Tuning for Schema Matching

Zohra Bellahsene and Fabien Duchateau

Abstract Schema matching has long been heading towards complete automation. However, the difficulty arising from heterogeneity in the data sources, domain specificity or structure complexity has led to a plethora of semi-automatic matching tools. Besides, letting users the possibility to tune a tool also provides more flexibility, for instance to increase the matching quality. In the recent years, much work has been done to support users in the tuning process, specifically at higher levels. Indeed, tuning occurs at every step of the matching process. At the lowest level, similarity measures include internal parameters which directly impact computed similarity values. Furthermore, a common filter to present mappings to users are the thresholds applied to these values. At a mid-level, users can adopt one or more strategies according to the matching tool that they use. These strategies aim at combining similarity measures in an efficient way. Several tools support the users in this task, mainly by providing state-of-the-art graphical user interfaces. Automatically tuning a matching tool at this level is also possible, but this is limited to a few matching tools. The highest level deals with the choice of the matching tool. Due to the proliferation of these approaches, the first issue for the user is to find the one which would best satisfies her criteria. Although benchmarking available matching tools with datasets can be useful, we show that several approaches have been recently designed to solve this problem.

Zohra Bellahsene
University of Montpellier II, France e-mail: bella@lirmm.fr

Fabien Duchateau
CWI, Amsterdam, The Netherlands e-mail: fabien@cwi.nl

1 Introduction

The gap between manual schema matching and semi-automatic schema matching has been filled in early, especially because of the need to handle large schemas and to accelerate the matching process (Carmel et al, 2007). The next step towards automatic schema matching is mainly motivated by the lack of human experts, for instance in dynamic environments. In all cases, tuning is mainly required to improve quality results and/or time performance. We illustrate this statement with Figure 1, on which four schema matchers (YAM, COMA, Similarity Flooding and YAM with tuning) have been run on the same scenario. Only one of them has been tuned and this plot compares the number of user interactions to manually obtain a 100% F-measure. In brief, a user interaction is a user (in)validation for a given pair of schema elements (Bonifati et al, To appear in 2011). When a tool discovers many relevant correspondences, the user has less interactions to correct and find the missing ones. The plot clearly shows that the tuned matcher improves quality and consequently reduces post-match effort.

Tuning, either automatic or manual, is performed during the pre-match phase of the schema matching process. The main motivation for tuning a schema matcher deals with the difficulty to know in advance, even for a human expert, the best configuration of the parameters for a given set of schemas. The heterogeneity, structure and domain specificity encompassed in every set of schemas to be matched make it more difficult for a schema matcher to achieve acceptable results in all cases. Thus, tuning enables schema matchers to provide flexibility and customization in order to cope with the different features of each set of schemas.

However, tuning the parameters to fulfill this goal is not an easy task for the user. Indeed, it has recently been pointed out that the main issue is how to select the most

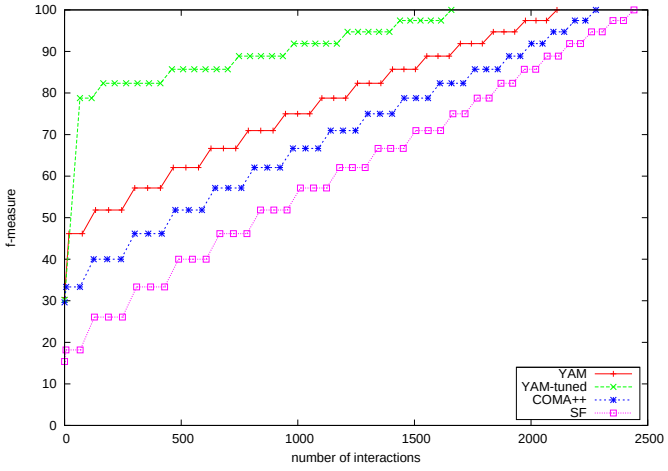


Fig. 1 A Tuned Matcher mainly Improves Matching Quality

suitable similarity measures to execute for a given domain and how to adjust the multiple parameters (Lee et al, 2007). Due to the numerous possible configurations of the parameters, it is not possible to try them all. Besides, they require specific knowledge from the users. Let us imagine that a user has to choose one similarity measure for matching his/her schemas and to assign a threshold to this measure. Selecting the appropriate similarity measure first implies that the user is a domain expert. Further, assigning the threshold means that the user has some background knowledge about the chosen measure, e.g., its value distribution.

