Discovery of Semantic Relationships between Schema Elements

AutoMed Technical Report 23, Version 1

Nikolaos Rizopoulos

Monday 18 th August 2003

Abstract

This report describes an approach to semi-automatically discover semantic relationships between schema elements. Based on these relationships data sources can be integrated as it has been previously shown in [14]. A bidirectional comparison of the schema elements is performed which enables the discovery of equivalent, subset, disjoint, overlapping and incompatible elements. We present a novel composite approach which includes different modules for semantic relationship identification and relationship clarification.

1 Introduction

Performing the data integration process manually is extremely time-consuming and considering the usefullness of this process, it is essential to be automated as much as possible. Several approaches can be found in the literature, dealing with automating data integration [5, 11, 2, 16, 10, 9, 15, 4, 3]. Most of them are focused on *schema matching*, i.e. identifying equivalent schema elements [12]. The discovery of other types of relationships, like subsumption, has been based either on a linguistic comparison of the elements or on the user's domain knowledge. This fact has motivated us in discovering such semantic relationships not only based on linguistic information but also utilizing metadata about elements, element instances and schema structure.

In order to achieve our goal, we perform a bidirectional comparison between schema elements. This approach allows the use of several kinds of information in the automatic identification of equivalence, subsumption, overlappingness, disjointness and incompatibility relationships [1]. Except from the innovative bidirectional comparison methodology to differentiate between each type of semantic relationship, our architecture is also novel. We apply different algorithms for the identification of related elements and different algorithms for the clarification of the type of their relationship.

The outline of this report is as follows. First, we show that even a brute-force algorithm cannot accurately discover semantic relationships between elements. In Section 3, we explain the bidirectional comparison approach and how it can be used to differentiate between each type of semantic relationship. Then, in Section 4, we present the architecture of our composite data integration approach. Section 5 explains in detail each component that has been implemented and illustrates the results of our prototype tool on real-world databases. Finally, we give our concluding remarks and directions of further work.

[1]Based on the definitions in [14].

2 Brute Force

The definitions of the semantic relationships in [14] indicate that the element entities need to be compared in order to determine semantic associations. Therefore, a brute-force approach, where the the element instances are compared, seems highly appropriate.

The implemented brute-force tool performs an exhaustive case-insensitive comparison of the elements' set of instances, called extensions $Val_{ext}(x)$. An element instance is represented as a string, thus an element's extension is a set of strings. For each pair of elements, a simple set-comparison operation is performed that examines if the elements' sets of instances are equivalent, if one set is a subset of the other, if the sets overlap or if the sets are disjoint. Based on the relationships identified between the extensions, the brute-force tool determines whether the elements are equivalent, subsets, overlapping or incompatible, respectively. Essentially, in the semantic relationships definitions the function $Dom_{int}(x)$ is replaced by $Val_{ext}(x)$. The set comparison is performed with the help of a hash table to reduce the complexity of the algorithm.

As it would have been expected the brute-force tool isn't accurate in discovering semantic relationships between schema elements. One reason is that different real-world entities might have the same representation. This problem may lead to producing false positive results, i.e. identifying compatibility relationships (equivalence, subsumption, overlappingness, disjointness) between incompatible schema elements. Additionally, when different representations are used for the same real-world entities, false negative results may be produced, i.e. identifying incompatibility between compatible elements. This can be partially solved by using more intelligent and complex techniques to compare instances, like string matching algorithms. However, such approaches lead to further time consumption, especially in an exhaustive comparison approach.

It is interesting to examine if the brute-force approach can provide an indication of the underlying relationships and assist in the data integration process when real-world entities have a unique representation. This assumption resolves the problem of false negative results, but false positives can still be produced. However, false positives are bearable because no compatibility relationships are lost.

Even with this assumption, one of the main problems that arise is the inability to detect disjoint schema elements. For example, if the sets of instances of two elements are disjoint then the approach is not able to identify whether the two elements are semantically disjoint or incompatible. This is due to the lack of knowledge about the semantics of the elements. Therefore, the brute-force approach can still produce false negative results and miss existing semantic relationships even if real-world entities have unique representations.

Another problem is the assignment of the wrong type of compatibility relationship between two compatible elements. This case is encountered when the data sources in their current state (when the data integration process is taking place) do not contain all the possible instances for their elements. For example, two data sources in their current state might indicate that two elements are equivalent because currently the elements have the same instances. However, semantically those two elements could be subsets or overlapping, a relationship which would be obvious if the elements in both data sources were populated with all their possible instances.

Thus, a brute-force approach except from exhibiting low efficiency and great time consumption, it is also not able to identify correctly all the existing semantic relationships. False negative results may be produced and compatibility relationships might be lost. However, the identified relationships could be helpful even if the type of the relationship is not always correct.

