# Automatic Direct and Indirect Schema Mapping: Experiences and Lessons Learned

David W. Embley
Brigham Young University
embley@cs.byu.edu

Li Xu
University of Arizona South
lxu@email.arizona.edu

Yihong Ding
Brigham Young University
ding@cs.byu.edu

## ABSTRACT

Schema mapping produces a semantic correspondence between two schemas. Automating schema mapping is challenging. The existence of 1:$n$ (or $n$:1) and $n$:$m$ mapping cardinalities makes the problem even harder. Recently, we have studied automated schema mapping techniques (using data frames and domain ontology snippets) that not only address the traditional 1:1 mapping problem, but also the harder 1:$n$ and $n$:$m$ mapping problems. Experimental results show that the approach can achieve excellent precision and recall. In this paper, we share our experiences and lessons we have learned during our schema mapping studies.

## 1. INTRODUCTION

Schema mapping is a key operation for many applications such as data integration, ontology matching, data warehousing, message translation in E-commerce, and semantic query processing. Schema mapping takes two schemas as input and produces as output a semantic correspondence between the schema elements in the two input schemas [14].

In this paper, we call each schema element an *object set*. There are three types of mapping cardinality problems. The most widely studied is the mapping between two individual object sets in different schemas, the *1:1 mapping cardinality* problem. Mappings, however, do not occur between just two object sets. For example, one object set, *Address*, may be a combination of *Street*, *City*, and *State* in a single string, while in another schema there may be three object sets—*Street*, *City*, and *State*. The one object set, *Address*, in the first schema maps to a combination of the three object sets in the second schema. This type of problem is called a *1:n mapping cardinality* problem (or an *n:1 mapping cardinality* problem). More generally, there may be $n$ object sets in one schema and $m$ object sets in another schema that correspond. For example, one schema may have object sets for *Day Phone* and *Evening Phone* while the other has object sets for *Home Phone*, *Office Phone*, and *Cell Phone*. This type of problem is called an *n:m mapping cardinality* problem, or called, by some researchers, a *cluster mapping* problem. To date, most of the research has focused on computing 1:1 mappings [2, 8, 5, 10, 11, 12, 13, 14]. Besides ourselves, some other researchers have also begun to study 1:$n$ mappings [5, 11, 12, 13, 14, 3], but very few have even touched on ways to automate $n$:$m$ mappings [14].

The cardinality denotations for schema mappings are from [14]. In our work we have used the terms *direct mapping* to denote 1:1 mappings and *indirect mapping* to denote the remaining types of mappings, 1:$n$, $n$:1, and $n$:$m$ [17]. Considering real-world examples, we have found that the percentage of indirect mappings is significant: 35% (221/638) for the three domains (*Car Advertisements*, *Cell Phones*, and *Real Estate*) for which we counted the occurrences. Another study [5] reported that the percentage is 22% (97/437) for the two domains they considered: *Course Catalog* and *Company Profile*.

In this paper, we focus particularly, though not exclusively, on indirect mappings. Our objective is to explain the techniques we have implemented for schema mapping, to share the experiences we have had with the automatic schema mapping prototypes we have built, and to share the lessons we have learned. Section 2 describes our schema-mapping techniques. These techniques rely heavily on two resources we use: data frames [6] and domain ontology snippets [7]. These resources are a key factor in automatically discovering both direct and indirect matches. In Section 3 we discuss the experiences we have had testing our prototype implementations. In Section 4 we summarize and list the lessons we have learned.

## 2. AUTOMATIC SCHEMA MAPPING

In this section, we first describe our schema representations. Next, we describe the resources on which we rely to discover schema mappings. We then use examples to explain the mapping process and the resulting output mappings. Details supporting this example-based discussion are in [1] and [16].