One of the ten challenges for ontology matching focuses on the tuning issue (Shvaiko and Euzenat, 2008). Authors claim that this tuning is splitted in three categories : (i) matcher selection, (ii) matcher tuning and (iii) combination strategy of the similarity measures. In the first category, we distinguish manual selection from automatic selection. In the former, evaluation of different matchers and benchmarking tools facilitate the choice of a matcher for a given task (Yatskevich, 2003; Do et al, 2002; Duchateau et al, 2007; Ferrara et al, 2008). On the contrary, there exists a few tools that automatically select and build a schema matcher according to various parameters (YAM (Duchateau et al, 2009a,b)). The second category is mainly dedicated to tools like eTuner (Lee et al, 2007), which automatically tunes a schema matcher with its best configuration for a given set of schemas. Any schema matcher which provides the possibility to manually change the value of one or more of its parameters also falls in this catagory. The last category gathers the matchers which provide a manual combination of similarity measures (e.g., COMA++ (Aumueller et al, 2005), BMatch (Duchateau et al, 2008b)) and those which automatically combines these measures (SMB (Anan and Avigdor, 2008) and MatchPlanner from (Duchateau, 2009)). Note that in the rest of this chapter, we consider that the combination strategy is one parameter that can be tuned. In other words, the third category is merged into the second one.

The rest of the chapter is a survey about most popular parameters in schema matching and the tuning systems. We have gathered these parameters according to the entities against which they are applied: input data, similarity measures, combination of similarity measures and finally the matcher. This means that a tool may be described at different levels, according to the parameters that they enable to tune. Thus, the chapter is organized as follows: Section 2 covers the main notions about tuning. We then present the different parameters that one might face when using a matcher. These parameters have been sorted in four categories: in Section 3, we present the parameters related to input data and user preferences. Then, we describe in Section 4 low-level parameters involved in the schema matching process, namely those dealing with the similarity measures. One level higher, we find parameters which aim at combining the similarity measures. They are presented in Section 5. The highest level is the matcher selection and the involved parameters are discussed in Section 6. Finally, we conclude and we outline perspectives in Section 7.

2 Preliminaries

For many systems, tuning is an important step to obtain expected results or to optimize either matching quality or execution time. In the schema matching context, this statement is easily checkable due to the large amount and the diversity of available parameters provided by schema matchers. We now formalize the problem of tuning a schema matcher. As depicted by figure 2, the schema matching process requires as inputs at least two schemas, and optional **parameters** which can be given a **value**. These values belong to a specific **domain**. We mainly distinguish three types of domains:

- a finite (multi-)valued set, e.g., a list of synonyms <(*author*, *writer*), ..., (*book*, *volume*)>
- an unordered discrete domain, e.g., mapping cardinality can be *1:1*, *1:n*, *n:1*, or *n:m*
- an ordered continuous domain, e.g., a threshold for a similarity measure in the range [0, 1]

Similarly to (Lee et al, 2007), we call **knob** a parameter with an associated value. However, we do not restrict knobs to have values from a finite valued set. Here are examples of knobs: *(mapping cardinality, 1:1)* and *(threshold$_{trigrams}$, 0.15)*.

