

OntoMatch: A Monotonically Improving Schema Matching System for Autonomous Data Integration*

Anupam Bhattacharjee Hasan Jamil

Department of Computer Science, Wayne State University, USA

anupam@wayne.edu, jamil@cs.wayne.edu

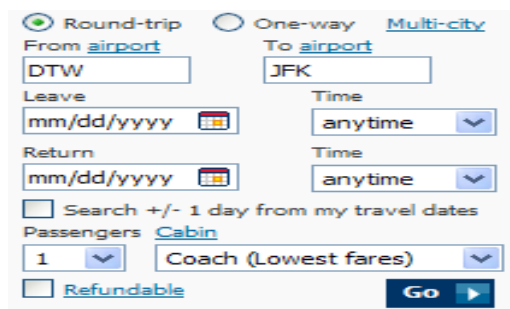
Abstract

Traditional schema matchers use a set of distinct simple matchers and use a composition function to combine the individual scores using an arbitrary order of matcher application leading to non-intuitive scores, produce improper matches, and wasteful and counterproductive computation, especially when no consideration is given to the properties of the individual matchers and the context of the application. In this paper, we propose a new method for schema matching in which wasteful computation is avoided by a prudent, and objective selection and ordering of a subset of useful matchers. This method thus has the potential to improve the matching efficiency and accuracy of many popular ontology generation engines. Such efficiency and quality assurance are imperative in autonomous systems because users rarely have a chance to validate the processing accuracy until the computation is complete. Experimental result to support the claim that such an approach monotonically improves the matching score at successive application of the matchers is also provided.

1 Introduction

Increasing interest in autonomous data integration makes it imperative that we develop accurate and scalable schema matchers for large scale applications. Schema matchers that rely on human intervention or user guidance do not perform too badly in small scale applications where the number of schemes are low. But applications that require matching a large and different sets of schemes per user query, cannot have the luxury of end user guidance. Application in ad hoc or on-the-fly data integration require such approach to schema matching. One such practical example is the recently proposed language BioFlow [15] in which the user view of the database is different from the real world view

on the internet and the expectation is that the system will be able to somehow map the user view into the real world accurately. In this environment users are allowed to ask any query using BioFlow’s SQL-like syntax using any web site in an ad hoc manner and as such a fully autonomous schema matching is not only prudent, its a necessity. To illustrate the issue on a more intuitive ground, let us consider the following example shown in figures 1(a) and 1(b) extracted from two popular flight search web sites.



(a) www.delta.com



(b) www.united.com

*Research supported in part by National Science Foundation grants CNS 0521454 and IIS 0612203.

Figure 1. Flight Search Websites

To find a good fare from Detroit to New York, a user must search both sites manually and integrate the two schemes in his mind where he reasons that the terms Leave and Departing are synonymous. Systems such as Expedia.com or Kayak.com actually resolved such schema heterogeneity among a set of predetermined sites including Delta and United to operate efficiently. In this approach, the autonomy of both systems, say Expedia and Delta, are lost as they cannot change without communicating the intention to change to the other side or risk failure. The downside is that, such systems and approach cannot accommodate composition of an arbitrary set of services or web sites to respond to arbitrary end user queries. On-the-fly or ad hoc integration systems require this ability to support a platform where no such prefabricated schema matching will be needed. But supporting such a flexible environment is by no means easy mostly because current schema matchers are not equipped to handle such visions. For example, instance-based matchers often rely on prior training and thus, fail to support automatic schema mapping on-the-fly where an abundance of prior knowledge about the domain of the schemas are not available [17, 21, 14, 12, 11].