### 2.1 Schema Representation

We write schemas as conceptual graphs. Figures 1 and 2 show two schemas we use as examples. Each schema has an *object/relationship-model instance* that describes sets of objects, sets of relationships among objects, and functional constraints over object and relationship sets. An object set contains either data values or object identifiers, respectively called a *lexical object set*, which we denote by a dotted box, or a *nonlexical object set*, which we denote by a solid box. Each edge represent a relationship set that contains tuples of objects representing relationships among connecting object sets. The arrows in an edge specify functional constraints from domain object set to range object set.

### 2.2 Schema Mapping Resources

Two schema mapping resources unique to our work are *data frames* [6] and *domain ontology snippets* [7].

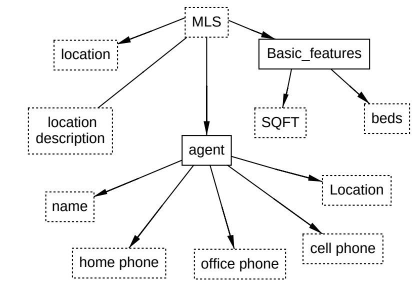


**Figure 1: Conceptual-model Graphs for Schema 1**

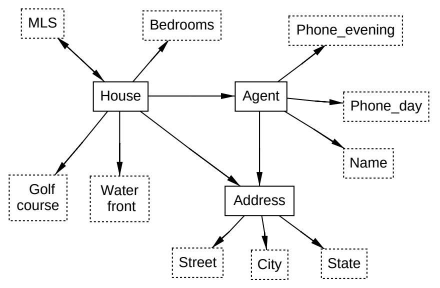


**Figure 2: Conceptual-model Graphs for Schema 2**

### 2.2.1 Data Frames

A *data frame* encapsulates the essential properties of everyday data items such as currencies, dates, weights, and measures. A data frame extends an abstract data type to include not only an internal data representation and applicable operations but also highly sophisticated representational and contextual information that allows a string that appears in a text document to be classified as belonging to the data frame. Thus, for example, a data frame for a phone number has regular expressions that recognize all forms of phone numbers and regular expression recognizers for keywords such as "cell phone," "home phone," and "day phone" to distinguish phone numbers from one another. Data frames may also simply have lists of values such as a list of all country names; they may also simply have sample values such as some well-known city names.

With each object set of interest we can associate a data frame to help us recognize values that may belong to the object set. Thus, for example, if we see a phone numbers in object sets from two different schemas, we have evidence that the two object sets may map to one another. Furthermore, as we shall show in Section 2.3, data-frame recognizers can help us split and join string values, which lets us match atomic string components of different granularities, and thus helps us resolve some 1:$n$, $n$:1, and $n$:$m$ mapping cardinality problems.

### 2.2.2 Domain Ontology Snippets

We have developed a type of light-weight domain ontology for data extraction [7]. Each data-extraction domain ontology describes the relationships among a collection of object sets in a specified domain. Molecular components of these extraction ontologies, which we call *domain ontology snippets*, describe a small, conceptual aggregate of information such as names being aggregates of titles, first names, surnames, and optional suffixes such as Jr. or Sr. Domain ontology snippets are similar to ontology base components as described in [15] and they serve the same purpose: to declare plausible domain fact types.

We represent domain ontology snippets as conceptual graphs. Figure 3 shows a domain ontology snippet for phones, which specifies the object set $Phone$ as a possible superset of the object sets $Day\ Phone$, $Evening\ Phone$, $Home\ Phone$, $Office\ Phone$, and $Cell\ Phone$.[1] Figure 4 shows a domain ontology snippet, which is a component of a larger real-estate domain ontology. The lot-feature ontology snippet specifies that $Lot\ Feature$ is a possible superset of $View$ and $Lot\ Size$ values and individual lot features $Water\ Front$, $Wooded$, etc.[2] As we show in Section 2.3, domain ontology snippets can provide guidance for resolving 1:$n$ and $n$:$m$ mapping cardinality problems.

