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. [9, 5]) 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 source schema and a target 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, yielding over 90% precision and recall for both direct and indirect element matches.

Keyword: Schema matching, data integration, schema integration, data exchange.

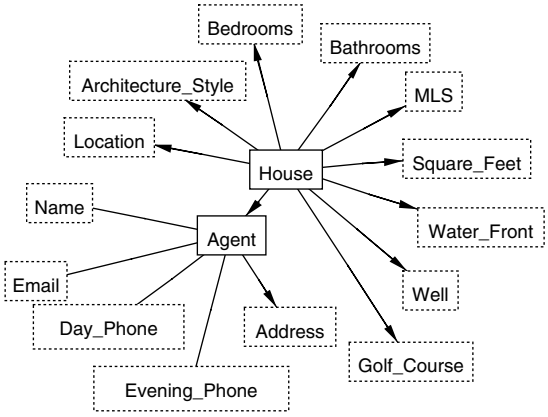
1. Introduction

In this paper, we focus on the long-standing and challenging problem of automating schema matching [9]. Schema matching is a key operation for many applications including data integration, schema integration, message mapping in E-commerce, and semantic query processing [15]. Schema matching takes two schemas as input and produces as output a semantic correspondence between the schema elements in the two input schemas [15]. 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 [2, 5, 7, 8, 9, 12, 13] has focused on computing direct element matches. Such simplicity, however, is rarely suf£cient, and researchers have thus proposed the use of queries over source schemas to form virtual schema elements to bind with target schema elements [3, 11]. 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.

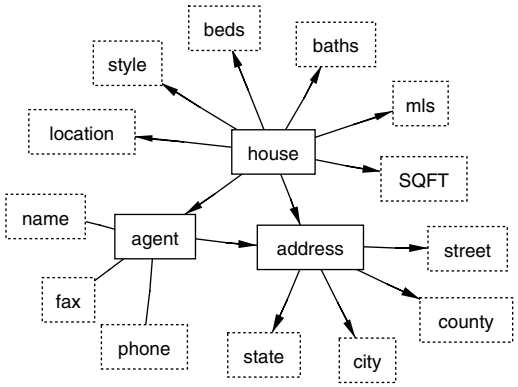
We assume that all source and target schemas are described using rooted conceptual-model graphs (a conceptual generalization of XML). Element nodes either have associated data values or associated object identi£ers, which we respectively call *value schema elements* and *object schema elements*. We augment schemas with a variety of ontological information. For this paper the augmentations we discuss are WordNet [10], sample data, and regular-expression recognizers. For each application, we construct a domain ontology [6], 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 augmentations, we exploit a broad set of techniques together to settle direct and indirect element matches between a source schema and a target 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 source schema S and a target schema T as well as the direct correspondences and (2) experimental results of our implementation to show that it performs as well (indeed better) than other approaches for direct element matches and also performs exceptional well for the indirect matches with which we work. We present the details of our contribution as follows. Section 2 explains what we mean by direct and indirect matches between S and T. Section 3 describes a set of basic matching techniques to £nd potential element matches between elements in S and elements

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



(a) Schema 1



(b) Schema 2

in T, and to provide con£dence measures between 0 (lowest con£dence) and 1 (highest con£dence) for each potential match. Section 4 presents an algorithm to settle direct and indirect matches between S and T. Section 5 gives experimental results for a data set used in [5] to demonstrate the success of our approach. In Section 6 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 2, for example, shows two schema graphs, each partially describing a real-estate application. In a schema graph we denote value schema elements as dotted boxes, object schema elements as solid boxes, functional relationships as lines with an arrow from domain to range, and nonfunctional relationships as lines without arrowheads.

The output of schema matching is a set of element mappings that match actual or virtual source schema elements with £xed 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 $(s,\ t)$ is *direct* when a source schema element s and a target schema element t 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 identi£cation of synonyms is not enough [7]. 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 2.

1. *Generalization* and *Specialization*. Two elements, Day_Phone and $Evening\_Phone$ in Figure 1(a) are both specializations of $phone$ values in Figure 1(b). Thus, if Figure 1(b) is the target, we need the union of Day_Phone and $Evening\_Phone$, and if Figure 1(a) is the target, we should £nd 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(b) and merged as *Location* of a house or *Address* of an agent in Figure 1(a). Thus, we need to split the values if Figure 1(b) is the target and merge the values if Figure 1(a) is the target.