Schema matchers usually employ a set of simple matching algorithms such as synonym matching, abbreviation matching, edit distance matching, and so on, and a score composition function to combine the individual matching scores. In most contemporary matchers it is assumed that all matching algorithms are independent and the final score can be computed by statically assigning a weight to each algorithm to reflect its contribution such as in [6]. The drawback of this approach is that the static weighting may not be appropriate for an application in hand, and the order of the application of the matching algorithm does not guarantee an overall improvement in the matching process as shown in figure 3. In particular, such composition functions assume that the weight assigned is absolute even if it brings down the overall score by failing to see that application of a matching algorithm may be incorrect in the first place. For example, let us consider the two terms Leave and Departing we have discussed earlier. For these two terms, $editDistance(Leave, Departing) \leq synonym(Leave, Departing)$ will always hold. But systems such as [20, 2, 3, 9] apply matching algorithms without any implicit order, and do not take the position that application of edit distance matching is not warranted at all when a synonym matcher succeeds. Not only that it is inappropriate, we need not spend the time to blindly compute edit distance only to ignore it at the end by adjusting the weight to zero and thereby incur wasteful computational cost without necessarily leading to any improvement in the matching quality [10].

Systems such as [1, 16, 13] on the other hand employ an ordering relation among the matching algorithms but they do so statically. Matching that survived the previous match

threshold move on to the next algorithm for matching. Going back to our example, the ordering relationship $editDistance \preceq synonym^1$ will not produce any match for $synonym(editDistance(Leave, Departing))$ if the match threshold is set too high to maintain accuracy, whereas computing $editDistance(synonym(Leave, Departing))$ based on the reverse ordering $synonym \preceq editDistance$ is likely to produce the correct match. Clearly, automatic ordering of the individual matchers for online matching remains as a huge challenge as observed in research such as [4, 5, 8, 7]. Our goal is to avoid the drawbacks of both approaches discussed above and devise a fully dynamic matcher selection function that always employs the right set of matchers in an intuitive order based on an objective function. We believe that this approach will eliminate wasteful computation, erratic matching behavior and improve match quality monotonically. Based on these two principles, we developed a schema matching system, called the *OntoMatch*, by developing methods to characterize the matching algorithms and using those characterizations to design an objective function for the selection and ordering of the simple matchers.

2 Components of OntoMatch

Before we describe OntoMatch, we formally define a few concepts we use later in this paper.

Ontology/Scheme An ontology O (or equivalently a scheme) is defined as a pair (T, R) where T is a non-empty set of terms $\{t_1, t_2, \dots, t_m\}$, and $R \subseteq T \times T \times \mathcal{X}$ is a possibly empty set of relationships of the form $\{(t_i, t_j, c) | i \neq j \text{ and } i, j \leq m\}$ among the terms $t_i, t_j \in T$, and $\mathcal{X} = \{one-to-one, one-to-many, many-to-one, many-to-many\}$.

For example, two ontologies, or schemes, O_1 and O_2 can be described as follows:

$$O_1 = (\{\text{student, major, student no.}\}, \{(\text{student, major, many-to-one}), (\text{student, student no., one-to-one}), (\text{major, student no., many-to-many})\})$$

$$O_2 = (\{\text{std, subject, studentID}\}, \{(\text{std, subject, many-to-one}), (\text{std, studentID, one-to-one}), (\text{subject, studentID, many-to-many})\})$$

Term Similarity Let t_1 and t_2 be two terms in T , and ψ be any similarity function². Any two terms t_1, t_2 are said to be similar if $\psi(t_1, t_2) < \tau$ for a given threshold τ , where $\psi(t_1, t_2) \in [0, 1]$, and $\psi(t_1, t_2) = 0$ and $\psi(t_1, t_2) = 1$ respectively mean complete dissimilarity and identity.

We are now ready to formally define schema mapping in terms of term similarity described above.

¹Here $editDistance \preceq synonym$ means edit distance matching must be employed before synonym matching.

²Such as string edit distance, thesaurus at WordNet [19], etc., or a combination of such functions.