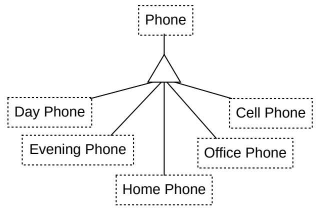


**Figure 3: Domain Ontology Snippet for Phone**

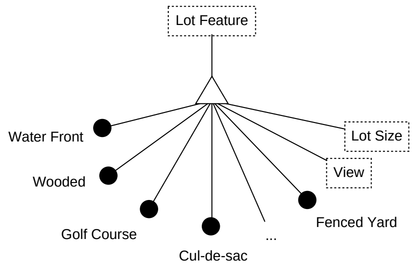


**Figure 4: Domain Ontology Snippet for Lot Feature**

## 2.3 Schema Mapping

A program that finds a mapping between two schemas is a *matcher*. During our studies, we found that different matchers have their own benefits and shortcomings. We therefore use a combination of multiple matchers to do schema mapping. The results from different matchers can corroborate and cross-validate each other to improve overall results.

There are different ways to classify matchers. We consider (1) *instance-level* versus *schema-level matchers*—matchers that use data-value instances versus those that use schema-level meta-information and (2) *object-set* versus *structure matchers*—matchers that compare individual object sets versus those that compare the structural similarities of two different schemas. As our main alternatives, we discuss matchers as object-set matchers and structure matchers, but in each section we discuss instance-level and schema-level matchers as well.

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

[2]Large black dots represent individual objects.

### 2.3.1 Object-set Matchers

Object-set matchers find correspondences based on similarities between the object sets of two schemas. Widely adopted techniques include name and description matching at the schema level and value matching at the instance level [14]. In our work we have used three object-set matchers: a name matcher, a value-characteristic matcher, and a data-frame matcher.

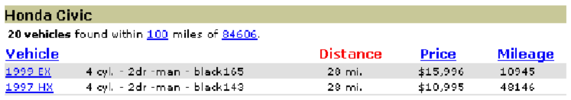
The name matcher matches object sets comparing the similarities of object-set names in the two schemas. In the implementation, we not only compare strings and substrings of the object-set names, but also provide some linguistic aids such as stemming, removing ignorable characters, and using a stop word list and digital synonym dictionaries. The particular synonym dictionary we use is WordNet.[3] WordNet not only defines a synonym list for each word, but also gives multiple senses of each word and places each word in a hypernym/hyponym hierarchy. To obtain a mapping rule we trained a C4.5 decision-tree generator over WordNet characteristics using several hundred synonym names found in a wide variety of database schemas, which were readily available to us.

Similar to [10], the value-characteristics matcher considers whether the values in two object sets have similar value characteristics such as means and variances for numerical data and string lengths and alphabetic/non-alphabetic ratios in alphanumeric data. As we did for our name matcher, we obtained a mapping rule by training a C4.5 decision-tree generator over value characteristics using sets of data values readily available to us.

The data-frame matcher finds instance-level mappings by applying data-frame recognizers to identify data values in object sets. Unlike more traditional instance matchers that directly compare data instances to measure similarity, our data-frame matcher compares object values indirectly. Using data-frame recognizers, we extract values from object sets as described in [7]. Then, if the same data frame recognizes values from two different object sets, there is a strong likelihood that these object sets match. As a motivating example, consider two object sets in two different schemas: if one contains instance data values *Ford* and *Honda* and the other contains data values *Dodge* and *Toyota*, most people can easily declare a match since both of them represent car models. Straight instance comparison, however, would not suggest a match. The use of a data-frame matcher not only solves this type of problem, but also helps even when the values do significantly overlap.