3. *Schema Element Name as Value*. In Figure 1(a), the features $Water\_Front$ and $Golf\_Course$ are 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(b).

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

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

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

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

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

5. *Boolean*. Attribute names with Boolean values (e.g. "Yes/No") of a source (target) schema are values in a target (source) schema.

The recognition and speci£cation 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, or element values as potential target element names. Then, in harmony with the source values (e.g. "Yes"/"No") and target element names or source element names and target values (e.g. "Yes"/"No"), we can determine the 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-speci£c, regular-expression matches (i.e. the appearance of expected strings), and (4) structure (e.g. structural similarities).

3.1. Terminological Relationships

The meaning of element names provides a clue about which elements match. To match element names, we use WordNet [10] which organizes English words into synonym and hypernym sets. Other researchers have also suggested using WordNet to match attributes (e.g. [2]), but have given few, if any, details. We use a C4.5 [14] learning algorithm to train a set of decision rules to compute a con£dence value, denoted $conf_1(s,t)$, where s is a source schema element and t is a target schema element. See [7] for details.

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 con£dence value $conf_1(s,t)$ 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 con£dence of ($location$, $Location$) is high, even though the meaning is entirely different; but, as we shall see, other techniques can sort out 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 [8] 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 [8], but generate a C4.5 decision tree rather than a neural-net decision rule. Based on the decision tree, we generate a con£dence value, denoted $conf_2(s,t)$, for each element pair (s, t) of value schema elements. See [7] for details.

Testing the decision rule using data values associated with Schema 2 in Figure 1(b) as a source schema and Schema 1 in Figure 1(a) as a target schema, the con£dence value $conf_2(s,t)$ is high for the matches such as ($beds$, $Bedrooms$), ($baths$, $Bathrooms$), ($phone$, Day_Phone), and (fax, Day_Phone) as expected. However, mls in the source and $Location$ in the target tend to look alike according to the value characteristics measured, a surprise which needs other techniques to £nd the difference. Interestingly, the lot features in $location$ of the source schema and the house locations in $Location$ of the target 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 speci£c application, we can specify a domain ontology [6], which includes a set of concepts and relationships among the concepts, and associates with each concept a set of regular expressions that matches values and keywords expected to appear for the concept. Then, using techniques described in [6], we can extract values from sets of data associated with source and target value elements and categorize their data-value patterns 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 domain ontologies independently of any target and source schemas. Figure 1 shows three components in our real-

estate domain ontology, which we used to automate matching of the two schemas in Figure 2 and also for matching real-world schemas in the real-estate domain in general. The three components include an address component specifying $Address$ as potentially consisting of *State*, $City$, $County$, and $Street$;[1] a phone component specifying $Phone$ as a possible superset of $Day\ Phone$, $Evening\ Phone$, $Home\ Phone$, $Office\ Phone$, and $Cell\ Phone$;[2] and a lot-feature component specifying $Lot\ Feature$ as a possible superset of $View$ values and individual values $Water\ Front$ and $Golf\ Course$.[3] Behind a dotted box (or individual value), a regular-expression recognizer [6] describes the expected data values for a potential application concept. The ontology explicitly declares that (1) the expected values in $Address$ 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 in $Day\ Phone$, $Evening\ Phone$, $Home\ Phone$, $Office\ Phone$, and $Cell\ Phone$; and (3) the set of values associated with $Lot\ Feature$ is a superset of the values associated with the set of $View$ values and the singleton-sets $Water\ Front$ and $Golf\ Course$.

