where we measure the "importance" of an object node N by counting the number of value nodes related to N and all other object nodes in the functional closure of N.
Step 3: *Settle value element matches*. For each matching pair $(t,\ s)$ of object elements settled in Step 2, we first settle value element matches of children of t and s (or children of functionally dependent object children of t and s) that match with high confidence ($conf = 1.0$). For all remaining unsettled value schema elements of t and s, we find a best possible match so long as the confidence of the match is above a given, experimentally determined threshold. For each of the matches, given the structure information and the expected-value matches, we determine the

appropriate operation (or sequence of operations) required to transform source schema elements into virtual elements that directly match with target schema elements.
Step 4: *Output both direct and indirect element matches with manipulation operations.*

5 Experimental Results

We evaluate the performance of our approach based on three measures: precision, recall and the F-measure, a standard measure for recall and precision together [BYRN99]. Given (1) the number of direct and indirect matches N determined by a human expert, (2) the number of correct direct and indirect matches C selected by our process described in this paper and (3) the number of incorrect matches I selected by our process, we compute the recall ratio as $R = C/N$, the precision ratio as $P = C/(C + I)$, and the F-measure, as $2/(1/R + 1/P)$. We report all these values as percentages.

We tested the approach proposed here using the running example in our paper and also on several real-world schemas in three different application domains. In our experiments, we evaluated the contribution of different techniques and different combinations of techniques. We always used both structure and terminological relationships, however, (1) because without at least some way to tentatively match schema elements (e.g. through terminological relationships) and some way to sort out the structural conflicts, we can produce neither direct nor indirect schema element matches and (2) because these techniques always apply for any two given schemas we wish to match even when no data is available. Thus, we tested our approach with four runs on each source-target pair. In the first run, we considered only terminological relationships and structure. In the second run, we added data-value characteristics. In the third run, we replaced data-value characteristics with expected data values, and in the fourth run we used all techniques together.

5.1 Running Example

We applied the matching algorithm explained in Section 4 to the schemas in Figure 1 populated (by hand) with actual data we found in some real-estate sites on the Web. First we let Schema 1 in Figure 1(a) be the target and Schema 2 in Figure 1(b) be the source. Then, we reversed the schemas and let Schema 2 be the target and Schema 1 be the source.

Run Nr.	Number of Matches	Number Correct	Number Incorrect	Recall %	Precision %	F-Measure %
1 (WS)	20	10	1	50%	91%	65%
2 (WCS)	20	10	0	50%	100%	67%
3 (WES)	20	20	0	100%	100%	100%
4 (WCES)	20	20	0	100%	100%	100%

W = Terminological Relationships using WordNet
C = Data-Value Characteristics
E = Expected Data Values
S = Structure

Table 1: Results for Running Example

Table 1 shows a summary of the results for each run in the first test where we let Schema 1 be the target and Schema 2 be the source. In this first run, the algorithm discovered all 8 direct

Application	Number of Matches	Number Correct	Number Incorrect	Recall %	Precision %	F-Measure %
Course Schedule	128	119	1	93%	99%	96%
Faculty	140	140	0	100%	100%	100%
Real Estate	245	229	22	93%	91%	92%
All Applications	513	488	23	95%	95%	95%

Table 2: Results for Real-World Examples

matches correctly, but it also misclassified the source schema element *Location* (meaning address) by matching it with the target schema element *location* (meaning "views" or "on the water front" or "by a golf course"). In the first run, the algorithm also successfully discovered 2 of the 12 indirect matches—(*phone*, *Day_Phone*) and (*phone*, *Evening_Phone*)—and correctly output the *Union* operation. In the second run, by adding the analysis of data-value characteristics, the false positive (*location*, *Location*) disappeared, but the algorithm generated no more indirect matches than in the first run. In both the third and fourth runs, the algorithm successfully discovered all direct and indirect matches. Especially noteworthy, we observed that our approach correctly discovered context-dependent indirect matches (e.g. (*city*, *Address*), (*state*, *Address*), ...) and appropriately produced operations composed of a combination of *Decomposition* and *Union*.

The result of the second test on our running example, in which we switched the schemas and let Schema 2 be the target schema and Schema 1 be the source schema, gave the same results as in Table 1. We observe, however, that although we correctly generated a *Selection* operator to decompose *location* (meaning "views," etc.), we were not able to automatically select the right set of values for *Water_Front* and *Golf_Course* and discard the remaining values, which were inapplicable for Schema 2.

5.2 Real-World Examples

We considered three real-world applications: *Course Schedule*, *Faculty*, and *Real Estate* to evaluate our approach. We used a data set downloaded from the LSD homepage [DDH01] for these three applications, and we faithfully translated the schemas from DTDs used by LSD to rooted conceptual-model graphs. For testing these real-world applications, we decided to let any one of the schema graphs for an application be the target and let any other schema graph for the same application be the source. Because our tests are nearly symmetrical, however, we decided not to test any target-source pair also as a source-target pair (as we did in our running example). We also decided not to test any single schema as both a target and a source. Since for each application there were five schemas, we tested each application 10 times. All together we tested 30 target-source pairs. For each target-source pair, we made four runs, the same four (WS, WCS, WES, and $WCES$) we made for our running example. All together we processed 120 runs.