Schema Mapping Let $O = \bigcup_i O_i = (T_i, R_i)$ be the set of all ontologies, Ω be a family of matching functions μ , τ be a similarity threshold, and \mathcal{R} be the set of all real numbers between τ and 1. Then schema mapping between two ontologies O_1 and O_2 is defined as a function $\Psi : O \times O \rightsquigarrow [\Theta : T \times T \mapsto [\Phi : 2^\Omega \rightsquigarrow \mathcal{R}]]$, where $[A \rightsquigarrow B]$ means a set of partial functions and $[A \mapsto B]$ means a set of total functions. In other words, for any two ontologies O_1 and O_2 in O , and two terms t_1 and t_2 in T , $\Psi(O_1, O_2)\Theta(t_1, t_2)\Phi(\{\mu_1, \dots, \mu_k\}) = r$ such that $r \in \mathcal{R}$. Furthermore, we require that $r \geq \tau$ chosen to be the highest possible value for a given pair of terms (t_1, t_2) , and that Θ only choose unique pairs. This means that once a pair of terms is matched with the highest possible similarity, they both will be excluded from other attempts. Our goal in this paper is to propose a set of functions μ_i , a function Θ and the overall matching function Ψ . Notice that Ψ and Φ are partial functions and are not required to map all elements, but once Ψ makes a choice, Θ is required to find a set of matching functions μ to match the terms. As an example, the moment (student, std) is selected as the highest match pair, matching (student, subject) is no longer possible.

As we discussed in section 1, it is not possible for any matcher to have correct mappings all the time. Our approach, in OntoMatch, is to leverage the strengths of existing matchers in a way that allows for higher quality mapping. In this paper, we identify five conventional matchers and show how these matchers can be characterized and their characteristics used to induce an ordering among them based on the target term pairs. The five simple matchers we consider in this paper are described briefly below:

Edit Distance Matcher (EDM): Schema, table, and attribute names carry information about their semantic meanings and thus play a vital role in ontology mappings. Most matchers start with this heuristic as a first step. As an example, the terms "EmployeeName" and "Empl.Name" may be treated as semantically synonymous as the edit distance between them is significantly lower. In this paper, we adopt the traditional definition of edit distance where it is the number of corrections needed to make two terms look identical. The EDM of OntoMatch uses case normalization, diacritic suppression, blank normalization, link stripping, digit suppression, punctuation elimination etc. Formally, for two terms t_1, t_2 , $\delta(t_1, t_2) = [0, 1]$ and the similarity $\psi(t_1, t_2)$ is given by $1 - \delta(t_1, t_2)$.

Abbreviation Matcher (AM): The Abbreviation matcher equates two terms by abbreviating them. Abbreviations are often used to represent items in real life as a shorthand. Given two terms t_1, t_2 , AM returns 1 if $t_1 = abbreviation(t_2)$ or $t_2 = abbreviation(t_1)$ or $abbreviation(t_1) = abbreviation(t_2)$, otherwise it returns 0.

Synonym Matcher (SynM): Given two terms t_1, t_2 , SynM returns 1 if the terms are synonymous to each other, otherwise it returns 0.

Constraint Matcher (CM): Given two terms t_1, t_2 , CM checks whether the terms are related to each other by some constraints. For example, both terms may be primary keys or they may have identical or compatible types. Compatible types are (integer, number), (real, float), (string, varchar) and so on. We define a simple distance matrix for CM in the following way:

$$dist(t_1, t_2) = \begin{cases} 1 & \text{if both terms are primary keys} \\ & \text{or both have same types} \\ 0.5 & \text{if one is primary key and the} \\ & \text{is foreign key or vice versa} \\ & \text{or both have compatible types} \\ & \text{or } t_2 \text{ is a foreign key or vice versa} \\ 0.25 & \text{if both are foreign keys} \\ 0 & \text{otherwise} \end{cases}$$

Structure Matcher (SM): For nested structures, SM measures the similarity between the subtrees rooted at the nodes corresponding to the terms. The more the two subtrees look alike, the nearer the two terms are. The similarity of two subtrees are defined by a combination of the cardinality relationships, type homogeneity, and other constraints, and computed recursively.