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

1. *Composition* and *Decomposition*. Based on the $Address$ declared in the ontology in Figure 1, the recognition-of-expected-values technique [6] can help detect that (1) the values of both $Address$ and $Location$ in Schema 1 match with the ontology concept $Address$, and (2) the values of $street$, $county$, $city$, and $state$ in Schema 2 match with the ontology concepts $Street$, $County$, $City$, and *State* respectively. Thus, if Schema 1 is the source and Schema 2 is the target, 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 2 be the source and Schema 1 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 speci£cation of the regular expression matched for $Phone$, the schema elements Day_Phone and $Evening\_Phone$ in Schema 1 match with the concepts $Day\ Phone$ and $Evening\ Phone$ respectively, and $phone$ in Schema 2 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 $Day\ Phone$ and $Evening\ Phone$. Thus, we are able to identify the indirect matching schema elements between $phone$ in Schema 2 and Day_Phone and $Evening\_Phone$ in Schema 1. If Schema 1 is the source and Schema 2 is the target, we can apply a *Union* operation over Schema 1 to derive a virtual schema element $Phone'$, which can directly match with $phone$ in Schema 2. If Schema 2 is the source and Schema 1 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

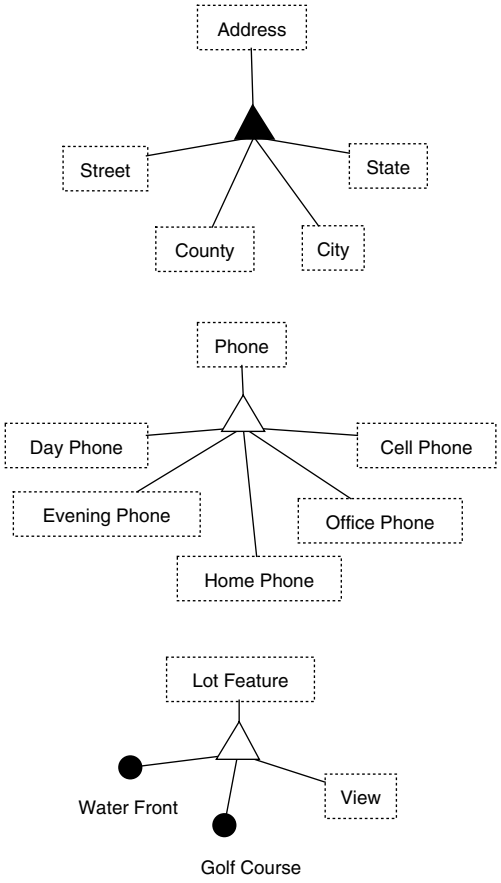


Figure 1. Application Domain Ontology (Partial)

[1] Filled-in (black) triangles denote aggregation ("part-of" relationships).

[2] Open (white) triangles denote generalization/specialization ("ISA" supersets and subsets).

[3] Large black dots denote individual objects or values.

out either).

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

We now more formally describe these three types of indirect matches. 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 £rst 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_s match a source element s and a set of concatenated concepts C_t match a target element t. For each concept in C_s or in C_t, we have an associated hit ratio. Hit ratios give the percentage of s or t values that match (or are included in at least some match) with the values of the concepts in C_s or C_t respectively. We also have a hit ratio r_s associated with C_s that gives the percentage of s values that match the concatenation of concepts in C_s, and a hit ratio r_t associated with C_t that gives the percentage of t values that match the concatenation of concepts in C_t. To obtain hit ratios for Boolean £elds recognized as schema-element names, we distribute the schema-element names over all the Boolean £elds.

We decide if s matches with t directly or indirectly by comparing C_s and C_t. If C_s equals C_t, we declare a *direct* match (s, t). Otherwise, if $C_s \supset C_t$ ($C_s \subset C_t$), we derive an *indirect* match (s, t) through a *Decomposition* (*Composition*) operation. If both C_s and C_t contain one individual concept c_s and c_t respectively, and if the values of concept c_s (c_t) are declared as a subset of the values of concept c_t (c_s), we derive an *indirect* match (s, t) through a *Union* (*Selection*) operation. When we have schema-element names as values, distribution of the name over the Boolean value £elds converts these schema elements into standard schema elements with conventional value-populated £elds. Thus, no additional comparisons are needed to detect direct and indirect matches when schema-element names are values. [4] We must, however, remember the Boolean conversion for both source and target schemas to correctly derive indirect matches.