Table 2 shows as summary of the results for the real-world data using all four techniques together. In two of the three applications, *Course Schedule* and *Faculty*, there were no indirect matches. For all four runs on *Faculty* every measure (recall, precision, F-measure) was 100%. For *Course Schedule*, the first and second run achieved above 90% and below 95% on all measures; and the third and fourth run gave the same results—those shown for *Course Schedule* in Table 2.

The *Real Estate* application exhibited several indirect matches. The problem of *Merged/Split Values* appeared twice, the problem of *Subsets/Supersets* appeared 24 times, and the problem of *Schema Element Name as Value* appeared 5 times. The experiments showed that the application of expected data values in the third and fourth run greatly affected the performance. In the first run, the measures were only about 75%. In the second run, the use of data-value characteristics improved the performance, but only a little because the measures were still below 80%. By applying expected data values in the last two runs, however, the performance improved dramatically. In the third run, the F-measures reached 91% and reached 92% by using all four techniques as Table 2 shows.

Our process successfully found all the indirect matches related to the problems of *Merged/Split Values* and *Schema Element Name as Value*, and correctly found 22 of the 24 indirect matches related to the problem of *Subsets/Supersets*. Our process, however, also declared two false positives for the problem of *Subsets/Supersets*, i.e. incorrectly declared two *Subsets/Supersets* that were not *Subsets/Supersets* matches. Over all the indirect element mappings, the three measures (recall, precision, and F-measure) were (coincidentally) all 94%.

5.3 Discussion

The experimental results show that the combination of terminological relationships and structure alone can produce fairly reasonable results, but by adding our technique of using expected data value, the results are dramatically better. Unexpectedly, the technique of using data-value characteristics did not help very much, if at all, for these application domains. Our analysis of data-value characteristics is similar to the analysis in SEMINT [LC00], which produced good results for their test data. The data instances in the real-world applications we used, however, do not appear to be as regular as might be expected. The statistics are highly variant, for example, in applications such as *Course Schedule* and *Real Estate*. For these applications, a large amount of training data would be needed to train a universal decision tree required in our approach.

Some element matches failed in our approach partly because they are potentially ambiguous, and our assertions about what should and should not match are partly subjective.[6] Even though we tested our approach using the same test data set as in LSD [DDH01], the answer keys were generated separately and may not be the same. Furthermore, neither the experimental methodologies nor the performance measures used are the same. Thus, although our raw performance numbers are an improvement over [DDH01], we do not try to draw any final conclusion.

One obvious limitation of our approach is the need to construct an application-specific domain ontology. Currently, we manually construct these domain ontologies. As we explained in Section 3, however, these domain ontologies are lightweight and are relatively easy to construct and need not be complete. It is possible, however, to make use of statistical learning techniques to collect a set of informative and representative keywords for application concepts. Thus, without human interaction, except for some labeling, we can make use of many keywords taken from the data of the application itself and thus specify regular-expression recognizers for the application concepts at least in a semi-automatic way. Furthermore, many values, such as dates, times, and currency amounts are common across many application domains and can easily be shared. Since domain ontologies appear to play an important role in indirect matching, finding ways to semiautomatically generate them is a goal worthy of some additional work.

6 Related Work

[RB01] provides a survey of several schema mapping systems. We do not repeat this work here, but instead describe work related to our approach from two perspectives: (1) work on discovering direct

[6]It is not always easy to do ground-truthing [HKL$^+$01].

matches for schema elements and (2) work on discovering indirect matches for schema elements.

Direct Matches. Most of the approaches [DDH01, EJX01, MBR01, LC00, MZ98, PTU00, BCV99] to automating schema matching focus only on generating direct matches for schema elements.

- In some of our previous work [EJX01], we experimented with using data instances to help identify direct element matches. In this paper, we refine this work and also extend it to the harder problem of discovering indirect matches.

- Like our approach, the LSD system [DDH01] applies a meta-learning strategy to compose several base matchers, which consider either data instances, or schema information. LSD largely exploits machine learning techniques. There are two phases in the LSD system: one is training and the other is testing. In the training phase, LSD requires training data for each matching element between two schemas for base matchers and the meta matcher. In our approach, however, we applied machine learning algorithms only to terminological relationships and data-value characteristics. For each of these two techniques, our system learned a universal decision tree for all application domains based on a domain-independent training set. To combine techniques, we let structure features guide the matching based on the results from multiple kinds of independent matches. Thus, our approach avoids the work of collecting and labeling training data as in LSD.

- SEMINT [LC00] applies neural-network learning to automating schema matching based on instance contents. It is an element-level schema matcher because it only considers attribute matching without taking the structure of schemas into account.

- The structure matching algorithm in Cupid [MBR01] motivated our structure matching algorithm. Cupid, however, does not properly handle two schemas that are largely different. Moreover, the structure matching algorithm Cupid uses has a mutually recursive flavor and matches two schemas using a bottom-up strategy. Our matching algorithm discovers direct and indirect matches using a top-down strategy.