3 Characterization of the Term Matchers

The goal here is to determine, given two terms t_1 and t_2 , how likely it is that a given matcher μ is suitable for match evaluation. In particular, the question is how do we determine the goodness of a candidate matcher μ for terms t_1 and t_2 when a set of matchers $\{\mu_1, \dots, \mu_k\}$ has already been chosen, and if it is appropriate, in which order it should be applied along with the selected set of matchers to produce a monotonically improving match?

In EDM, an 'edit' operation can be performed by either inserting/deleting a character or replacing the character with another character. For example, we can transform the term 'flt' into 'flight' by inserting three characters 'igh'. Obviously there are several ways to transform a term into another term. EDM accomplishes this by a set of edit operations, such that the total cost of editing is minimum. A sample cost matrix is: -1 for each match, +1 for a replace operation, and +2 for any insert/delete operation. So, the edit distance between 'flt' and 'flight' is 6-3=3. We divide the distance by the maximum of the lengths of the terms to normalize it into [0, 1] range. Now, as we can see, the greater the length difference between the two terms, the higher the cost to convert the terms, and so, the less similar they are.

The edit distance between two longer terms and between two shorter terms could be the same (e.g., ‘student’ vs. ‘studnt’ and ‘co’ vs. ‘com’). Two non equivalent smaller terms usually have a very low edit distance and thus, have higher chance of being converted into the other. This happens frequently when both the terms are expressed in an abbreviated form. On the other hand, two semantically equivalent lengthy terms usually have a low edit distance. So, if we assume a fixed threshold for the cases not using the EDM, the chances of false positives increases. In other words, the suitability of EDM depends on at least two properties: the length of the terms, and the length difference between the terms. The question then arises, what should be the best way identify when a term is lengthy. It turns that it is a relative matter and depends upon the schemas at hand. We define a term to be lengthy if its length is longer than the average length of the terms in associated scheme. The same rule applies for SynM matchers.

- p_1 : if $l_1 \geq avg_l$ $s_1 = 1$ else $s_1 = 0$
- p_2 : if $l_2 \geq avg_l$ $s_2 = 1$ else $s_2 = 0$
- p_3 : $s_3 = |l_1 - l_2| / \max(l_1, l_2)$
- p_4 : $s_4 = 1 - s_3$
- p_5 : $s_5 = 1 - (|card_1 - card_2| / \max(card_1, card_2))$
- p_6 : $s_6 = dist(t_1, t_2)$ from constraint matcher

Table 1. Properties used to select matchers

Since abbreviations make terms shorter, we can make an educated guess that if both terms are lengthy, the chance that one of them is an abbreviation of the other is least likely, and consequently least chance of matching occurs when they are both short. The best result is obtained when one of the terms is lengthy and the other shorter. In the same way, structure matcher considers the cardinality, type homogeneity, and keys constraints of the terms. So, when two terms have different cardinalities, key constraints and incompatible types, SM is preferentially deselected. Again we often use higher threshold value for the SM. Thus, even if two terms have the same key constraint, SM may not be selected. In such cases, CM tries to match the terms. The intuition is that, the matching quality of the two terms is high in case that they match as well as both of them are primary or foreign keys of their own ontologies. These observations are the basis for our six properties we list in the table 1 using which we develop an objective function to select or deselect a matcher in our algorithm we present in the next section.

4 The OntoMatch Algorithm

The overall ontology mapping process can be divided into three distinct steps:

1. Select matchers with highest suitability scores,

2. Compute term similarity using the selected matchers, and
3. Apply stable marriage algorithm to obtain a final mapping based on the matching scores, threshold, and the distance matrix.