We compute the con£dence value for a mapping (s, t), which we denoted as $conf_3(s,t)$, as follows. If we can declare a direct match or derive an indirect match through manipulating *Union*, *Selection*, *Composition*, *Decomposition*, and *Boolean* for (s, t), and the hit ratios r_s and r_t are above an accepted threshold, we output the highest con£dence value 1.0 for $conf_3(s,t)$. Otherwise, we construct two vectors v_s and v_t whose coef£cients are hit ratios associated with concepts in C_s and C_t. To take the partial similarity between v_s and v_t into account, we calculate a VSM [1] cosine measure $cos(v_s, v_t)$ between v_s and v_t, and let $conf_3(s,t)$ be $(cos(v_s, v_t) \times (r_s + r_t)/2)$.

3.4. Structure

We consider structure matching as one more technique that provides a clue about which elements match. Given the con£dence measures output from the other matching techniques as a guide, structure matching determines element matches by considering contexts around schema elements.

As an example of how structure uses contexts of schema elements to help resolve schema matching, and especially how it helps identify indirect element matches, consider $address$ in Schema 2 (Figure 1(b)), which contains address objects that are functionally dependent on the object schema elements $house$ and $agent$. In Schema 1 (Figure 1(a)), there are two kinds of addresses: $Location$, which contains house location addresses, and $Address$, which contains agent contact addresses. Assume that Schema 2 is the source and Schema 1 is the target. By considering the value elements, we observe that $street$, $county$, $city$ and $state$ in Schema 2 match with both $Location$ and $Address$ in Schema 1 indirectly through the *Composition* operation with a con£dence factor, $conf_3$. Based on this observation and on structural observations, we can declare two sets of indirect element matches. One set includes ($state$, $Location$), ($county$, $Location$), ($city$, $Location$), and ($street$, $Location$). The other set includes ($state$, $Address$), ($county$, $Address$), ($city$, $Address$), and ($street$, $Address$). For each matching element pair, we add a *Selection* operation, based on the structure, in conjunction with the *Composition* operation to separate the concatenation of $street$, $county$, $city$, and $state$ in Schema 2 to match correctly with $Location$ and $Address$ in Schema 1.

[4] 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.

4. Matching Algorithm

We have implemented an algorithm using our matching techniques that produces both direct and indirect matches between a source schema S and a target schema T. We informally explain this algorithm as follows.

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

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

In this formula, w_s and w_v are experimentally determined weights.[5] When the con£dence value $conf_3(s,t)$ = 1.0, we let $conf_3$ dominate and assign $conf(s,t)$ as 1.0 and keep the detected manipulation operations ($Selection$, $Union$, $Composition$, $Decomposition$, $Boolean$) for indirect element matches. The motivation for letting $conf_3(s,t)$ 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 con£dence 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 $conf(s,t)$.

Step 2: *Settle object element matches*. When comparing two object element s and t, we take three factors into account: (1) the combined con£dence measure $conf(s,t)$, (2) an importance similarity measure $sim_{importance}(s,t)$, and (3) a vicinity similarity measure $sim_{vicinity}(s,t)$. We can declare a matching pair (s,t) if $conf(s,t)$, $sim_{importance}(s,t)$, and $sim_{vicinity}(s,t)$ are high. The latter two measures together represent the similarity between the contexts of s and t. We let $atoms_d(e)$ denote the set of value elements directly connected to an object schema element e and let $atoms(e) = \bigcup_{e' \in E'} atoms_d(e')$ 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 $atoms_{value}(S)$ and $atoms_{value}(T)$ as the sets of all value elements collected from S and T respectively. Given an experimentally determined threshold, th_{conf},[6] we calculate $sim_{importance}(s,t)$ and $sim_{vicinity}(s,t)$ based on the following formulas.

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

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

Intuitively, $sim_{vicinity}$ measures the similarity of the vicinity surrounding s and the vicinity surrounding t, and $sim_{importance}$ measures the similarity of the "importance" of s and the "importance" of t 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. When the number of schema elements is largely different, it is dif£cult to decide the vicinity similarity based on one measure, $sim_{vicinity}$ [9]. The conceptual analysis techniques discussed in [4] motivated $sim_{importance}$, which helps measure the context similarity from an additional perspective.