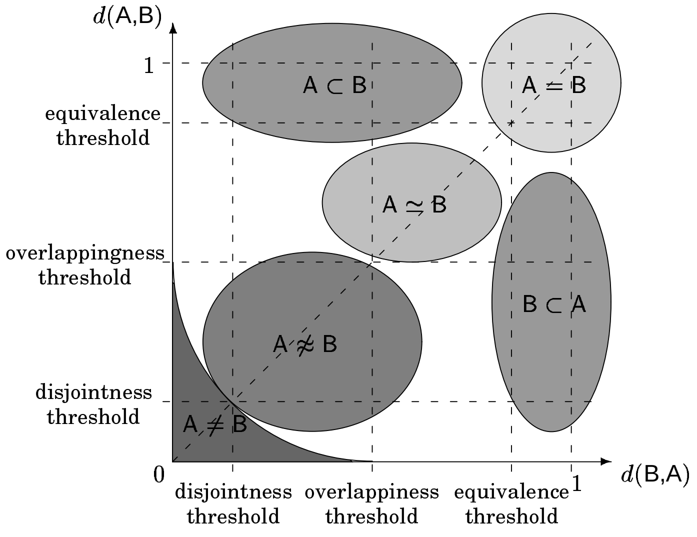


Figure 1: Bidirectional Similarity Comparison

3 Bidirectional Comparison

Most of the existing approaches that attempt to automate data integration discover equivalent schema elements and unify them. They perform a uni-directional comparison of the schemas by producing a similarity degree for each pair of elements. If this degree is above a pre-defined threshold then the elements are considered to be equivalent. The uni-directional comparison usually arises from the fact that the global schema is already known and it is attempted to match elements of the local schemas to the global schema.

In our approach, there is a bidirectional comparison of the schema elements which enables not only the identification of equivalence between elements, but also the discovery of subsumption, overlappingness, disjointness and incompatibility relationships. GLUE [1] is the only methodology, as far as we know, that performs a bidirectional comparison of the schemas, but only identifies disjoint (actually incompatible) and equivalent elements.

The idea behind the bidirectional comparison of schema elements comes from the fact that the size of the similarity degree between two schema elements indicates their similarity or dissimilarity. Supposing that there are two schema elements A and B in schemas S_A and S_B respectively, we define as $d(\mathsf{A},\mathsf{B})$ the similarity degree produced by the comparison of element A against B and d(B,A) the similarity degree produced by the comparison of B against A. Intuitively, the more similar A is to B, the higher the similarity degree $d(\mathsf{A},\mathsf{B})$ will be. The same reasoning applies for the reverse comparison of B against A. Figure 1 illustrates an insight on the way the bidirectional comparison can be applied in the identification of semantic relationships between schema elements.

Essentially, d(A,B) indicates to what extent A is a subset of B ranging from 0, if none of the entities of A are entities of B, to 1, if the set of entities of A is a proper subset ($\subset$) of the entities of B. When elements A and B are equivalent both similarity degrees $d(\mathsf{A},\mathsf{B})$ and $d(\mathsf{B},\mathsf{A})$ will be high. If A subsumes B, then $d(\mathsf{B},\mathsf{A})$ will be high since all the entities of B are also entities of A, but d(A,B) will be low since A includes entities that do not appear in B. In the reverse case where B subsumes A, d(A,B) is high and d(B,A) is low. When A and B are overlapping both similarity degrees are considerably high since entities of A appear in B and entities of B appear in A. On the other hand, when A and B are disjoint then the degrees are considerably low, since no common entities exist. Finally, if A and B are incompatible, the similarity degrees are close to 0.

The disjointness, overlappingness and equivalence thresholds in Figure 1 roughly define the areas that specify each type of semantic relationship. Two schema elements with bidirectional similarity degrees below the disjointness threshold are probably incompatible. If the similarity degrees are inside the range of the disjointness and overlappingness thresholds then the elements are probably disjoint. The area between the overlappingness and equivalence thresholds defines overlapping elements and the area above the equivalence threshold defines equivalent elements. Finally, the subsumption relationship is identified when one similarity degree is above the equivalence theshold and the other is below it.

4 Architecture

Existing data integration methodologies adopt either a hybrid or a composite approach to identify relationships between schema elements. In the hybrid approach, one module, using a single algorithm, combines several criteria, e.g. structure, names, constraints etc, to compare schema elements. We are going to adopt the composite approach in our architecture, where several different modules, that work independently and produce separate results, are combined to provide the final relationships between schema elements. Our composite architecture is illustrated in Figure 2.

Our approach differs from previous ones on the fact that there are two types of individual modules. There are modules that identify semantically related schema elements and modules that clarify the type of the semantic relationship, like in the Interschema Relationship Identification (IRI) approach, where first is the identification of related elements and then the classification of the relationships amongst these elements [13]. Using this kind of architecture and exploiting the bidirectional comparison as explained in the previous section, has the advantage of identifying the five types of semantic relationship between schema elements and directly integrating the input schemas as explained in [14].

Individual modules of both types perform a bidirectional comparison of the elements of the input schemas and produce the two similarity degrees, d(A,B) and d(B,A), for each pair of elements A, B in S_A and S_B respectively. Their results are combined in the `Aggregator` component, which can apply different aggregation algorithms. The `Degree Combinator` can refine existing similarity degrees or produce new degrees based on the results from the `Aggregator`. For example, if elements A,B and B,C are identified as similar in the form of similarity degrees d(A,B), d(B,C) then it can be deduced that elements A,C are also similar. If similarity degrees have not been previously produced for A,C then new degrees can be defined based on the degrees between A,B and B,C. If similarity degrees already exist then they can be refined based on the new degree. Optionally, the similarity degrees can be used as feedback for the individual modules, which can detect the incompatible elements with very low similarity degrees, remove the incompatible pairs and perform the same bidirectional comparison, producing more accurate results. The `Degree Combinator` outputs the final similarity degrees.