Our name matcher, value-characteristic matcher, and basic data-frame matcher can all help solve the 1:1 mapping cardinality problem. The data-frame matcher, however, can also help solve a particular kind of 1:$n$ and $n$:1 mapping cardinality problems. When we apply a data frame to recognize data values, we can recognize proper substrings of values as well as full strings. For example, a data frame may recognize the full string "120 N. University Ave., Provo, UT" as an instance of *location* in Figure 1. Considering *Street* in Figure 2, a data frame might see "120 N. University Ave.", which is not an instance of an address, but rather a substring of an address. Similarly, it would also recognize the instances of *City* (e.g. "Provo") and *State* (e.g. "UT") in Figure 2 as substrings of address. It is easy to see that the

[3]http://www.cogsci.princeton.edu/∼wn/

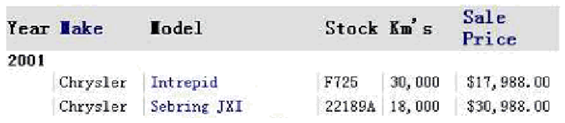


Honda Civic

**20 vehicles** found within 100 miles of 84606.

| Vehicle | | Distance | Price | Mileage |
|---|---|---|---|---|
| 1999 EX | 4 cyl. - 2dr -man - black165 | 28 mi. | $15,996 | 10945 |
| 1997 HX | 4 cyl. - 2dr -man - black143 | 28 mi. | $10,995 | 48146 |

**Figure 5: Car Table 1**



| Year | Make | Model | Stock | Km's | Sale Price |
|---|---|---|---|---|---|
| 2001 | | | | | |
| | Chrysler | Intrepid | F725 | 30,000 | $17,988.00 |
| | Chrysler | Sebring JXI | 22189A | 18,000 | $30,988.00 |

**Figure 6: Car Table 2**

combination of the three substrings produces a full address, and thus we can map the three object sets in Schema 2—*Street*, *City*, and *State*—together to the object set *location* in Schema 1 and vice versa. This technique can also suggest partial matches as well. If there were no object set *State* in Schema 2, a matcher could suggest the combination of *Street* and *City* as a match for *location*, but with lower confidence.

The data-frame matcher can also help solve a particular kind of $n$:$m$ mapping cardinality problem. Figures 5 and 6 show two HTML tables for car advertisements. The first table has an implied attribute[4] *MakeModel* (for *Honda Civic*) and a regular attribute *Vehicle* (for *1999 EX 4 cy - 2dr - man - black165*, the year, trim, number of cylinders, number of doors, transmission, color, and stock number). The second table has four matching elements, which are *Year*, *Make*, *Model* which includes the model and the trim (see for example *Sebring JXI*), and *Stock*. Assume we have data frames for *CarMake*, *CarModel*, *CarTrim*, *Year*, *CarFeature* (which includes number of cylinders, number of doors, and transmission), and *Color*, but not for *CarStockNumber*. When we check the object sets (attributes) in Table 1, we find that values recognized for *MakeModel* are a concatenation of the values recognized by the data frames *CarMake* and *CarModel* and that the values recognized by the data frames *Year*, *CarTrim*, *Feature*, and *Color* are all substrings of *Vehicle*. On the other hand, in Table 2, the values for *Year* are recognized by the data frame for *Year*, the values for *Make* are recognized by the data frame *CarMake*, and the values for *Model* are recognized as a concatenation of the values recognized by the data frames *CarMake* and *CarTrim*. Therefore, the mapping program detects that the cluster of object sets *MakeModel* and *Vehicle* in the Table 1 maps to the cluster of the object sets *Year*, *Make*, and *Model* in the Table 2, which is a 2:3 mapping. Further, the mapping program knows how to extract, split, and concatenate the various strings to build the appropriate values for the mapping.

