AUTOMATIC DISCOVERY OF SEMANTIC RELATIONSHIPS BETWEEN SCHEMA ELEMENTS

Nikos Rizopoulos
Imperial College
London, England
Email: nr600@doc.ic.ac.uk

Key words: Automatic schema matching, semantic relationships, data integration.

Abstract: The identification of semantic relationships between schema elements, or *schema matching*, is the initial step in the integration of data sources. Existing approaches in automatic schema matching have mainly been concerned with discovering equivalence relationships between elements. In this paper, we present an approach to automatically discover richer and more expressive semantic relationships based on a bidirectional comparison of the elements data and metadata. The experiments that we have performed on real-world data sources from several domains show promising results, considering the fact that we do not rely on any user or external knowledge.

1 INTRODUCTION

The integration of heterogeneous data sources is a well-known research subject. Its key issue is the identification of semantic relationships between schema elements (Kashyap and Sheth, 1996), which is a labor-intensive and time-consuming process when performed manually. Automatic schema matching resolves this problem by automatically discovering semantic relationships between schema elements.

Several approaches can be found in the literature concerned with automatic schema matching. Most of them are focused on discovering equivalence relationships between elements. However, in many cases more expressive relationships exist. For example, element person subsumes student and the elements postgraduate and undergraduate can be merged into student. Such matches are called *indirect* (Xu and Embley, 2003).

Example: Figure 1 illustrates cut-down representations of two databases of the Computing Department at Imperial College, London.

Schema S_1 shows that staff members tutor undergraduate students and supervise PhD students. In S_1, the element course represents all the non-laboratory courses, i.e. all the courses that have lectures in theatres. Undergraduate students register on these courses and members of staff teach them. In the college, PhD students assist in both tutorials and lab demonstrations. This is depicted in schema S_2, where course describes both laboratory courses and courses that have tutorials. Element staff in S_2 represents the members of staff that supervise tutorials and laboratories and can be both lecturers or teaching assistants.

We have asserted the constraints that each PhD student has to assist in at least one course and each lecturer has to teach at least one non-laboratory course and supervise one laboratory course. Also, non-laboratory courses might not have any tutorials. These constraints implicitly express that the phd elements in S_1 and S_2 represent identical sets of PhD students (direct match), that the concept of staff in S_2 subsumes the concept of staff in S_1 (indirect match) and that the two course elements intersect since both include those non-laboratory courses that have tutorials (indirect match).

In this paper, we describe a framework to automatically discover matches like the ones in the example. Our goal is to identify semantic relationships between elements without relying on external knowledge, like ontologies, user-knowledge or schema structure. We adopt a *composite* approach that exploits several types of information (element names, data instances, statistical information on the data) to discover **incompatible**, **disjoint**, **intersecting**, **subsumming** and **equivalent** elements. Our methodology performs a bidirectional comparison of the elements, which proves to be indicatory of these types of semantic relationships.

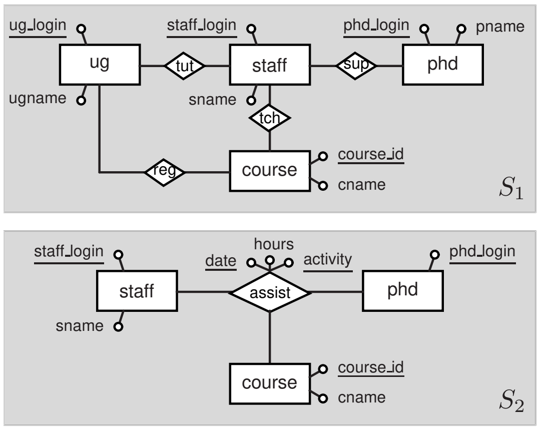


Figure 1: Source Schemas S_1 and S_2

This paper contributes in the formal definition of semantic relationships between schema elements and the automatic discovery of these relationships. As far as we know, no other automatic schema matching approach discovers **disjointness** and **intersection** relationships without relying on external knowledge. In addition, this paper proposes an innovative composite architecture which differentiates between modules that identify schema matches and modules that clarify the type of the relationship in each match.