- DIKE [PTU00], ARTEMIS [BCV99], and Cupid [MBR01] exploit auxiliary information such as synonym dictionaries, thesauri, and glossaries. All their auxiliary information is schema-level—does not consider data instances. In our approach, the auxiliary information including data instances and domain ontologies provide a more precise characterization of the actual contents of schema elements. The imported dictionary we use, WordNet, is readily available and no work is required to produce thesauri as in other approaches.

Indirect Matches. Some work on indirect matches is beginning to appear, but researchers are only beginning to scratch the surface of the multitude of problems.

- Both Cupid [MBR01] and SKAT [MWJ99] can generate global $1:n$ indirect matches [RB01]. To illustrate what this means, if in our running example in Figure 1 we let Schema 1 be the target and Schema 2 be the source, and if we make *address* a value element rather than an object element and discard *street*, *county*, *city*, and *state* in Schema 1, Cupid can match both *Address* and *Location* in the source directly with the modified *address* in the target. Thus Cupid can generate a global $1:n$ indirect match through a *Union* operation. Our approach, however, can find indirect matches for *Location* and *Address* in the source with *street*, *county*, *city*, and *state* in the target based on finding expected data values and using the *Decomposition* operator as well as the *Union* operator, something which is not considered in Cupid.

- The Clio system [MHH00] introduces an interactive mapping creation paradigm based on value correspondence that shows how a value of a target schema element can be created from a set of values of source elements. A user or DBA, however, is responsible to manually input the value correspondences.

- [BE02] proposes a mapping generator to derive an injective target-to-source mapping including indirect matches in the context of information integration. The mapping generator raises specific issues for a user's consideration. The mapping generator, however, has not been implemented. Our work therefore builds on and is complimentary to the work in [BE02].

7 Conclusion

We presented a framework for automatically discovering both direct matches and many indirect matches between sets of source and target schema elements. In our framework, multiple techniques each contribute in a combined way to produce a final set of matches. Techniques considered include terminological relationships, data-value characteristics, expected values, and structural characteristics. We detected indirect element matches for *Selection*, *Union*, *Composition*, and *Decomposition* operations as well as conversions for *Schema-Element Names as Values.* We base these operations and conversions mainly on expected values and structural characteristics. Additional indirect matches, such as arithmetic computations and value transformations, are for future work. We also plan to semi-automatically construct domain ontologies used for expected values, automate application-dependent parameter tuning, and test our approach in a broader set of real-world applications. As always, there is more work to do, but the results of our approach for both direct and indirect matching are encouraging, yielding over 90% in both recall and precision.

References

[BCV99] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, March 1999.

[BE02] J. Biskup and D.W. Embley. Extracting information from heterogeneous information sources using ontologically specified target views. *Information Systems*, 2002. (to appear).

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, Menlo Park, California, 1999.

[CA99] S. Castano and V. De Antonellis. ARTEMIS: Analysis and reconciliation tool environment for multiple information sources. In *Proceedings of the Convegno Nazionale Sistemi di Basi di Dati Evolute (SEBD'99)*, pages 341–356, Como, Italy, June 1999.

[DDH01] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD 2001)*, pages 509–520, Santa Barbara, California, May 2001.

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

[EJX01] D.W. Embley, D. Jackman, and Li Xu. Multifaceted exploitation of metadata for attribute match discovery in information integration. In *Proceedings of the International Workshop on Information Integration on the Web (WIIW'01)*, pages 110–117, Rio de Janeiro, Brazil, April 2001.

[Fel98] C. Fellbaum. *WordNet: An Electronic Lexical Database.* MIT Press, Cambridge, Massachussets, 1998.

[HKL$^+$01] J. Hu, R. Kashi, D. Lopresti, G. Nagy, and G. Wilfong. Why table ground-truthing is hard. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, pages 129–133, Seattle, Washington, September 2001.

[LC00] W. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data & Knowledge Engineering*, 33(1):49–84, 2000.

[MBR01] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 49–58, Rome, Italy, September 2001.

[MHH00] R. Miller, L. Haas, and M.A. Hernandez. Schema mapping as query discovery. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB'00)*, pages 77–88, Cairo, Egypt, September 2000.

[Mil95] G.A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41, November 1995.

[MWJ99] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. In *FUSSION 99*, 1999.

[MZ98] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB-98)*, pages 122–133, August 1998.

[PTU00] L. Palopoli, G. Teracina, and D. Ursino. The system DIKE: Towards the semi-automatic synthesis of cooperative information systems and data warehouses. In *Proceedings of ADBIS-DASFAA 2000*, pages 108–117, 2000.

[Qui93] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, California, 1993.

[RB01] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.

[Ull97] Jeffrey D. Ullman. Information integration using logical views. In Foto N. Afrati and Phokion Kolaitis, editors, *Proceedings of the 6th International Conference on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40, Delphi, Greece, January 1997. Springer-Verlag.