### 2.3.2 Structure Matchers

Structure matchers use structural information both to suggest mappings and to confirm mappings suggested by object-set matchers. In our work, we not only consider using the structural similarity of the two schemas, but we also

[4]Implied attributes are common in web tables. Once we have a matching data frame (or in this case two matching data frames, *Make* and *Model*), we can supply an attribute name for an implied attribute by using the name of the data frame (or in the case of several matching data frames, a concatenation of the names of the data frames) as the attribute name.

use reference structures, available to us as domain ontology snippets (as discussed in Section 2.2.2). The idea is similar to using data frames as intermediaries—it may be difficult to map a structure $A$ directly to a structure $B$, but easy to map both $A$ and $B$ to the structure $C$, which may help us decide that $A$ corresponds to $B$.

With the help of the ontology snippet in Figure 4, for example, we can discover that the object set *location description* in Figure 1 maps to the pair of object sets *Golf course* and *Water front* in Figure 2. The object set *Lot Feature* in the ontology snippet in Figure 4 matches with *location description* in Figure 1 in the usual way. The match between the ontology snippet in Figure 4 and the object sets in Figure 2, however, is unusual. When we look at the values for *Golf course* and *Water front*, we find that they are Boolean values, "Yes" or "No", stating whether the house is next to a golf course or is on water-front property. Thus, although values do not match, the object set names do match values that are in the *Lot Feature* object set. Hence, we can convert the "Yes" and "No" values to *Lot Feature* values and consider a combination of the two converted object sets as a single object set with a many-many relationship-set connection to *House*. This virtual object set for Schema 2 now matches directly with the object set *location description* in Schema 1. Thus, the way in which object sets in Schema 1 and Schema 2 match to the intermediate ontology snippet allows the system not only to match the object set *location description* in Figure 1 with the pair of object sets *Golf course* and *Water front* in Figure 2, but also allows the system to know how they match.

As another example of using ontology snippets as intermediaries consider mapping the various phones in Figures 1 and 2 to one another. Using object-set matchers alone, the mapping program can detect that the object set *cell phone* in Figure 1 may match either *Phone_evening* or *Phone_day* or a combination of both in Figure 2. Object-set matchers can suggest similar mappings for *home phone* and *office phone* as well. If we choose any injective subset of the possible mappings, we will probably be wrong. We can, however, appeal to the ontology snippet in Figure 3, which indicates that the five individual object sets are all subsets of the object set *Phone* with no constraints about whether any one phone can be in two or more subsets. Therefore, the best way to map them together may be to establish a mapping that maps the cluster *cell phone*, *home phone*, and *office phone* in Schema 1 with the cluster of *Phone_evening* and *Phone_day* in Schema 2. To determine the phone mappings fully automatically, information about day and evening phones must be associated with individual phones in Schema 1 and information about cell, office, and home phones must be associated with individual phones in Schema 2. If so, data-frame extraction which considers context keywords can help; otherwise even a person would not likely be able to fully sort out the mapping among phones.

In our example, we map the object set *MLS* in Figure 1 with the object set *MLS* in Figure 2. However, suppose the object set *MLS* were not in Figure 2. In this case, we would like to match *MLS* in Figure 1 with *House* in Figure 2 (otherwise we would be missing a fundamental mapping between the schemas). With the help of an ontology snippet containing *House*, *MLS*, and a bijection declaring the equivalence between them, we can declare a mapping between *MLS* and *House*. There is likely to be no way to automatically establish this mapping without the help of such a real-estate ontology snippet.

Another interesting example is the direct match between object set *beds* in Figure 1 and object set *Bedrooms* in Figure 2. The context does not initially appear to be the same since *beds* connects to *Basic_features* in Schema 1 and connects to *House* in Schema 2. However, it is typical to model relationships both directly with relationship sets and indirectly with multiple relationship sets through named object sets (e.g. we can think of a direct uncle relationship, or we can think of an uncle as a brother of a parent). This transitivity is further strengthened if constraints, such as functional constraints, are identical for both direct and transitive relationships. In Figure 1, $MLS$ functionally determines *beds* through *Basic_features*, and in Figure 2, $MLS$ functionally determines *Bedrooms* through *House*. This along with with name matchers declaring the object sets to be possible synonyms, value-characteristic matchers observing that both contain small integer numbers, and a possible ontology snippet providing knowledge that houses typically have a small number of bedrooms yields sufficient evidence to declare a match.