For a pair of ontologies, we take each of the terms from the first ontology and match it against every term in the other ontology. Thus, after running the algorithm we get a term \times term distance matrix. Now, given a set of matchers $\Omega = \{\mu_1, \mu_2, \dots, \mu_n\}$, for every pair of terms (t_1, t_2) OntoMatch first selects $\Omega' \subseteq \Omega$ and then applies matchers successively from Ω' to find a match. A matcher μ is selected based on a subset of properties from $P = \{p_1, p_2, \dots, p_k\}$, where k is the number of properties. Some of the properties depend on term lengths while the rest depends on the distance functions as used in constraint matcher. Table 1 shows the set of properties P , where l_1, l_2 are lengths of t_1 and t_2 ; avg_l is the average length of all terms, and $card_1, card_2$ are cardinalities of two schemas. We use a different set of properties to select different matchers. For EDM and SynM we use the property set $P_{EDM} = P_{SynM} = \{p_1, p_2, p_4\}$, for AM we use $P_{AM} = \{p_3\}$, for SM we use $P_{SM} = \{p_5, p_6\}$ and for CM we use $P_{CM} = \{p_6\}$.

If we consider P_{EDM} it says that, if both terms are longer, EDM will get higher score to be selected; if one term is longer and the other is shorter the score will be less but probably high enough to apply EDM; and if both terms are short the score will be lower not to consider EDM. Similarly for the abbreviation matcher the length difference plays a vital role in preferentially selecting it. In the same way, if the terms are keys of the schemas, SM gets higher priority. We use GORDIAN [18] to automatically identify the keys of the schemas if they are not given in the schema definition.

Algorithm 1 Algorithm to select the best applicable matcher

Require: \mathcal{M} , where $\mathcal{M} \subseteq M$, $bestMatcher = \emptyset$, $bestScore = 0$

Returns: The most suitable matcher

- 1: **for** each matcher $m \in \mathcal{M}$ **do**
 - 2: $score = 0$;
 - 3: **for** each property $p_i \in P_m$ **do**
 - 4: $score = score + s_i$;
 - 5: **end for**
 - 6: **if** $score \geq \tau_m$ and $score > bestScore$ **then**
 - 7: $bestMatcher = m$, $bestScore = score$;
 - 8: **end if**
 - 9: **end for**
 - 10: return $bestMatcher$;
-

Once the best applicable matcher has been identified, the next step is to apply it to the terms to possibly get a new pair of transformed terms. For example, the abbreviation matcher transforms a term to its abbreviations, e.g. (‘mobilenumber’, ‘cellnumber’) becomes (‘mobnum’,

‘cellnum’). Similarly, the synonym matcher transforms a term to its synonyms, e.g., (‘mobilenum’, ‘cellnumber’) becomes (‘cellnum’, ‘cellnumber’). EDM, SM, and CM have no transformations on the terms. We continue in this way unless we get a distance which is less than the threshold for overall matching. Thus, we get a two dimensional matrix $|O_1 \times O_2|$ denoting the match distances between the terms of the two ontologies. Algorithm 2 finds the distance between the terms t_1, t_2 .

Algorithm 2 Algorithm to calculate the distance matrix

Require: a pair of terms (t_1, t_2) , and a set of simple matchers M

- 1: $score = 0$;
- 2: matcher $m =$ get best matchers from M by algorithm 1;
- 3: terminate the algorithm and return $score$ if infinite loop occurs;
- 4: get distance $dist$ by applying matcher m to t_1, t_2 ;
- 5: $score = (score + 1 - dist)/2$;
- 6: **if** $score \leq \tau$ **then**
- 7: return $score$;
- 8: **end if**
- 9: **if** $m = \text{SynM}$ or $m = \text{AM}$ **then**
- 10: transform (t_1, t_2) to (t'_1, t'_2) by applying m ;
- 11: **else**
- 12: $(t'_1, t'_2) = (t_1, t_2)$
- 13: **end if**
- 14: call this algorithm with t'_1, t'_2, M ;
- 15: return $score$;

One we have such a metric at hand, the next task is to select a list of matching as the final output of the OntoMatch algorithm. Of several available greedy approaches for such a selection from a distance matrix, we choose the *stable marriage* algorithm. Given an $|m \times n|$ distance matrix ($m \leq n$) the algorithm can select m matched elements out of n elements such that the least distant pair of elements is selected first and removed from the matrix. The correspondence with distance $\leq \tau$ is then filtered out from m correspondences. These are basically the final output of the OntoMatch algorithm.