The final degrees are essential for the identification of the semantic relationships between schema elements. The `Relationship Extractor` component uses the final degrees with the graph in Figure 1 and the user-supplied disjointness, overlappingness and equivalence thresholds to produce proposed semantic relationships.

For each pair of elements that a semantic relationship has been identified by the `Relationship Extractor` other than incompatibility, the `brute-force` component produces its own semantic relationship as explained in Section 2. The relationships of the `brute-force` component are useful because they can be contradicted with the ones produced by the `Relationship Extractor`, thus providing an insight on the accuracy of the `Relationship Extractor`'s results. Based on this information, the user can validate the semantic relationships that the `Relationship Extractor` correctly identified and invalidate the

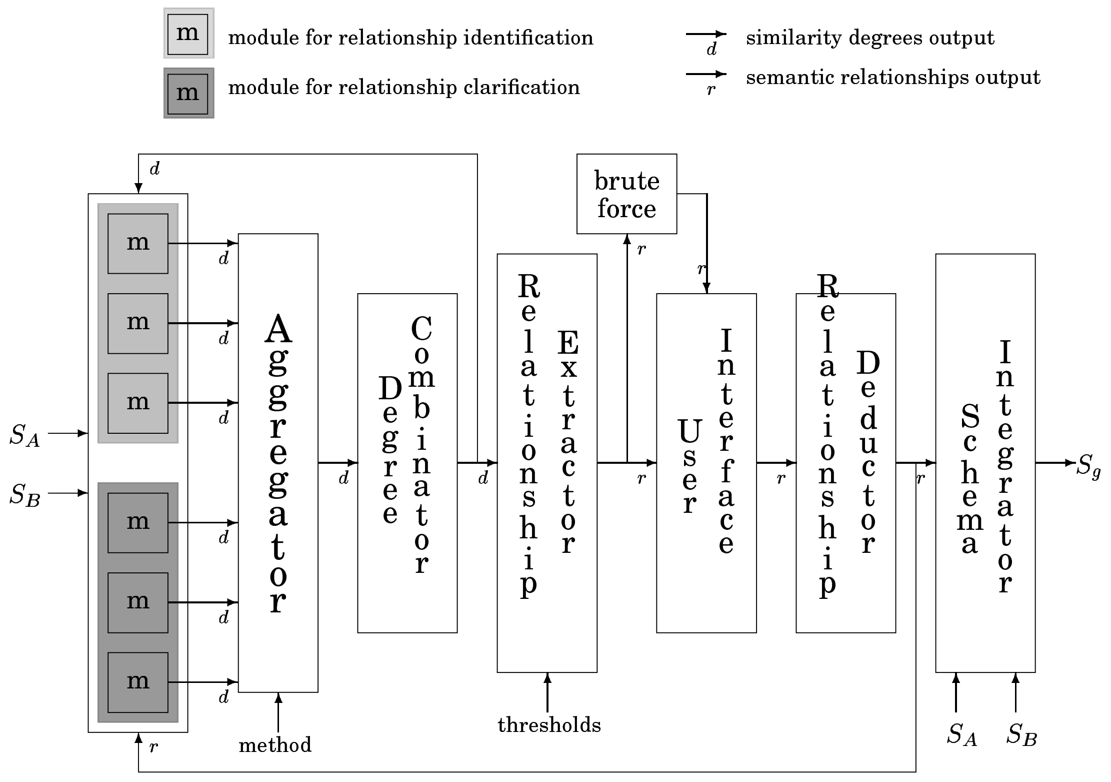


Figure 2: Architecture

false relationships.

The `Relationship Deductor` component can combine the user-validated relationships and deduce new logically-implied relationships that haven't been previously identified. For example, if the user has identified that $\mathsf{A} = \mathsf{B}$ and $\mathsf{C} \subset \mathsf{A}$ then logically it is implied that $\mathsf{C} \subset \mathsf{B}$. Finally, the user-validated relationships and the relationships deduced by the `Relationship Deductor` can be used by the `Schema Integrator` to perform the appropriate transformations and integrate the input schemas. Additionally, the final semantic relationships can be used as feedback by the individual modules that can learn from these examples to alter their behaviour and produce more accurate results.

The `Schema Integrator` based on the identified relationships checks the conditions defined in [14] and performs the corresponding transformations. First the schemas are conformed, then they are superimposed producing an intermediate schema, and then the intermediate schema is restructured to produce the final schema.

5 Prototype

5.1 Implemented Modules

The prototype tool consists of eight modules that attempt to find similarity relationships and clarify the type of each relationship, and the Aggregator, which combines the modules' results. In this section, we explain in detail each of these components and illustrate their results on two real-world databases that come from the Department of Computing in the Imperial College of London: CATE and DEPT.

CATE	vs	DEPT
exercise.type	=	xhelperstaff.roletype
attendancepmtppt.type	=	xhelperstaff.roletype
markspmtppt.type	=	xhelperstaff.roletype

Table 1: Equivalence relationships between attributes in CATE and DEPT

5.1.1 Linguistic Comparison Module

A very simple linguistic module has been implemented that performs a case insensitive comparison of element names. When element A has exactly the same name with B then $d(\text{A,B}) = d(\text{B,A}) = 1$ and if A's name is a substring of B's then $d(\text{A,B}) = 0.2$ and $d(\text{B,A}) = 1$, i.e. the similarity degrees are predefined.