More formally, we define $S = < s_1, s_2, ..., s_n >$ the set of input schemas that the user wants to match. The parameters are represented by the set $P = < p_1, p_2, ..., p_k >$. The value domains are gathered in a set $D = < d_1, d_2, ..., d_n >$ where each $d_i \in D$ is a set $< value_{i1}, value_{i2}, ..., value_{it} >$. Finally, $K = < k_1, k_2, ..., k_l >$ stands for the set of knobs or the configuration of a schema matcher. With these definitions, we propose a *Match* function which uses any schema matcher with a configuration k to match a set of schemas s. The output of the *Match* function with the configuration k is a set of correspondences m_k.

$Match(s, k) = m_k$

As described in (Bonifati et al, To appear in 2011), it is possible to measure the quality of this set of correspondences, e.g., precision, recall and F-measure. We note

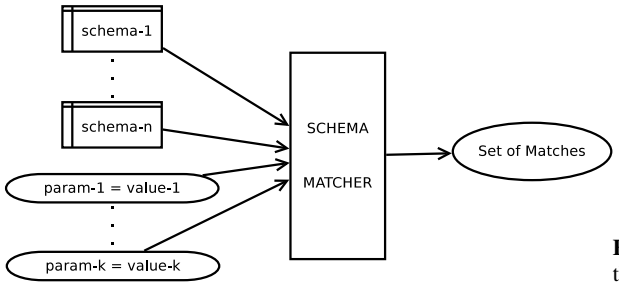


Fig. 2 Inputs and Outputs of the Schema Matching Process

Fmes the F-measure applied to a set of correspondences m_k.

$Fmes(m_k) := [0, 1]$

Thus, an optimal tuning k in this context consists of finding a configuration function ζ applied to parameters and domains so that the output of the schema matcher m_k is optimal given the input schemas. In other words, the configuration of the knobs would be perfectly tuned to achieve the best matching quality. That is, changing the value of any knob would decrease the matching quality.

Given that ζ(s, p, d) = k and *Match*(s, k) = m_k,
$\nexists\, z = \zeta$(s, p, d) and *Match*(s, z) = m_z with Fmes(m_z) > Fmes(m_k)

Likely, the measure of satisfaction over the ouput deals with quality (F-measure). But it is also possible to tune a schema matcher to optimize time performance, for instance with decision trees (Duchateau et al, 2008a).

In the next sections, we discuss the different parameters that one may face with when using a schema matcher, based on these definitions.

3 Input and Data Parameters

In this section, we gather the data and input parameters that one may have to configure when using a schema matcher. We do not consider that the input schemas belong to the tuning parameters. Indeed, a set of input schemas is compulsory to run the matcher. Thus, this section is dedicated to parameters that may side along with the input schemas (e.g., expert correspondences, data instances) or parameters related to techniques used by the matcher (e.g., machine learning, external resources used by a similarity measure). Indeed, most of these parameters directly impact the quality or the time performance. Deciding whether to provide any of them, as well as the choice of the parameters' values, is inherent to the tuning phase. The section is organized according to the type of parameters. First, data parameters include expert feedback. Such a reliable knowledge aims at improving the matching quality by reusing entities that have been checked by a domain expert. This feedback, as well as data instances, is often combined with machine learning techniques to exploit them. These machine learning techniques hold various parameters to be efficient and/or flexible, and we study them in the second part. The third category gathers external resources, which mainly consist of providing an ontology or dictionary. Finally, due to the complexity of the matching process and the design of numerous matchers, there exist very specific parameters that one may only face by using a given tool.

3.1 Expert feedback

Expert feedback mainly consists of correct mappings between the schemas to be matched. These mappings can be seen as a bootstrap for the schema matcher, i.e., this is knowledge taken as input by machine learning algorithms to classify schema instances. It may be a compulsory parameter like in LSD/Glue (Doan et al, 2001, 2003) and APFEL (Ehrig et al, 2005).