The structure of this paper is as follows. In Section 2, we define the five types of semantic relationships that our methodology identifies. Section 3 explains the way a bidirectional comparison of schema elements can assist in the clarification of their semantic relationship. Section 4 shows the architecture of our framework and describes the implemented relationship identification and relationship clarification modules. In Section 5 the results of the experiments we have conducted to evaluate our approach are presented. Section 6 describes related schema matching approaches and Section 7 gives our concluding remarks and directions for further work.

2 SEMANTIC RELATIONSHIPS

Various types of semantic relationships between schema elements have been defined in the literature. In (Larson et al., 1989), a non-automated approach for schema integration is proposed, based on manually identified semantic relationships between elements. We adopt similar relationship definitions, except from **disjointness** and **incompatibility**. We define $Inst_{ext}(x)$ to be the instances of an element x that are currently stored in the data source, and $Dom_{ext}(x)$ the extentional domain of the element, i.e. all its possible valid instances. We also define as $Ent_{int}(x)$ the intentional entities of x, i.e. the real-world entities that map to the instances of $Inst_{ext}(x)$, and $Dom_{int}(x)$ the intentional domain of x, i.e. the real-world entities that map to $Dom_{ext}(x)$.

Five types of semantic relationship between schema elements are identified based on the comparison of their intensional domains. These relationships are:

1. **equivalence**: Two schema elements A and B are equivalent, $A = B$, iff
   $Dom_{int}(A) = Dom_{int}(B)$
2. **subsumption**: Schema element A subsumes schema element B, $B \subset A$, iff
   $Dom_{int}(B) \subset Dom_{int}(A)$
3. **intersection**: Two schema elements A and B are intersecting, $A \cap B$, iff
   $Dom_{int}(A) \cap Dom_{int}(B) \neq \emptyset$,
   $\exists C : Dom_{int}(A) \cap Dom_{int}(B) = Dom_{int}(C)$
4. **disjointness**: Two schema elements A and B are disjoint, $A \not\cap B$, iff
   $Dom_{int}(A) \cap Dom_{int}(B) = \emptyset$,
   $\exists C : Dom_{int}(A) \cup Dom_{int}(B) \subseteq Dom_{int}(C)$
5. **incompatibility**: Two schema elements A and B are incompatible, $A \neq B$, iff
   $Dom_{int}(A) \cap Dom_{int}(B) = \emptyset$,
   $\neg\exists C : Dom_{int}(A) \cup Dom_{int}(B) \subseteq Dom_{int}(C)$

The notation $\exists C : condition$ means that there is a real-world concept that can be represented by an existing or non-existing schema element C that satisfies the $condition$. The notation $\neg\exists C : condition$ in the definition of incompatibility means that there is no real-world concept that would be represented by a schema element C to satisfy the specified *condition*.

Throughout this paper, we are going to use the term semantically *compatible* schema elements when elements are related with a semantic relationship other than incompatibility.

There are four pairs of compatible elements in the example of the previous section. The phd elements in S_1 and S_2 are **equivalent** because they represent identical sets of PhD students based on the constraint that each PhD student in S_1 must assist in at least one course, i.e. each entity of S_1.phd belongs to the set of entities represented by S_2.phd and vice versa. The elements ug and phd in S_1 and S_2 respectively are **disjoint** because each student in this particular example can be either an undergraduate or a PhD student. Thus, there is a concept (student) that subsumes the union of ug and phd. Element staff in S_2 **subsumes** staff in S_1, because S_2.staff represents all the lecturers, i.e. all the entities of S_1.staff, in addition to teaching assistants. The two course elements **intersect** because they have a common set of entities that can be represented by the concept courses_with_tutorials;

some courses of S_1 have tutorials, i.e. they belong to the entities represented by S_2.course, and the lab courses of S_2.course are not included in S_1.course.