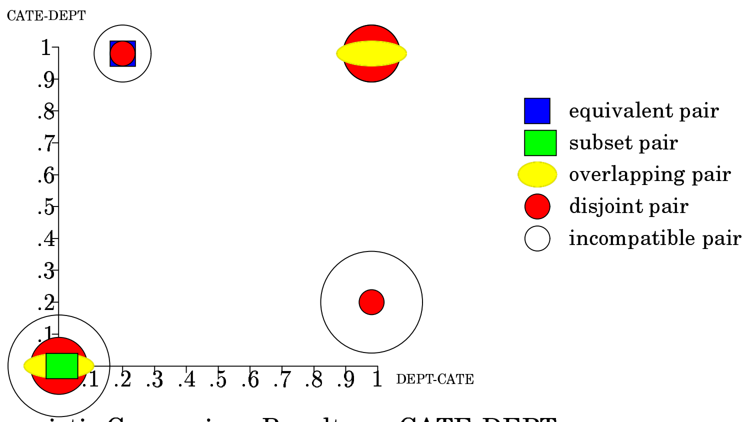


Figure 3: Linguistic Comparison Results on CATE-DEPT

Figure 3 illustrates the results of the linguistic comparison. Equivalent pair of elements are represented as squares, subset elements as rectangles, overlapping elements as ellipses, disjoint elements as filled circles and incompatible elements as empty circles. The size of each shape increases with the number of pairs of elements represented in each position of the graph. As it can be seen in the figure, the equivalent pairs of attributes type and roletype (Table 1) are not given the highest bidirectional similarity degrees (1.0, 1.0) because their names are substrings. Additionally, subset attributes are assigned the lowest degrees (0.0, 0.0). The module, also, incorrectly identifies disjoint, overlapping and incompatible attributes as equivalent (top-right corner) because they share the same names.

Also, there have been implemented four non-structural metadata-based modules that compare data types, length, statistics about numeric instances and statistics about the existence of special characters.

5.1.2 Data Type and Length Comparison Modules

The *data type* module compares element data types, using a 3-dimensional vector. The first two positions in the vector represent the character and numeric data types while the third position represents all other types. Each position's value is either 0 or 1 to represent true and false, respectively. For example, a character schema element which includes numerical instances (represented as strings) will have the first two positions in the vector equal to 1, i.e. true. The elements of each schema are clustered based on their data-type vector using the Self-Organizing Map [8] algorithm. The module can learn the clusters using a neural network, as done in Semint, and then classify the elements of one schema to the clusters of elements in the other schema. In our current implementation, similarity degrees between elements are computed by comparing the euclidean distances between each element and the clusters of the counter schema. Both *absolute* and *relative* similarity degrees can be produced. Absolute degrees are computed by comparing the distances between each element and the clusters against the maximum possible distance. However, if the distance between the clusters is relatively small then all absolute similarity degrees will be approximately equivalent. In this case, relative similarity degrees are useful. These are produced by comparing the euclidean distance between each element and the clusters against the maximum distance of the element from the clusters.

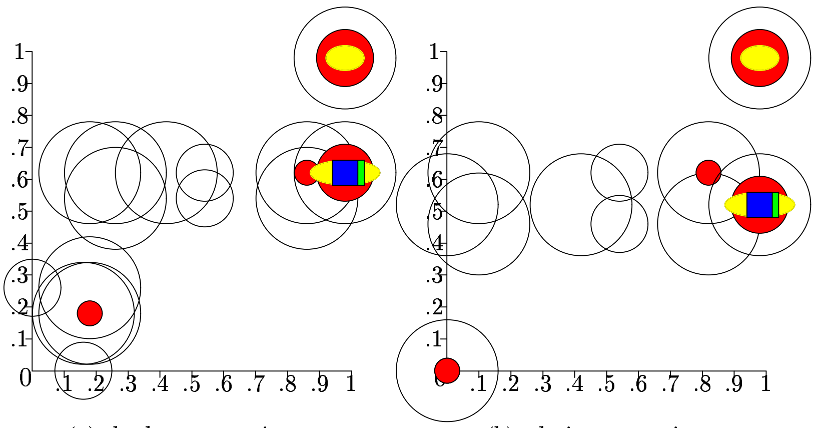


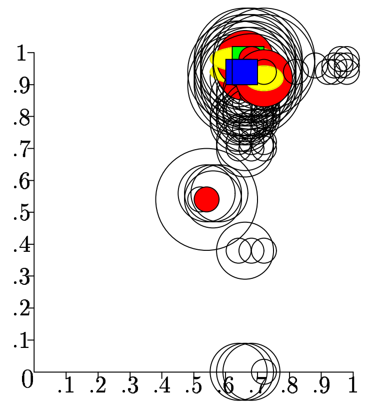
Figure 4: Data Type Comparison Results on CATE-DEPT

Figure 4 illustrates the results of the data type comparison using both absolute and relative similarity degrees. In this case, the results do not differ significantly, except that the absolute comparison assigns higher similarity degrees to all pairs of elements, both incompatible and similar pairs. The equivalent pairs, which consist of character attributes, are not assigned the maximum bidirectional similarity degrees because of incorrectly clustering character elements with numerical elements in the y-axis (DEPT-CATE comparison direction). Ideally, the implementation should create four clusters in each axis: one cluster for character elements, one for numerical elements, one cluster for character elements that contain numerical instances and a final cluster for all other types.