Conversely, providing mappings is an extra option to improve matching quality with tools such as YAM. As explained in (Duchateau et al, 2009a,b), each schema is built with a given "design methodology" (e.g., naming labels using underscores between tokens, using labels from an ontology, etc.). Consequently, by providing correct mappings, the system is able to infer, during the learning process, which similarity measures are the most efficient between elements of the mappings. Since a schema designer mainly keeps the same "design methodology" to build the whole (sub)schema, the similarity measures which have been detected as efficient with the correct mappings may also be efficient to discover new mappings.

Other tools have been designed to store correspondences and reuse them later (Madhavan et al, 2005). This is called a *reuse strategy* in matchers such as COMA++ (Aumueller et al, 2005) or Quickmig (Drumm et al, 2007). Actually, these tools are able to derive new correspondences when different successive matching processes involve the same schema. This feature is specifically useful when one of the schemas has been modified.

3.2 Machine Learning Parameters

Many schema matchers (partly) rely on machine learning techniques to discover correspondences between schemas. We distinguish two use cases of machine learning techniques: (i) as a similarity measure or (ii) as a "matcher" to combine measures.

3.2.1 Parameters at the Similarity Measure Level

In most cases, these learning techniques are applied against schema instances as part of a similarity measure. We can cite many works which have at least one such measure (Drumm et al, 2007; Li and Clifton, 2000; Berlin and Motro, 2002; Hernandez et al, 2002; Doan et al, 2003; Dhamankar et al, 2004).

The most common machine learning parameter deals with the training data. First, a suitable set of training data is a crucial issue common for all machine learning approaches. Second, users also have to cope with the number of training data. Matching tools are either provided with a knowledge base, thus enabling the storage and

reuse of these data. Or the tools do not require too many training data to be efficient, since this woud not be realistic. For example, if a user needs to match 100 data sources, (s)he can manually find the mappings for a few data sources and LSD discovers the others for the remaining sources (Doan et al, 2001). Due to the availability of training data and the classifier used, tuning this parameter is complicated. To illustrate this, we have partly reproduced a table from (Ehrig et al, 2005), shown as Table 1. We have limited this excerpt to two matching datasets (*Russia* and *biblio*) and to one Apfel's classifier (the decision tree). It depicts how the number of training data has a significant impact on the matching quality (in terms of precision, recall and F-measure). For instance, we notice that providing 20 training data in the *Russia* dataset enables the best precision (83%). This precision value tends to decrease with more training data. On the contrary, using 20 training data with the *biblio* dataset is clearly not sufficient.

Table 1 Impact of the number of training data on the matching quality with Apfel's decision tree

Dataset	Number of training data	Precision	Recall	F-measure
Russia	20	83%	48%	60%
	50	82%	47%	60%
	150	72%	59%	65%
Biblio	20	01%	28%	01%
	50	46%	25%	32%
	150	63%	38%	47%

Not only the number of training data may be crucial, but their validity too. For instance, APFEL (Ehrig et al, 2005) uses both positive and negative examples for training its classifiers. In this context, it is easier to provide sufficient training data to the system: authors explain that an initial matcher performs a matching over sample data and let users rate the discovered correspondences. The rated list of correspondences is then given as input to APFEL. From this list, the tools determines heuristic weights and threshold levels using various machine learning techniques, namely decision trees, neural networks, and support vector machines.

Another work aims at classifying candidate correspondences (either as relevant or not) by analysing their features (Naumann et al, 2002). The features represent boolean properties over data instance, such as presence of delimiters. Thus, selecting an appropriate feature set is a first parameter to deal with. The choice of a classifier is also important, and authors propose by default the Naive Bayes classifier for categorical data and quantile-based classifier for numerical data.

Similarity measures based on machine learning may not always stand for the most effective. The ASID matcher (Bozovic and Vassalos, 2008) considers its Naive Bayes classifier (against schema instances) as a less credible similarity measure, which is applied after user (in)validation of initial results provided by more reliable

measures (Jaro and TF/IDF). We think that this credibility of machine learning based similarity measures heavily depends on the quality of their training data.