3 DISCOVERING SEMANTIC RELATIONSHIPS

In order to discover the semantic relationships defined in the previous section, we perform a bidirectional comparison of the schema elements. Supposing that there are two schema elements X and Y, we define as $d(X, Y)$ the similarity degree produced by the comparison of element X against Y and $d(Y, X)$ the similarity degree produced by the comparison of Y against X. We call $d(X, Y)$ and $d(Y, X)$ *bidirectional similarity degrees*. Intuitively, the more similar X is to Y, the higher the similarity degree $d(X, Y)$ will be. Essentially, $d(X, Y)$ indicates to what extent X, with $Dom_{int}(X) \neq \emptyset$, is a subset of Y, ranging from 0, if none of the entities of X are entities of Y, to 1, if the set of entities of X is a proper subset of the entities of Y. This can also be described by the following formula:

$d(X, Y) = \frac{|Dom_{int}(X) \cap Dom_{int}(Y)|}{|Dom_{int}(X)|}$,

where $|S|$ defines the number of entities in set S.

The above formula will give high bidirectional similarity degrees for equivalent elements, high $d(X, Y)$ and low $d(Y, X)$ when Y subsumes X and average-high bidirectional degrees when the elements intersect. The problems arising when using this formula are: (a) it cannot be computed automatically since the comparison of the elements' real-world entities ($Dom_{int}(X)$) is required, and (b) it cannot differentiate between incompatibility and disjointness, since in both of these cases the intensional domains of the elements are disjoint, producing bidirectional similarity degrees with values equal to 0. A more detailed description of the problems arising when an *idealized* approach, like the above formula, is used to discover semantic relationships between schema elements can be found in (Rizopoulos, 2003).

In our framework, we attempt to resemble the same formula by examining the elements instances ($Inst_{ext}(X)$) and their metadata, e.g. data types, names, lengths, etc. Based on this information, even disjoint elements exhibit similarity, which arises from their relationship with the same *super* element C (see definition in the previous section). For example, usernames of PhD (phd_login) and undergraduate (ug_login) students follow the same format as any student username (login). Therefore, our approach resolves the problems mentioned previously: (a) automatic computation of the similarity degrees is feasible because element instances and metadata are freely

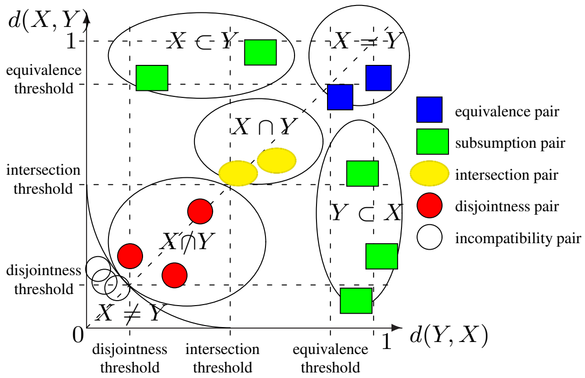


Figure 2: Bidirectional Similarity Comparison Graph

available and (b) disjointness and incompatibility can be distinguished because disjoint pairs of elements will have higher bidirectional similarity degrees than incompatible pairs. In the case of intersecting elements, their common set of entities in their intensional domains defines a common set of instances in their extensional domains, i.e. the relationship is preserved across the domains. Thus, intersecting pairs of elements are going to display higher similarity degrees than disjoint pairs. Relationship preservation also applies to subsumption and equivalence, which suggests that the same bidirectional similarity degrees should be expected whether the intensional or the extensional domains of the elements are examined.

Based on these observations, our insight on the way the bidirectional comparison can be applied in the identification of semantic relationships between elements is illustrated in Figure 2, the *bidirectional comparison graph*. The graph shows the areas where we expect the bidirectional degrees to position each pair of elements based on their semantic relationship. It is important to notice that the defined areas in the graph are fuzzy areas because they only represent an estimation of the expected results. For the same reason some areas of the graph are not covered.

4 ARCHITECTURE

In this section, we describe our framework's architecture (Figure 3) and present each implemented component.