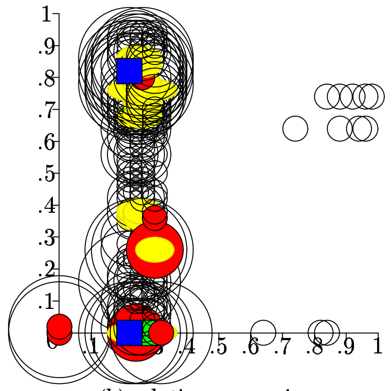
The *length* module uses the same vector-and-cluster technique to compare elements and produce bidirectional similarity degrees. In this case, each element is represented by a 5-dimensional vector. The positions in the vector correspond to the metadata length of each element, i.e. the length as provided by the DBMS, the maximum and minumum length of the element's instances, the average length of the instances and the average length of the distinct set of instances, i.e. when duplicates are removed, all of them compared against a maximum predefined length. The results of the length module, shown in Figure 5, are not very useful because similar pairs are not differentiated from incompatible pairs. In addition, there are incompatible pairs that have even higher similarity degrees than similar pairs.

5.1.3 Statistics-based Modules

The *numeric* metadata module is a hybrid module because it compares schema elements based both on their data type and on statistical information over their instances. Numerical elements are compared on their average value, the medium value and the standard devia-



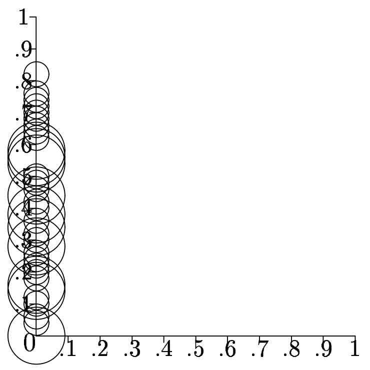
(a) absolute comparison



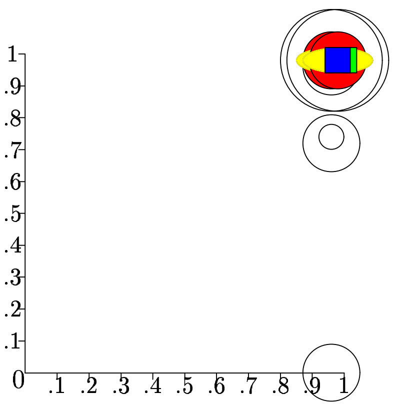
(b) relative comparison

Figure 5: Length Comparison Results on CATE-DEPT

tion of their instances. In the *character* metadata module, elements are compared on their average number of appearances of special characters, like @, $, etc. Both of the modules use the vector-and-cluster approach to produce bidirectional similarity degrees. As it can be seen in Figure 6 , there are not any similarity relationships between numerical elements. Their similarity degrees imply incompatibility. Some incompatible elements are assigned a higher similarity degree because they have similar average, medium, etc. values. In the character module, elements are assigned high similarity degrees because most of the special characters searched for do not appear in the elements. Therefore, incompatible and similar elements are not easily distinguishable.



(a) numeric module



(b) character module

Figure 6: Instances Statistic Comparison Results on CATE-DEPT

5.1.4 Instances Comparison Module

In the prototype tool, an instance comparison module has been implemented that uses the Naive Bayes technique to compare elements. The instances of each element are used as training data and the module learns each element using these instances. Similarity degrees are produced between each instance of the element and each learned element, i.e. an estimate of how likely is for each instance to belong to the learned element. Then, the average similarity degree of the instances is computed and this determines the overall similarity degree of the element and the learned element. In the reverse direction, the similarity degree is similarly computed.

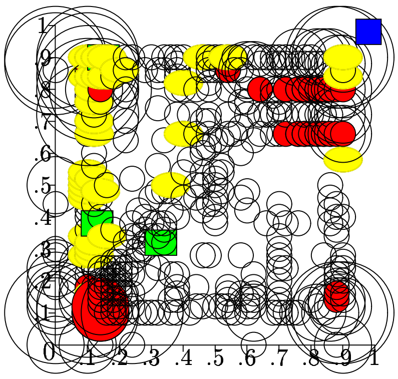


Figure 7: Instances Comparison Results on CATE-DEPT

5.1.5 Relationship Clarification Modules

Finally, two more modules have been implemented that rely on content-level metadata to perform the bidirectional comparison. These modules are more suitable for relationship clarification, because implicitly they assume that the elements being compared are similar elements. The *precision* module is hybrid because it differentiates between numerical elements and non-numerical elements. It examines the range of the element instances by comparing the maximum and minimum values for numerical elements and the maximum and minimum lengths for non-numerical elements. The bidirectional similarity degrees are computed based on the overlapping fraction of the elements' ranges and lengths. If the range of element A is within the range of element B, or the length of A is within the length of B, then B is considered to subsume A, producing similarity degree $d(A, B) = 1$. The reverse similarity degree, $d(B, A)$, is computed based on the overlapping fraction. If the ranges or lengths are distinct then the bidirectional similarity degrees are equal to zero.