3.2.2 Parameters at the Matcher Level

The second category of matchers uses machine learning techniques to combine similarity measures. However, they share almost the same parameters than the first category.

SMB (Anan and Avigdor, 2008) is based on the *Boosting* algorithm. In addition to training data, this approach also needs two parameters. The former is a hypothesis space, which is in this case a pair of similarity measures chosen among a pool. It appears that the similarity measures that perform well when used alone are mainly not included in the hypothesis space when combined with another one. The latter is an error measure which aims at both stopping the algorithm (when the computed error value reaches a threshold, 0.5 by default) and at selecting at each iteration the similarity measure which produced less errors. The authors have noticed that this error value is quickly reached, and therefore have added a pre-processing step to remove all pairs of schema elements that have been classified as irrelevant by all classifiers.

In YAM (Duchateau et al, 2009a,b), the number of training data, extracted from a knowledge base, is either provided by users or chosen according to empirical evaluation results. This tool can also be trained with similar schemas. This means that users may already have schemas that have been matched and could be reused to improve the results. By similar, authors indicate that either the schemas belong to the same domain (e.g., biology, business) or share some features (e.g., degree of heterogeneity, nested structure).

3.3 External Resources

External resources have long been useful to bring reliable knowledge into the schema matching process. In addition to the availability and security issues, user should check the adequacy of the resource content for the given matching task and its integration within the matcher. Different types of resources are accepted by schema matchers. The simplest one is a list of similar labels, also called list of synonyms. COMA++ (Aumueller et al, 2005) and Porsche (Saleem et al, 2008) let users fill in these resources. List of abbreviations are very common to extend the labels of ambiguous schema elements, like in COMA++ (Aumueller et al, 2005).

Another type of external resources is the domain ontology, used by Quickmig (Drumm et al, 2007) for instance. Similarly, Porsche (Saleem and Bellahsene, 2009)

is enhanced by data mining techniques applied to many domain ontologies to extract mini-taxonomies, that are finally used to discover complex mappings.

The Wordnet dictionary (Wordnet, 2007) is also used in different fashions: it facilitates the discovery of various relationships (e.g., synonyms, antonyms, etc.) in approaches such as YAM (Duchateau et al, 2009a,b) and S-MATCH/S-MATCH++ (Giunchiglia et al, 2004; Avesani et al, 2005). A dictionary can also become the core of the system against which all schema elements are matched, as performed by AUTOPLEX/AUTOMATCH (Berlin and Motro, 2001, 2002).

3.4 Other Input Parameters

Due to their diversity and their internal algorithms, schema matchers may have very specific parameters and/or user preferences. Here we propose to detail some of them.

In (Drumm et al, 2007), the Quickmig approach requires users to fill in a **questionnaire**. It then uses the answers to reduce the size of input schemas based on user domain knowledge. This parameter is useful when only a subpart of schemas needs to be matched or when dealing with large schemas.

Although most schema matchers implicitly promote precision, YAM (Duchateau et al, 2009a,b) is the first tool that enables users to tune **a preference towards precision or recall**. This choice impacts the machine learning process by avoiding the discovery of irrelevant mappings or by preventing the missing of relevant ones. As explained by the authors, promoting precision (respectively recall) often has a negative impact on recall (respectively precision). Figure 3 depicts the evolution of precision, recall and F-measure averaged for 150 datasets when the weight applied to false negatives increases (thus promoting recall). It appears that F-measure slightly increases by 7% while recall value improves up to 20% to the detriment of precision. Approaches that use a threshold to select correspondences also have a means of promoting recall by lowering the value of this threshold.

In Anchor-PROMPT (Noy and Musen, 2001), authors have chosen to compare paths of schema elements. As a consequence, specific parameters are used, like **the maximum length of a path**, **the number of elements involved in an equivalence group**, etc. End-users may have to understand the basics of Anchor-PROMPT algorithm to be able to correctly tune its parameters.