Our framework consists of several comparison modules that exploit different types of information to determine the similarity of schema elements. These modules take as input the source schemas and their data instances and work independently to produce *partial* bidirectional similarity degrees. Partial in the sense that they are produced by just comparing partial

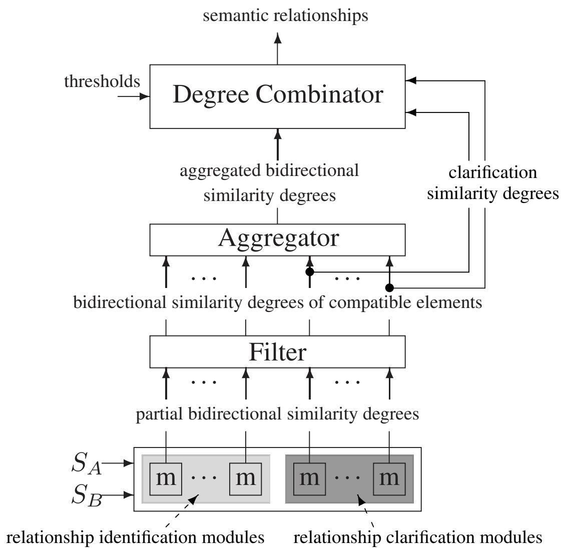


Figure 3: Architecture

information, e.g. element names only, and are therefore partially correct. These degrees are later combined to provide the final bidirectional similarity degrees, which indicate (according to the discussion in the previous section) the semantic relationships between the schema elements.

In our framework, there are two types of modules: *relationship identification* and *relationship clarification* modules. Relationship identification modules attempt to discover compatible pairs of elements and relationship clarification modules attempt to specify the type of the semantic relationship in each compatible pair.

Initially, the bidirectional similarity degrees produced by the modules are examined by the Filter component to separate the compatible from the incompatible pairs of elements. Then, the Aggregator combines the similarity degrees of the compatible elements and indicates their semantic relationships. It achieves this by mapping the compatible pairs onto the bidirectional graph. The output of the Aggregator becomes the input of the Degree Combinator, which based on the relationship clarification modules and the fuzzy areas defined in the bidirectional graph outputs the discovered semantic relationships. The user is then able to validate or reject these relationships and proceed to the data integration process.

All the components of the architecture that have been implemented in the prototype tool are now briefly described. More details can be found in (Rizopoulos, 2003).

The **Element Name Module** performs a case-insensitive comparison of element names. When element X has exactly the same name with Y then $d(X,Y) = d(Y,X) = 1$ and if X's name is a substring of Y's then $d(X,Y) = 0.2$ and $d(Y,X) = 1$. The similarity degrees have been chosen so that when two elements have the same names, then their bidirectional similarity degrees are going to map them in the equivalence area of the graph. If one element is a substring of another, e.g. login and phd_login, then this is an indication of a subsumption relationship, therefore the pair is mapped in the subsumption areas of the graph.

The rest of the modules operate in a similar way. The **Data Type Module** compares element data types. The **Numerical Statistics Module** compares numerical elements on their average value, medium value and the standard deviation of their instances. The **Non-numerical Statistics Module** compares non-numerical elements based on the average number of appearances of special characters (@, $, -, etc) in their instances. The **Instances Module** uses a Naive Bayes classifier to identify similarities between elements by comparing their instances. The **Number of Instances Module** is a naive module that is used for relationship clarification and compares the number of distinct instances of the elements. The **Precision Module** is a relationship clarification module that compares the range of each element's instances and the **Length Module** compares the range of the elements' lengths. The **Existence Module** is a relationship clarification module that examines the existence of instances in elements.

The **Filter** component separates the compatible from the incompatible pairs of elements. For each pair, it computes the average bidirectional similarity degrees and compares them to a user-defined threshold. The **Aggregator** component indicates the type of the semantic relationship for each pair of compatible elements by computing the product bidirectional similarity degrees. Modules in both the Filter and the Aggregator can have auxiliary roles, i.e. they can only increase the similarity degrees produced by other modules. The **Degree Combinator** uses the output of the relationship clarification modules and the Aggregator to determine the semantic relationships between the elements.

