Direct and Indirect Matching of Schema Elements for Data Integration on the Web

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

Current approaches (e.g. [MBR01, DDH01]) to automating schema matching for data integration 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 way to generate 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.

1 Introduction

In this paper, we focus on the long-standing and challenging problem of schema matching [MBR01] for the purpose of data integration on the Web. We assume that we wish to integrate data from multiple *source* schemas into a *target* schema. The schema matching problem is to find a *semantic correspondence* between one or more source schemas and a target schema [DDH01]. 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] 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 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. Element nodes either have associated data values or associated object identifiers, which we respectively call *value schema elements* and *object schema elements*. We augment target schemas with a variety of ontological information. For this paper the augmentations we discuss are WordNet [Mil95], sample data, and regular-expression recognizers. 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 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 source schema S and a target schema T 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. Section 3 describes a set of basic matching techniques to find potential element matches between elements in S and elements in T, and to provide confidence measures between 0 (lowest confidence) and 1 (highest confidence) for each potential match. Section 4 presents an algorithm to settle element matches between S

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

and T. Section 5 gives experimental results for a data set used in [DDH01]. In Section 6 we summarize and draw conclusions.

2 Source-to-Target Mappings

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 derived data, including missing generations and specializations, and aggregated and decomposed 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, $Price$ may be the selling price of a car while $Payment$ (or even $Price$ itself) may be the monthly payment of a lease. Our approach considers both schema information and data instances to help settle direct element matches, and thus largely avoids this problem.

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.

1. *Merged Values.* Two elements, *Make:Ford* and *Model:Taurus*, are separate in one schema and merged as *Make&Model:Ford Taurus* in another schema.

2. *Subsets.* Two elements, $BodyType$ and $Color$, in one schema are both special $Feature$ values in another schema. Thus the set of body types and colors are subsets of the feature values in $Feature$.

3. *Schema Element as Value.* In one schema, the features $Auto$, $AirCond$, AM/FM, and CD 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 $Auto$, $AirCond$, AM/FM, and CD should be included as $Feature$ values in the other schema.

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. Union* is the inverse of *Selection.*

3. *Composition.* The facts associated with a target schema element match the values merged from source schema elements.

4. *Decomposition. 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 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.

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. appearance of expected strings), and (4) structure (e.g. similarities in vicinity and importance). For the first two techniques we obtain a vector of measures for the features of interest and then apply machine learning over this feature vector 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] which organizes English words into synonym and hypernym sets. We select a set of 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 confidence value, denoted $conf_1(t, s)$, is computed based on trained decision rule (see [EJX01]).

3.2 Data-Value Characteristics

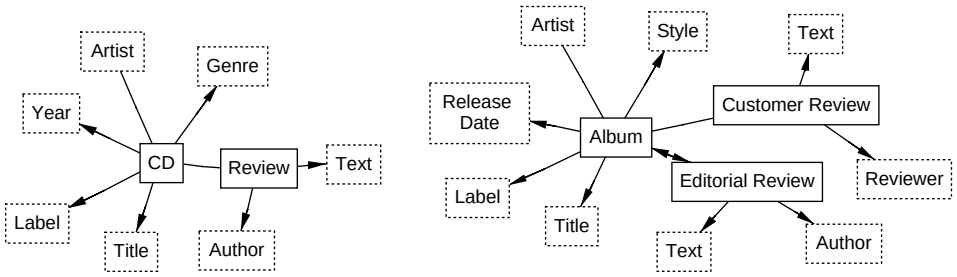
Whether two sets of data have similar value characteristics provides another a clue about which elements match. Previous work in [LC00] shows that this technique can successfully help match elements by considering such characteristics as string-lengths and alphabetic/non-alphabetic ratios of alphanumeric data and means and variances of numerical data. We use similar features to those in [LC00], but generate a C4.5 decision rule rather than a neural-net decision rule. Based on the decision rule (see [EJX01]), we generate the confidence value, denoted $conf_2(t, s)$, for an element pair (t, s).

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 domain, we can specify a set of concepts and associate with each concept a set of regular expressions that matches values and keywords expected to appear for this concept. Then, using techniques described in [ECJ+99], we can extract values from sets of data associated with source/target elements and categorize their data-value patterns based on the regular expressions declared for application concepts. For example, we can detect that the values of a source schema element $Make\&Model$ match with the concatenation of values specified for two concepts $Make$ and $Model$ in a car-ad application. Thus, we can use *Decomposition* over $Make\&Model$ in the source to indirectly match with $Make$ and $Model$ in the target. We can also derive indirect matches for *Selection* and *Union* operations. For example, if $Phone_{day}$ and $Phone_{evening}$ are specializations of $Phone$ in a target schema, and a source schema has only $PhoneNumber$, we may be able to recognize keywords such as Day, $Work$, $Evening$, $Home$ associated with each phone number in the source. If so, we can use a *Selection* operation to sort out which phone numbers belong in which specialization (if not, a human expert may not be able to sort these out either).

Let c_i be an application concept, such as $Make$, and consider a concatenation of concepts such as $Make\&Model$. 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.

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. 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 $(cos(v_t, v_s) \times (r_t + r_s)/2)$.