5 Experimental Results

We demonstrate the effectiveness and strengths of OntoMatch in three ways. First, OntoMatch performs better on the synthetic data than other popular matchers. Second, experimental results indicating that OntoMatch provides the best ordering of the subset of the simple matchers among all possible orderings and subset selections of the matchers. Lastly, we show that OntoMatch improves the quality of matching monotonically while other matchers behave unpredictably and often significantly poorly than OntoMatch.

We evaluate the performance of the matcher against the synthetic data available in the UIUC web integration repos-

itory. Of several data sets available to download from the repository, we chose the BAMB extracted query schemas and Tel-8 schemas. The BAMB section contains four types of schemas: automobile, books, movies, and music. From each category, we selected a schema and compared it against all other schemas of the same category. From the available existing composite matchers, we selected CUPID and COMA. From the available simple matchers of COMA library, we selected the same set of simple matchers OntoMatch uses. We use the area under precision recall curve (AUPRC) as a performance measure. As can be seen from Figure 2, OntoMatch shows greater precision and recall than CUPID and COMA.

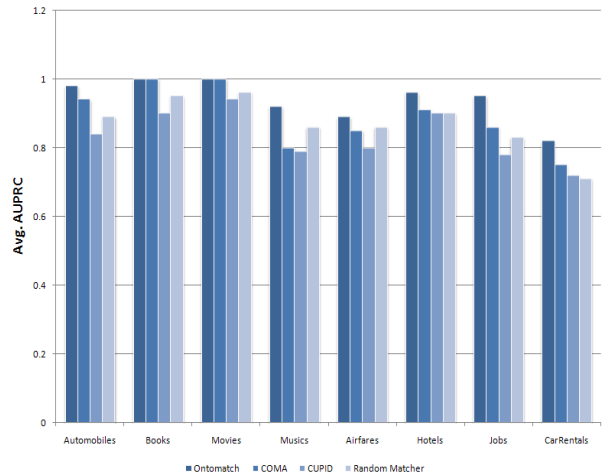


Figure 2. Precision Recall Curve

To establish our second claim, we compare the ordering and matcher selection of OntoMatch against all possible orderings and subset selections of the five simple matchers. There are 324 such possible compositions (${}^5P_1 + {}^5P_2 + {}^5P_3 + {}^5P_4 + {}^5P_5 - 1$). We consider all the combinations on the synthetic data, select the best scorer composition for each matching (we call it *random matcher*), and finally compare it with OntoMatch using AUPRC measure. Figure 2 shows the performance of OntoMatch over the random matcher. As expected, the ordering provided by OntoMatch is clearly better than all other possible combinations.

Finally, to show the high quality matching performance of OntoMatch, we compare the matching scores generated by the four matchers after successive application of the simple matchers. For the sake of simplicity and brevity, we only show the result of the music domain. In this category, OntoMatch performs substantially better than the other matchers. Figure 3 shows the average score of the total matches after every successive application of the simple matchers. After applying every simple matcher, the scores vary unpredictably in the case of the three other matchers, whereas

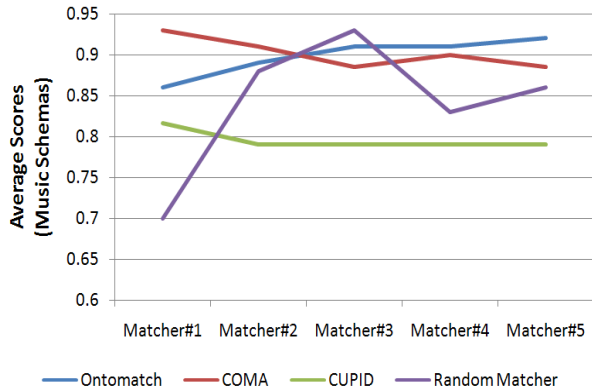


Figure 3. Match Quality Comparison

in OntoMatch, it is always monotonic improvement of precision and recall.