Step 3: *Settle value element matches*. For each matching pair (s, t) of object elements settled in Step 2, we £rst settle value element matches of children of s and t (or children of functionally dependent object elements of children of s and t) that match with high con£dence (*conf* = 1.0). For all remaining unsettled value schema elements of s and t, we £nd a best possible match so long as the con£dence of the match is above the threshold, th_{conf}. 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 [1]. 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

[5] The two parameters w_s, which weights schema element names, and w_v, which weights schema element values, are application dependent. Using a heuristic guide, however, we can determine the two parameters based on schemas and available data even without experimental evidence. If the schema element names are informative and the data is not self descriptive, we assign w_s as 0.8 and w_v as 0.2. On the other hand, if the schema element names are not informative and the data is semantically rich, we assign w_s as 0.2 and w_v as 0.8. For all other cases, we assign both w_s and w_v as 0.5.

[6] For any application, the computed con£dence values tend to converge to a speci£c high measure for element matches between two schemas. Thus, we use a universal threshold value. Experimentally, we have determined that 0.7 works well across all applications.

precision ratio as $P = C/(C + I)$, and the F-measure as $F = 2/(1/R+1/P)$. 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 because given any two schemas, these techniques always apply even when no data is available. Thus, we tested our approach with four runs on each source-target pair. In the £rst 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 2 populated (by hand) with actual data we found in some real-estate sites on the Web. In the £rst test, we let Schema 2 in Figure 1(b) be the source and Schema 1 in Figure 1(a) be the target. In the £rst run, the algorithm discovered all 8 direct matches correctly, but it also misclassi£ed the source schema element $location$ (meaning "view" or "on the water front" or "by a golf course") by matching it with the target schema element $Location$ (meaning address). In the £rst run, the algorithm also successfully discovered 2 of the 12 indirect matches—($phone$, Day_Phone) and ($phone$, $Evening\_Phone$)—and correctly output the *Selection* 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 £rst 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 *Composition* and *Selection*. The result of the second test on our running example, in which we switched the schemas and let Schema 1 be the source schema and Schema 2 be the target schema, gave the same results as in the £rst test.

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 [5] for these 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, 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 £ve 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 we made for our running example. All together we processed 120 runs.

Application	Number of Matches (N)	Number Correct (C)	Number Incorrect (I)
Course Schedule	128	119	1
Faculty	140	140	0
Real Estate	245	229	22
All Applications	513	488	23

Table 1. Results for Real-World Examples

Table 1 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 £rst and second run achieved above 90% and below 95% on all measures; and the third and fourth run gave the results for *Course Schedule* as Table 1 shows.

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 £rst 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 1 shows.

Our process successfully found all the indirect matches related to the problems of *Merged/Split Values* and *Schema Element Name as Value*. For the problem of *Subsets/Supersets*, our process correctly found 22 of the 24 indirect matches and declared two extra indirect 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 values, the results are dramatically better. Unexpectedly, the technique of using data-value characteristics did not help very much for these application domains. Our analysis of data-value characteristics is similar to the analysis in SEMINT [8], 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. For these applications, a large amount of training data would be needed to train a universal decision tree required for this 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.we tested our approach using the same test data set as in LSD [5], 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. With this understanding, we remark that [5] reported approximate accuracies of 70% for *Course Schedule*, 90% for *Faculty*, 70% and 80% for the two experiments they ran on the *Real Estate* application. Thus, although our raw performance numbers are an improvement over [5], we do not try to draw any £nal conclusion.

6. 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 £nal 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 $Boolean$ 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

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

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

[3] J. Biskup and D. Embley. Extracting information from heterogeneous information sources using ontologically speci£ed target views. *Information Systems*, 28(1), 2003. To appear, currently at http://www.deg.byu.edu/papers/int.pdf.

[4] S. Castano, V. D. Antonellis, M. Fugini, and B. Pernici. Conceptual schema analysis: Techniques and applications. *ACM Transactions on Database Systems*, 23(3):286–333, September 1998.

[5] 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*, pages 509–520, Santa Barbara, California, May 2001.

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

[7] D. Embley, D. Jackman, and L. 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.

[8] 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.

[9] J. Madhavan, P. 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.

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

[11] R. Miller, L. Haas, and M. 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.

[12] 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, New York City, New York, August 1998.

[13] 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.

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

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