As a final example, consider the 2:1 mapping from the two location object sets in Figure 1 to the object set *Address* in Figure 2, which is an aggregate of *Street*, *City*, and *State*. First, the object-set matchers suggest two 1:1 mappings between either of the location object sets and *Address*. Then, by observing functional constraints, the system detects that the two locations are functionally determined by the object sets *agent* and *MLS* in Figure 1 and similarly that *Address* is functionally determined by the object sets *Agent* and *MLS* (through *House*) in Figure 2. The combination of all this information suggests that we need to establish a 2:1 mapping between the two location object sets in Figure 1 and the object set *Address* in Figure 2.

## 2.4 Mapping Output

After discovering the mappings, we could choose to output them in several different ways. We have chosen to output our mappings in terms of an extended relational algebra [1, 16]. The idea is to create a view over one of the schemas so that the view identically matches the schema of the other. As a result, we can directly query one schema in terms of the other and obtain results by simple query unfolding.

The mappings can be written as views for either of the schemas. Complementary operators such as union and selection and, in the extended algebra, value-split/value-merge and attribute-to-value/value-to-attribute transformations, let us choose either schema as the source for the view or the target to which the view should correspond.[5] We illustrate with some examples.

If we let Figure 2 be the source and Figure 1 be the target, we can write

$$MLS_1\text{—}agent_1 = \pi_{MLS_2, Agent_2}(MLS_2\text{—}House_2 \bowtie House_2\text{—}Agent_2)$$

where subscripts designate the schema for an object set and the notation with a dash, $A$—$B$, denotes a relationship set. Note that after the virtual relationship set on the right-

[5]For many real-world applications, there is often a natural target, such as a predetermined global schema, to which a source should conform. In this case, we would choose the natural source and target and map in only one direction.

hand-side of this view definition has been established for Schema 2, there is a one-to-one correspondence between it and the *MLS*—*Agent* relationship set in Schema 1.

In the other direction, if we let Figure 1 be the source view and Figure 2 be the target view, we can write

$$House_2\text{—}MLS_2 = \varphi_{f_{House_1}}(MLS_1)$$

as a view definition. The view expression here uses a skolemization operator, $\varphi_{f_A} r$, where $f_A$ is a skolem function that produces object identifiers for a new attribute $A$ and places them in a one-to-one correspondence with the tuples in $r$. As a result, $\varphi_{f_{House_1}}(MLS_1)$ populates a new virtual object set $House_1$ with object identifiers and places the set of object identifiers in $House_1$ in a one-to-one correspondence with the values in $MLS_1$ via a virtual relationship set $House_1$—$MLS_1$. Again, note that after the virtual relationship set on the right-hand-side of this view definition has been established for Schema 1, there is a one-to-one correspondence between it and the *House*—*MLS* relationship set in Schema 2.

As a final example, we write the mapping for *location* in Schema 1 over the corresponding object sets in Schema 2:

$$MLS_1\text{—}location_1 = \pi_{MLS_2, location_2}(MLS_2\text{—}House_2 \bowtie \pi_{House_2, location_2}\lambda_{(Street_2, ",", City_2, ",", State_2), location_2} (House_2\text{—}Address_2 \bowtie Address_2\text{—}Street_2 \bowtie Address_2\text{—}City_2 \bowtie Address_2\text{—}State_2))$$

Here, $\lambda_{(A_1,\ldots,A_n),A} r$ is a composition operator that forms a new relation by adding a new column $A$ to $r$ whose value for tuple $t$ in $r$ is a concatenation, in the order specified, of the $A_i$'s and the string literal values among the $A_i$'s for tuple $t$ in $r$. Thus, the expression forms a virtual relationship set between $MLS_2$ and the new, virtual object set $location_2$ whose values are comma-separated strings created by concatinating $Street_2$, $City_2$, and $State_2$ values.