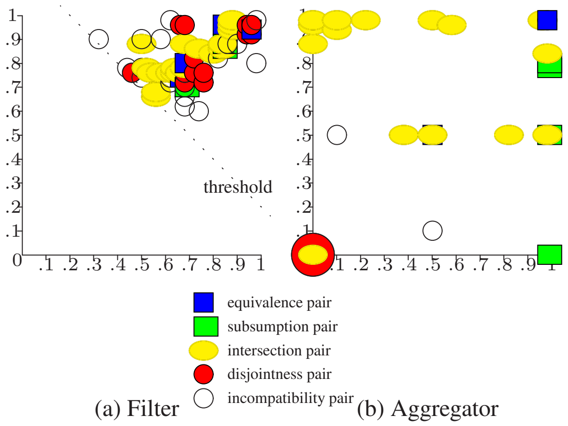
5 EXPERIMENTS

We have evaluated our prototype tool on three schema matching tasks of real-world data sources that come from three different domains.

The first task, called Pop&Geo, is between two data sources with geography and population data. The second task, University, is on two relational databases that store information about tutorials and students, and the third task, Real-Estate, in on two real-estate

	Pop&Geo	University	Real-Estate
#pairs	897	4389	868
#equivalents	0	7	0
#subsets	12	6	0
#intersections	8	46	0
#disjoints	18	57	18
#incompatibles	859	4273	850

Table 1: Problem Size and Schema Similarity in each Integration Task



(a) Filter (b) Aggregator

Figure 4: Filter and Aggregator Results

data sources that list houses for sale.

Table 1 presents the problem size and the schema similarity in each matching task. It shows the number of all possible pairs of elements between the schemas and for each type of semantic relationship the number of its appearances manually detected. The semantic relationships discovered by the tool are compared to these manually identified relationships.

Figure 4 illustrates the results of the Filter and Aggregator components for the University task. Each pair of elements is plotted using the appropriate symbol based on the manually identified relationship of the pair. The position where each pair is plotted depends on the bidirectional degrees produced by the tool. In Figure 4(a), the pairs of elements below the user-defined threshold have been omitted for clarity. As it can be seen, the Filter discards most of the incompatible pairs and the Aggregator attempts to map the compatible ones onto the correct areas of the comparison graph.

To examine the *reliability* and the *cover* of the tool, i.e. how many relationships are identified correctly and how many relationships are indeed identified, we have used *precision* and *recall*. If C is the number of the correctly identified relationships, F the number of the incorrectly identified relationships and A all the manually identified relationships, then precision is the fraction $C/(C+F)$ and recall is C/A.

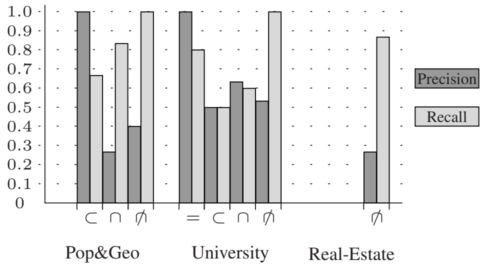


Figure 5: Precision and Recall

Figure 5 shows the precision and recall of the tool for each type of semantic relationship in each task. In the first task, the precision and recall bars for the equivalence relationship are missing because there do not exist any equivalent elements between the schemas. The same applies in the third task where there are not any equivalent, subsuming or intersecting elements.

In the first experiment, the low precision for intersection and disjointness comes from incompatible pairs of elements with intersecting ranges. The high recall in the first experiment means that very few relationships are lost and therefore the user will only have to reject the wrongly identified ones.

In the second experiment the main problems are caused by (a) elements that are sequences of numbers, in particular automatically incremented primary key attributes, (b) elements that have a small number of distinct instances, and (c) character elements whose instances are small strings.

In the third experiment, the precision for disjointness is affected by incompatible pairs of elements that should have been discarded by the Filter. The problems are caused by composite elements that contain multiple values in each one of their instances, and elements with small numerical instances.

Overall in the three experiments, the average precision for each semantic relationship is 100% for equivalence, 75% for subsumption, 46% for intersection and 39% for disjointness. The average recall is 77% for equivalence, 58% for subsumption, 71% for intersection and 96% for disjointness [1]. These results are encouraging, considering the fact that there was no user intervention on the data or the data sources. In addition, no external knowledge was used, like user-supplied training data, user-defined concept hierarchies, synonym tables, online ontologies, dictionaries, etc, neither any assumptions were made about the data instances or the sources.

[1]The average values for the equivalence relationship come from the University task, but in general we expect high precision and recall for equivalence in all tasks.