(a) Target Schema (b) Source Schema

Figure 1: Schema Graphs for Target and Source

3.4 Structure

We consider structure matching as one more technique that provides a clue about which elements to match. We represent the structure of both source and target schemas as rooted graphs. Nodes of the graph denote object and value schema elements, and edges of the graph denote relationship sets among object and value schema elements. Figure 1, for example, shows a target schema graph (Figure 1(a)) and a source schema graph (Figure 1(b)) for a music CD application. In a schema graph we denote value schema elements as dotted boxes, object schema elements as solid boxes, functional relationship sets as lines with an arrow from domain to range, and nonfunctional relationship sets as lines without arrowheads.

As an example of how structure helps resolve schema matching, and especially how it helps identify indirect element matches, consider the indirect element matches in Figure 1. In the target (Figure 1(a)), *Review* represents the reviews for a CD, *Author* represents the authors of reviews, and $Text$ represents the review texts. In the source (Figure 1(b)), there are two types of reviews: $CustomerReview$ and $EditorialReview$; *Author* represents the authors of editorial reviews, *Reviewer* represents the authors of customer reviews, and $Text$ appears twice, once for $CustomerReview$ and once for $EditorialReview$. Observe that both $CustomerReview$ and $EditorialReview$ in the source match with *Review* in the target. Based on this observation and on structural observations, we can declare two indirect element matches (*Review*, $CustomerReview$) and (*Review*, $EditorialReview$), and add a *Union* operation. Further, we can propagate the operation to the children and match $Text$ in the target with the *Union* of the two $Text$ elements in the source and match *Author* in the target with the *Union* of *Author* and *Reviewer* in the source.

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. A sketch of the algorithm is in the Appendix. We informally explain the algorithm as follows.

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. If both t and s are value elements, we compute $conf(t, s)$ using the formula $conf(t, s) = w_s \times c_s + w_v \times c_v$, where c_s is the confidence value $conf_1(t, s)$, c_v is the average of confidence values $conf_2(t, s)$ and $conf_3(t, s)$, and w_s and w_v are experimentally determined weights for c_s and c_v respectively. 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. If both t and s are object schema elements, we let $conf(t, s)$ be c_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 by 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 a threshold, th_{conf}, we calculate $sim_{importance}(t, s)$ and $sim_{vicinity}(t, s)$ based on the following formulas.

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

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

Our vicinity and importance measures are motivated by the structural-similarity measure in [MBR01], but, unlike [MBR01], we are able to control for large differences in numbers of value schema elements. To assign an appropriate operation for an indirect element match(t, s), we assign *Union* if multiple source schema elements match with t and assign *Selection* if s matches with multiple target schema elements.

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 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 assigned manipulation operations.

5 Experimental Results

To test the approach proposed here, we considered three applications: *Course Schedule*, *Faculty*, and *Real Estate*. We used a data set downloaded from the LSD homepage [DDH01], and we faithfully translated the schemas from DTDs to rooted conceptual-model graphs. For testing these applications, we decided to let any one of the schema graphs for an application be the target and let any other schema graph 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. We also decided not to test any single schema graph as both the target and the source. Since for each application there were five schemas, we tested each application ten times. All together we ran 30 tests. Table 1 shows as summary of the results for each application and over all applications together. Table 1 gives (1) the number of 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, (3) the number of incorrect matches I selected by our process, (4) the recall ratio, $R = C/N$, which we express as a percentage, (5) the precision ratio, $P = C/(C + I)$, which we express as a percentage, and (6) the F-measure, $2/(1/R + 1/P)$, a standard measure for recall and precision together, which we also express as a percentage.

In the test applications, there are no indirect element matches in *Course Schedule* and *Faculty*. In *Real Estate*, however, the problem of *Merged/Split Values* appeared twice, the problem of *Subsets/Supersets* appeared 24 times, and the problem of *Schema Element as Value* appeared 5 times. Our process successfully found all the indirect matches related to the problems of *Merged/Split Values* and *Schema Element as Value*, and correctly found the indirect matches related to the problem of *Subsets/Supersets* 22 times. Besides the two false negatives for the problem of *Subsets/Supersets*, our process also declared two false positives, that is, twice incorrectly detected *Subsets/Supersets* when there were none. Over all the indirect element mappings, the three measures including precision, recall and F-measure are (coincidentally) all 94%.

We tested our approach using the same test data set as in LSD [DDH01]. Although our raw numbers are an improvement over [DDH01], we do not try to draw any final conclusion because ground truthing the schema matching problem is inherently subjective, and neither the experimental methodologies nor the performance measures used are the same.

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

Application	Number of Matches	Number Correct	Number Incorrect	Recall %	Precision %	F-Measure %
Course Schedule	128	119	2	93%	98%	96%
Faculty	140	140	0	100%	100%	100%
Real Estate	245	235	20	96%	92%	94%
All Applications	513	494	22	96%	96%	96%

Table 1: Results

in a combined way to produce a final set of matches. 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, value transformations, and additional compound combinations of our operations, are for future work. The results of our approach to both direct and indirect matching are encouraging, yielding over 90% in both recall and precision, and show that the approach indeed has promise.

References

[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.

[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.

[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.

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

Appendix: Source-to-Target Matching Algorithm

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