6 Summary and Future Research

In this paper, we presented a new approach toward the ordering of a set of matchers for higher quality automatic ontology matching. Our approach was based on the claim that a fixed ordering of matchers lead to poor match quality. The approach is not instance or application dependent and does not require any prior training making it suitable for ad hoc data integration applications. We have also demonstrated that an objective function can be used to dynamically select a set of matchers and establish their match ordering to guarantee monotonic improvement of the match quality. Thus OntoMatch avoids wasteful computation and improves overall efficiency as shown in the experimental results. Our results also indicate that our approach was better than the leading contemporary matchers. Moreover, the add-on property of this technique makes it flexible and extensible to use any number of matchers for better matching prospects. We view our present work as a first step towards the development of a robust dynamic schema matching system. Hence our plan for future work includes the formulation of a better objective function and development of a theoretical foundation for matcher selection that does not depend upon a static enumeration of the matcher property. We also plan to develop a system where the actual properties of matchers can be used to judge their suitability.

References

- [1] A. Anaby-Tavor, A. Gal, and A. Trombetta. Evaluating matching algorithms: the monotonicity principle. *IJCAI Workshop on Information Integration on the Web*, pages 47–52, 2003.
- [2] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. *SIGMOD Conference*, pages 906–908, 2005.
- [3] M. Benerecetti, P. Bouquet, and S. Zanobini. Soundness of schema matching methods. *Second European Semantic Web Conference*, pages 211–225, 2005.
- [4] P. A. Bernstein, S. Melnik, and J. E. Churchill. Incremental schema matching. *VLDB*, pages 1167–1170, 2006.
- [5] P. Bohannon, E. Elnahrawy, W. Fan, and M. Flaster. Putting context into schema matching. *VLDB*, September.
- [6] H. Do and E. Rahm. Coma - a system for flexible combination of schema matching approaches. *VLDB*, 2002.
- [7] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. *SIGMOD Conference*, 2001.
- [8] A. Doan, P. Domingos, and A. Y. Levy. Learning source description for data integration. *WebDB*, pages 81–86, 2000.
- [9] D. Engmann and S. Maßmann. Instance matching with coma++. pages 28–37, 2007.
- [10] A. Gal. On the cardinality of schema matching. In *OTM Workshops*, pages 947–956, 2005.
- [11] A. Gal. Why is schema matching tough and what can we do about it? *SIGMOD Record*, 35(4):2–5, 2006.
- [12] A. Gal, G. A. Modica, H. M. Jamil, and A. Eyal. Automatic ontology matching using application semantics. *AI Magazine*, 26(1):21–32, 2005.
- [13] G. Gal A., Modica and H. Jamil. Improving web search with automatic ontology matching. Available at <http://citeseer.ist.psu.edu/558376.html>, 2003.
- [14] A. Heß. An iterative algorithm for ontology mapping capable of using training data. *European Semantic Web Conference*, pages 19–33, June 2006.
- [15] H. Jamil and B. El-Hajj-Diab. BioFlow: A web-based declarative workflow language for Life Sciences. In *2nd IEEE Workshop on Scientific Workflows*, pages 453–460. IEEE Computer Society, 2008.
- [16] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. *VLDB*, pages 49–58, 2001.
- [17] N. F. Noy. Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, 33(4):65–70, December 2004.
- [18] Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald. Gordian: efficient and scalable discovery of composite keys. In *VLDB*, pages 691–702, 2006.
- [19] M. M. Stark and R. F. Riesenfeld. Wordnet: An electronic lexical database. In *Eurographics Workshop on Rendering*, 1998.
- [20] A. Thor, T. Kirsten, and E. Rahm. Instance-based matching of hierarchical ontologies. *BTW*, 103:436–448, 2007.
- [21] F. van Harmelen. Ontology mapping: A way out of the medical tower of babel? *10th Conference on Artificial Intelligence in Medicine*, pages 3–6, July 2005.