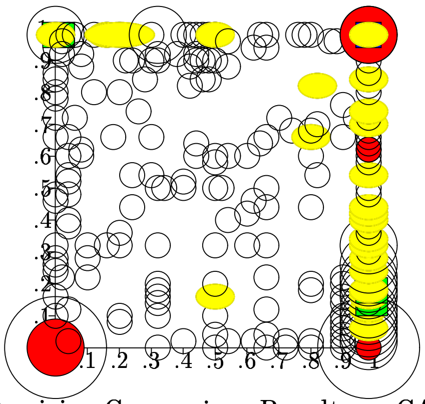


Figure 8: Precision Comparison Results on CATE-DEPT

The second relationship clarification is the *#instances* module, which is more naive. It compares the number of distinct instances of each element. If element A has i_A instances and element B has i_B instances with $i_A < i_B$, then A is considered to be subsumed by B. The bidirectional similarity degrees will be $d(A, B) = 1$ and $d(B, A) = i_A/i_B$ that show the percentage of the instances of B that is covered by A, i.e. the extent that B subsumes A.

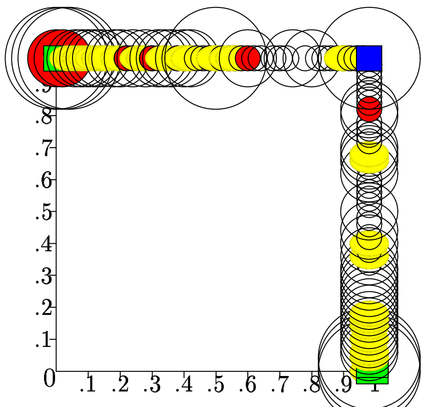
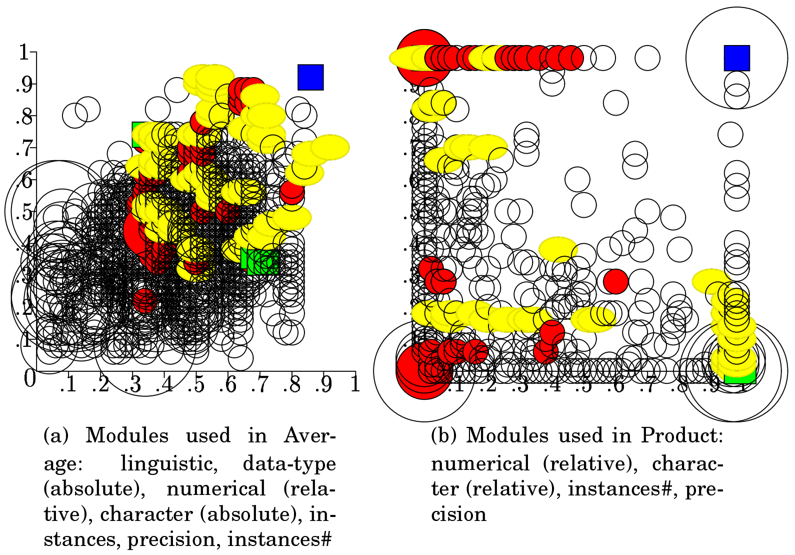


Figure 9: Number of Instances Comparison Results on CATE-DEPT

5.2 Aggregation

Different techniques can be adopted to combine the similarity degrees produced by the bidirectional comparison modules and compute the final bidirectional similarity degrees between each pair of elements. The average and weighted average strategies, adopted in COMA, can be used. In the latter approach, the weights are predefined by the user according to the importance of each module. More important modules have higher weights in order to have a greater impact on the final similarity degrees. Additionally, weight assignment is useful to minimize the effect of one type of information in the similarity degrees, when multiple modules comparing the same type of information exist. In LSD [5] weights are *learned* by the tool based on the ability of the modules to discover pre-defined relationships between elements.

In the prototype tool, the weighted average aggregation strategy has not been implemented because there are not multiple modules that exploit the same type of information and in order not to make any assumptions about the importance of the comparison modules. The average strategy has been implemented. Additionally, the product strategy is used, where the similarity degrees for each pair of elements and in each direction are multiplied to produce the final similarity degrees in that direction. Figure 10 illustrates the results of the two aggregation strategies.



(a) Modules used in Average: linguistic, data-type (absolute), numerical (relative), character (absolute), instances, precision, instances#

(b) Modules used in Product: numerical (relative), character (relative), instances#, precision

Figure 10: Average and Product aggregation strategies

The experiments show that the average and product aggregation strategies can be complementary. The average strategy is useful to distinguish similar pairs of elements from incompatible pairs. The incompatible pairs should be located in the bottom-left corner of the graphs, because their average bidirectional similarity degrees should be close to 0, and the similar pairs should be located in the top-right corner with bidirectional similarity degrees close to 1. Ideally, all the incompatible pairs of elements should be below a user-defined similarity threshold, and therefore, they could be discarded. The product strategy is then able to determine the type of the semantic relationship between the pairs of elements that have remained. Figure 11 shows the results of Figure 10 for the product strategy with all the incompatible pairs removed.