The relational-algebra expressions we generate are only mapping expressions for direct and indirect correspondences between the schemas. These mapping expressions neither convert object-set values nor resolve object identity, both of which are needed for schema merging and interoperative queries, which are the typical goals of integration and of which schema mapping is only a part.

## 3. EXPERIENCES WITH EXPERIMENTS

We have evaluated the performance of our approach based on recall and precision. *Recall* measures the number of correct mappings our system generates with respect to the number of correct mappings it should have generated. *Precision* measures the number of correct mappings with respect to the total number of correct and incorrect mappings the system generated. We have considered several real-world applications, including *Car Advertisements*, *Music CDs*, *Course Schedule*, *Faculty Information*, *Real Estate*, and *Cell Phones*.

In our earliest work on schema mapping, we considered only direct mappings [8]. By using a name matcher, a value-characteristics matcher, a data-frame matcher, and a primitive structure matcher, we were able to obtain recall and precision measures above 90%. We also confirmed our belief that multiple matchers typically work better than a single matcher. When we removed the data-frame matcher, the recall and precision for direct mappings for *Car Ads* dropped a half dozen percentage points and dropped even more for *Music CDs*.

To evaluate both direct and indirect mappings [17, 18], we experimented with data downloaded from [4] for three applications: *Course Schedule*, *Faculty Information*, and *Real Estate* [17, 18]. We faithfully translated the schemas from DTDs to conceptual-model graphs. For each application there were five schemas. We tested each application 10 times, once for each unique pair of schemas. In the results the recall and precision of all three applications were over 90%. In order to evaluate the contribution of data frames and domain ontology snippets, we made two additional test runs for just the *Real Estate* application, in which most of the indirect mappings occurred. Among the schemas we tested, $n$:$m$ mapping cardinality problems occurred 4 times and $n$:1 or $n$:1 problems occurred 48 times. All together, including both direct and indirect mappings, there were a total of 876 object-set and relationship-set matches. In the first run, without data frames and domain ontology snippets, the system achieved 73% recall and 67% precision. In the second run, with data frames and ontology snippets, the performance improved dramatically and reached 94% recall and 90% precision.

In an entirely different experiment, we considered HTML tables found on the web similar to the ones in Figures 5 and 6 for two applications: *Car Advertisements* and *Cell Phones* [9]. We collected a total of 46 *Car Ads* tables and 12 *Cell Phone* tables. For the 46 *Car Ads* tables, there were 319 mappings, of which we correctly discovered 296 (93%), missed only 23 (7%), and (incorrectly) declared only 13 false mappings (4%). Of the 296 mappings we correctly discovered, 147 (50%) were indirect (1:$n$, $n$:1, or $n$:$m$) mappings. For the 12 *Cell Phone* tables, there were 103 mappings, of which we correctly discovered 88 (85%), missed 15 (15%), and (incorrectly) declared 9 false mappings (9%). Of the 88 mappings we correctly discovered, 50 (57%) were indirect mappings.

One limitation to our approach is the need for creating data frames and domain ontology snippets. They are all manually constructed at this time. Our experience in teaching others to use our system, however, suggests that typical data frames and domain ontology snippets can be created in a few person-hours.

## 4. SUMMARY AND LESSONS LEARNED

Based mainly on two unique techniques—data frames and domain ontology snippets—we presented a way to automatically discover many direct and indirect mappings between the elements of two schemas. Based on our experience, we pass along the following lessons, which we have learned.

1. Indirect matches between schemas are common. For the application domains we considered, it was typical for us to find that 20 – 40% of the mappings were indirect. In some applications, the percentage reached above 50%.