To the best of our knowledge, AgreementMaker (Cruz et al, 2007, 2009) is the only tool that enables users to select a type of cardinality to be discovered. Given two input schemas, mapping cardinality is either *1:1*, *1:n*, *n:1* or *n:m*. In a *1:1* configuration, the matcher is limited to discover mappings between one element of the first schema and one element in the other schema. Only a few matchers empha-

size the complex mappings such as *n:m*, in which any number of elements in both schemas can be involved in a mapping.

3.5 Conclusion

In this section, we have mainly presented user inputs, i.e., optional preferences and parameters applied to data. To sum up, the quality can be improved by using **external resources** and **expert feedback**. Several tools are based on **machine learning techniques** either as a similarity measure (mostly at the instance level) or as a means of combining the results of similarity measures. In both cases, training data is a crucial issue. Finally, many tools propose preferences or options which add more flexibility or may improve the matching quality. The next section focuses on the parameters at the similarity measure level.

4 Similarity Measures Parameters

Similarity measures are the basic components of schema matchers. They can be used as individual algorithms or combined with an aggregation function. Consequently, they may have internal parameters. In most cases, schema matchers do not enable users to tune such low-level parameters. Another parameter applied to similarity measures is the threhold. It filters the pair of schema elements in different categories (e.g., is a correspondence, or should apply another type of similarity measure) based on the output of the similarity measures. The last part of this section is dedicated to parameters specific to one or several matchers.

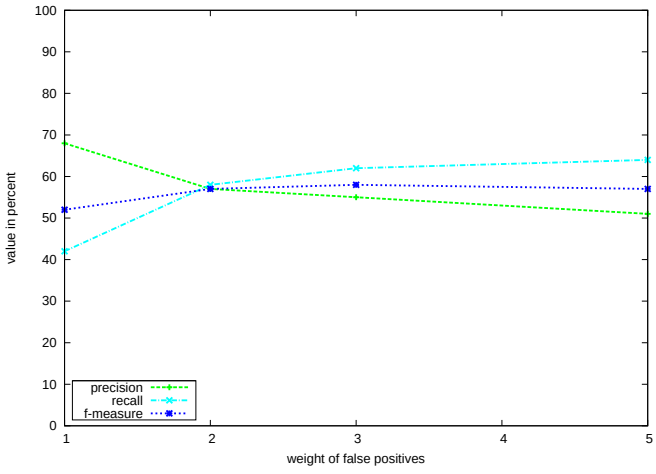


Fig. 3 Impact on the Matching Quality when Promoting Recall with YAM

4.1 Internal Parameters

Similarity measures takes as input two schema elements, and it outputs a similarity between them. This similarity value may be a numerical value (e.g., a distance, a real in the range [0, 1]) or a relationship (e.g., equivalence, generalization). Similarly to black-box algorithms, similarity measures can have internal parameters which impact the output. Due to the numerous available similarity measures, we do not intend to describe all of them with their parameters. Thus, we focus on two simple examples to illlustrate various types of such internal parameters.

The first example is the Levenhstein distance (Levenshtein, 1966) between two character strings. It computes the minimal number of operations costs needed to transform one source string into the target string, where an operation is an insertion, deletion, or substitution of a single character. Each operation may have a different cost. For instance, a substitution can have a cost equal to 2, while insertions and deletions may cost 1. Users can tune these costs according to their needs.

Between the two string *dept* and *department*, one requires 6 character insertions to transform *dept* into *department*. If an insertion costs 1, then the Levenhstein distance between both strings is 6.

The second similarity measure that we study is Jaro Winkler (Winkler, 1999). This measure is also terminological and it compares two character strings. It extends Jaro measure by taking into account the order of the characters of both strings. Furthermore, it promotes higher similarity values between strings which share similar prefixes. Consequently, it includes two parameters. The first one is the length of the prefix substring while the latter represents a constant scaling factor for how much the score is adjusted upwards for having common prefixes.