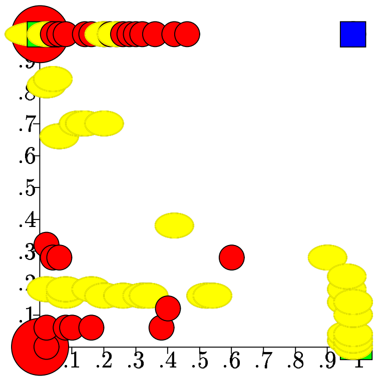


Figure 11: Product Strategy with incompatible pairs removed by the average strategy

Another idea for the aggregation of similarity degrees, which has not been considered in the literature previously, is the auxiliary role of particular modules. For example, the linguistic comparison modules can have an auxiliary role by only increasing and not decreasing the similarity degrees produced by other modules. In this way, they do not affect the similarity degrees of the elements whose names are not related according to the used dictionary/ontology, but only increase the similarity degrees of the elements who are linguistically similar. Depending on the overall aggregation strategy, average or product, the similarity degree produced by an auxiliary module can increase the current aggregated similarity degree according to the following formulae:

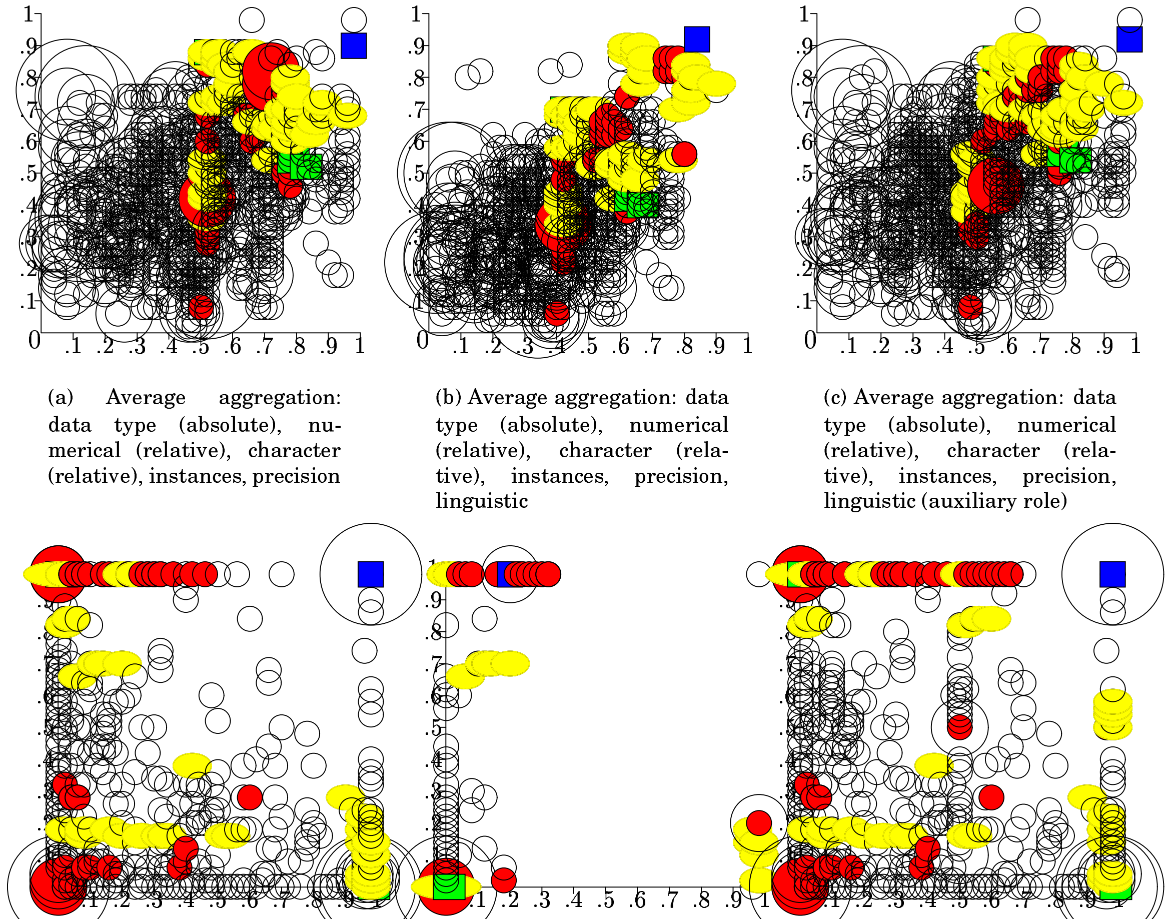
average aggregation:

$$auxSimDegree = \frac{currentSimAvg + (1 - currentSimAvg) * moduleSimDegree}{numOfModules}$$

product aggregation:

$$auxSimDegree = \frac{currentSimProduct + (currentSimProduct + (1 - currentSimProduct) * moduleSimDegree))}{2}$$

In both cases, the similarity degree produced by the aggregation of the rest of the modules (*currentSimAvg* or *currentSimProduct*) is increased by a percentage of the fraction ($1 - currentSimAvg$ or $1 - currentSimProduct$) remaining for the current aggregated similarity degree to become equal to 1. This percentage is based on the auxiliar module's similarity degree (*moduleSimDegree*). In the average aggregation strategy, the result of the above procedure is considered to be the new similarity degree for the auxiliar module and therefore it is treated like the similarity degrees of all the other modules, i.e. the sum of the similarity degrees is computed and then it is divided by the number of the modules participating in the sum. In the product aggregation strategy, the average of the current aggregated similarity degree and the auxiliar module's new similarity degree is computed. Figure 12 shows the results of the average and product aggregation strategies when the linguistic module is not included in the aggregation and when it is included either as an auxiliar module or as any other module.