2. Data-frame recognizers can increase schema mapping accuracy. When data-frame recognizers are not used, experimental evidence has shown a decrease in recall and precision percentages. This should not be surprising, since by introspection most people realize that recognizing common or expected data values in context

helps increase understanding, especially in semistructured data or data-rich applications.

3. Domain ontology snippets can increase schema mapping accuracy. Combined with data-frame matchers, these larger granularity patterns appear to be helpful. Experimental evidence has shown dramatic drops in recall and precision percentages when neither is part of the suite of matchers. Again, this should not be surprising. Most people realize that recognizing common or expected patterns of small groups of meaningful value combinations increases understanding.

4. Multiple matchers working together can increase schema mapping accuracy. Experimental evidence shows that dropping one of the matchers for some applications can decrease recall and precision percentages by several percentage points. It appears that different matchers can work well in different situations. In order to achieve the best mapping performance, we probably need to combine several different object-set matchers and structure matchers together and use both schema-level and instance-level matchers to cross-validate results. It is likely that we can use the cumulative work of the larger research community together to help resolve schema-mapping problems.

## 5. ACKNOWLEDGMENTS

We acknowledge the contributions of many discussions with our colleagues in the Data Extraction Group at Brigham Young University and with Joachim Biskup at Dortmund Universität. This work has been partially funded by the National Science Foundation under grant number IIS-0083127.

## 6. REFERENCES

[1] J. Biskup and D. Embley. Extracting information from heterogeneous information sources using ontologically specified target views. *Information Systems*, 28(3):169–212, May 2003.

[2] S. Castano, V. D. Antonellis, and S. D. C. di Vemercati. Global viewing of heterogeneous data sources. *IEEE Transaction of Data Knowledge Engineering*, 13(2):277–297, 2001.

[3] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering complex matches between database schemas. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD 2004)*, Paris, France, June 2004.

[4] 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'01)*, pages 509–520, Santa Barbara, California, May 2001.

[5] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the semantic web. *The VLDB Journal*, 12(4):303–319, November 2003.

[6] D. Embley. Programming with data frames for everyday data items. In *Proceedings of AFIPS National Computer Conference (NCC'80)*, pages 301–305, Anaheim, California, May 1980.

[7] 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.

[8] 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. An extended version of this paper appeared in *Journal of the Brazilian Computing Society*, 8(2):32–43, November, 2002.

[9] D. Embley, C. Tao, and S. Liddle. Automatically extracting ontologically specified data from HTML tables with unknown structure. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER'02)*, pages 322–327, Tampere, Finland, October 2002. An extended version of this paper is to appear in *Data & Knowledge Engineering*.

[10] W. Li and C. Clifton. Semint: a tool for identifying attribute correspondences in heterogeneous databases using neural network. *Data Knowledge Engineering*, 33(1):49–84, 2000.

[11] J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 49–58, Rome, Italy, September 2001.

[12] D. Melnik, H. Molina-Garcia, and E. Rahm. Similarity flooding: a versatile graph matching algorithm. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 117–128, San Jose, California, February 2002.

[13] L. Popa, Y. Velegrakis, M. Hernandez, R. Miller, and R. Fagin. Translating web data. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB'02)*, pages 598–609, Hong Kong, China, August 2002.

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

[15] P. Spyns, R. Meersman, and M. Jarrar. Data modeling versus ontology engineering. *SIGMOD Record*, 31(4):12–17, December 2002.

[16] L. Xu. *Source Discovery and Schema Mapping for Data Integration.* Brigham Young University, Provo, Utah, 2003. PhD Dissertation.

[17] L. Xu and D. Embley. Discovering direct and indirect matches for schema elements. In *Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA 2003)*, pages 39–46, Kyoto, Japan, March 2003.

[18] L. Xu and D. Embley. Using domain ontologies to discover direct and indirect matches for schema elements. In *Proceedings of the Semantic Integration Workshop Collocated with the Second International Semantic Web Conference (ISWC-03)*, Sanibel Island, Florida, October 2003.