6 RELATED WORK

Several approaches concerned with automatic schema matching exist in the literature. Most of the approaches are focused in discovering equivalence relationships (A. Doan and Halevy, 2002; Madhavan et al., 2001), some of them also identify subsumption relationships (Bergamaschi et al., 1998) and some intersection (Hakimpour and Geppert, 2002). However, subsumption and intersection are discovered using external knowledge, like ontologies and thesauri, or user-knowledge. Our approach identifies equivalence, subsumption, intersection and disjointness relationships by only examining element metadata and data instances, without any user-intervention.

The work most related to ours is the one presented in (Xu and Embley, 2003), where direct and indirect matches between elements are discovered. Direct matches are identified between equivalent elements and indirect matches are identified between (a) subsuming elements, (b) boolean elements and elements whose instances contain the boolean elements' names, and (c) elements whose instances can be merged or splitted. These relationships are discovered based on schema information, ontologies and regular expressions defined to match the instances of elements.

Our framework covers all the relationships of (Xu and Embley, 2003), except from the last one (c) which in some cases is similar to our disjointness relationship. In the case of boolean elements, our methodology replaces their true and false instances with the elements' names and the concatenation of not and their names, respectively, since the actual instances do not provide much information. Therefore, if one element contains the name of a boolean element in its instances, this relationship will be identified. In our framework, we also identify intersecting elements that are not considered in (Xu and Embley, 2003).

GLUE (A. Doan and Halevy, 2002) is also similar to our work. It proposes a bidirectional comparison of schema elements, but it produces a single similarity degree which takes the lowest value when the elements do not have any common instances and the highest when the elements are equivalent. Therefore, the semantic relationships described in this paper cannot be discovered by this approach.

7 CONCLUSIONS

In this paper, we have presented our approach to automatically discover semantic relationships between schema elements. Based on a bidirectional comparison of the elements metadata and instances and without any user or external knowledge, we are able to discover equivalence, subsumption, intersection, disjointness and incompatibility relationships. We have shown our framework's architecture and described the components that we have implemented in the prototype tool. Our experimental results are promising with a 66% average precision and 75% average recall.

In the future, we are going to focus in the filtering process, since low precision has been mainly caused by incompatible pairs of elements that have not been discarded. We can consider assinging weights to modules based on their importance and reliability. Precision can also be improved by detecting automatically incremented elements and elements with small domains. A brute-force module can assist in this process and it would only impose a small overhead to exhaustively compare a small number of instances.

Additionally, in the future we are going to extend our prototype tool with a graphical user interface, which will permit the user to validate or reject the semantic relationships identified by our methodology, and a component which will integrate the input data sources based on the validated relationships.

REFERENCES

A. Doan, J. Madhavan, P. D. and Halevy, A. (2002). Learning to map ontologies on the Semantic Web. In *Proceedings of the World-Wide Web Conference (WWW-02)*, pages 662–673.

Bergamaschi, S., Castano, S., di Vimercati, S., Montanari, S., and Vincini, M. (1998). An intelligent approach to information integration. In *In International Conference on Formal Ontology in Information Systems (FOIS'98), Italy, 1998*, pages 253–267.

Hakimpour, F. and Geppert, A. (2002). Global schema generation using formal ontologies. In *Proceedings of ER02*, volume 2503 of *LNCS*, pages 307–321. Springer-Verlag.

Kashyap, V. and Sheth, A. (1996). Semantic and schematic similarities between database objects: a context-based approach. *VLDB Journal*, 5(4):276–304.

Larson, J., Navathe, S., and Elmasri, R. (1989). A theory of attribute equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*, 15(4):449–463.

Madhavan, J., Bernstein, P. A., and Rahm, E. (2001). Generic schema matching with Cupid. In *Proc. 27th VLDB Conference*, pages 49–58.

Rizopoulos, N. (2003). Discovery of semantic relationships between schema elements. Technical report, AutoMed Project.

Xu, L. and Embley, D. W. (2003). Discovering direct and indirect matches for schema elements. In *8th International Conference on Database Systems for Advanced Applications (DASFAA '03), Kyoto, Japan, March 26–28, 2003*, pages 39–46.