(a) Average aggregation: data type (absolute), numerical (relative), character (relative), instances, precision

(b) Average aggregation: data type (absolute), numerical (relative), character (relative), instances, precision, linguistic

(c) Average aggregation: data type (absolute), numerical (relative), character (relative), instances, precision, linguistic (auxiliary role)

(d) Product aggregation: numerical (relative), character (absolute), instances#, precision

(e) Product aggregation: numerical (relative), character (absolute), instances#, precision, linguistic

(f) Product aggregation: numerical (relative), character (absolute), instances#, precision, linguistic (auxiliary role)

Figure 12: The auxiliary role of the linguistic module

The figure shows that using the linguistic module in an auxiliary role is not appropriate in the average aggregation strategy, because the degrees of incompatible but linguistically similar elements are also increased, making the distinction of incompatible and similar elements more difficult. On the other hand, in the product aggregation strategy, the linguistic module is more useful as an auxiliary module, because it makes the similarity relationships clearer [2].

For auxiliar modules, more formulae can be used to increase the aggregated results. For example, in the product aggregation strategy the new similarity degree computed for the auxiliar module can be used as the new aggregated similarity degree without computing the average with the current aggregated degree.

[2]It is assumed that the incompatible pairs of elements have already been identified during the average aggregation and removed for the product aggregation

6 Concluding Remarks

In this report, we have presented our composite approach to automatically discover semantic relationships between schema elements. Based on a bidirectional comparison of the elements metadata and instances, we are able to discover equivalence, subsumption, overlappingness, disjointness and incompatibility relationships. We have shown the architecture and described the components that we have implemented in the prototype tool.

In our future work, we are going to focus in the implementation of all the mandatory components of the architecture, which are the Relationship Extractor, the user-interface and the Schema Integrator (all have been described in Section 4), making the tool fully capable of integrating real data sources.

We are also going to examine ways of improving the current bidirectional comparison modules or building new ones. The *length* module, described in Section 5.1, must be improved because the length information is very important in detecting compatibility relationships and it is not exploited by our current implementation. A technique similar to the one used in the *precision* module can be adopted, or a different clustering algorithm can be implemented instead of SOM, e.g. the k-means algorithm [7]. Additionally, a decision tree or a neural network can be used to classify elements, instead of examining euclidean distances. These modification can also be useful in the other modules that use the vector-and-clustering technique, e.g. the *type* module which does not cluster elements as expected.

A more intelligent *name* module should also be implemented using external knowledge from dictionaries and ontologies. Additionally, different machine learning classification techniques can be employed and tested for instance comparison modules and element classification, e.g. the k-nearest neighbour technique [6]. Structural comparison modules can also be built in order to exploit all the available information from the data sources. Finally, concerning techniques for similarity degree aggregation, the weighted average strategy (Section 5.2) can be tested as well as different formulae for auxiliary modules, to determine the most accurate and useful approach.

References

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

[2] Domenico Beneventano, Sonia Bergamaschi, Silvana Castano, Alberto Corni, R. Guidetti, G. Malvezzi, Michele Melchiori, and Maurizio Vincini. Information integration: The MOMIS project demonstration. In *The VLDB Journal*, pages 611–614, 2000.

[3] Jacob Berlin and Amihai Motro. Database schema matching using machine learning with feature selection. In *Advanced Information Systems Engineering, 14th International Conference CAiSE'02*, pages 452–466, 2002.

[4] H. Do and E. Rahm. Coma - a system for flexible combination of schema matching approaches. In *Proc. 28th VLDB Conference*, pages 610–621, 2002.

[5] AnHai Doan, Pedro Domingos, and Alon Y. Levy. Learning source description for data integration. In *WebDB (Informal Proceedings)*, pages 81–86, 2000.

[6] K. Fukunaga and P. M. Narendra. A branch and bound algorithms for computing k-nearest neighbors. *IEEE Computer*, 24(7):750–753, 1975.

[7] J. Hartigan and M. Wong. A k-means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979.

[8] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. Som pak: The self-organizing map program package, 1996.

[9] Wen-Syan Li and Chris Clifton. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data and Knowledge Engineering*, 33:49–84, 2000.

[10] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proc. 27th VLDB Conference*, pages 49–58, 2001.

[11] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources, 1999.

[12] Erhard Rahm and Philip A. Bernstein. On matching schemas automatically. Report, 2001.

[13] S. Ram and V. Ramesh. *Schema integration: Past, present, and future.* Morgan-Kaufmann, San Mateo, CA, 1998.

[14] N. Rizopoulos. BAV transformations on relational schemas based on semantic relationships between attributes. Technical report, AutoMed Project, 2003.

[15] H. Garcia-Molina S. Mehiik and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of ICDE'02*, 2002.

[16] Giorgio Terracina. Uniform management of heterogeneous semi-structured information sources, 1999.