For further reading, we advise you to check the following list of resources (Cohen et al, 2003; Euzenat et al, 2004). Several packages also describe similarity measures and their parameters, for instance SecondString[1] or SimMetrics[2].

4.2 Thresholds

Most similarity measures are normalized to return a value in the range [0, 1]. Among all candidate pairs of schema elements, selecting the ones to propose as mappings can be performed with a threshold. That is, all candidate pairs whose similarity value (from one measure or resulting from a combination of several measures) is above a given threshold become mappings. Many tools (Avesani et al, 2005; Mad-

[1] SecondString (May 2010): *http://sourceforge.net/projects/secondstring/*

[2] SimMetrics (May 2010): *http://www.dcs.shef.ac.uk/∼sam/stringmetrics.html*

havan et al, 2001; Duchateau et al, 2008b; Drumm et al, 2007) have a threshold for selecting mappings. In most cases, a default value for the threshold is provided with the tool, e.g., 0.6 for the string-matching threshold in S-Match (Giunchiglia et al, 2007). COMA++ (Aumueller et al, 2005) includes a threshold often combined with a *top-K* strategy (i.e., the best *K* correspondences are returned) and a *MaxDelta* strategy (i.e., the best correspondence is returned with the closest ones, whose score only differs by a *Delta* tolerance value). Conversely, APFEL (Ehrig et al, 2005) is a machine learning based tools which features an automatic threshold tuning.

As the value distribution is very different from a similarity measure to another, a schema matcher can have one specific threshold for each similarity measure. This is the case with MatchPlanner (Duchateau et al, 2008a). The extended version of this matcher enables the automatic learning of these thresholds thanks to machine learning techniques (Duchateau, 2009). Similarly, Anchor-PROMPT (Noy and Musen, 2001) automatically computes the threshold values by averaging all similarity scores obtained on different runs with various parameter configurations.

In a broader way, authors of (Melnik et al, 2002) discuss the notion of filters to select the mappings. These filters not only include the thresholds, but also constraints between elements (types and cardinality) and a selection function.

Note that the threshold may be a parameter applied to a global similarity value, i.e., different similarity values are aggregated into a global one (given a strategy, see Section 5) and the threshold represents the decision-maker for accepting the pair of schema elements as a correspondence or not.

4.3 Various

Contrary to most aggregation-based approaches, Similarity Flooding/Rondo (Melnik et al, 2002, 2003) uses a graph propagation mechanism to refine similarities between schema elements. Thus, it holds specific parameters. The first one is fixpoint formula, which enables the computation of updated similarities and the end of execution of the propagation. Different fixpoint formulas have been tested and evaluated in (Melnik et al, 2002). In addition, several filters are proposed to select among all candidate pairs the ones that Rondo displays as mappings. Constraints (on cardinality and types) or thresholds are examples of filters.

For a given schema element, we do not know in advance to how many elements it should be matched (Avigdor, 2005). However, approaches such as COMA++ (Aumueller et al, 2005) or iMAP (Dhamankar et al, 2004) can display the top-K correspondences (for future interactive mode), thus enabling users to disambiguate complex correspondences. Other works have been specifically designed to discover complex mappings, such as Porsche (Saleem and Bellahsene, 2009).

4.4 Conclusion

This section describes the parameters related to similarity measures. Although they have a significant impact, **parameters inside the similarity measures** are often set to default values. Schema matching tools let users tune the **thresholds**, which is a traditional decision maker for deciding what happens to a pair of schema elements. Finally, we have detailed **specific parameters**, that users have to understand before optimizing the matchers. In the next section, we reach one level up by studying the parameters related to the combination of similarity measures.

5 Parameters for Combining Similarity Measures

At a higher level, schema matchers have to combine the results computed by different similarity measures. This enables an increase of the matching quality in most cases. However, the method for combining these results is crucial to derive high-quality mappings. The matcher first normalizes all similarity values. Different strategies are adopted to fulfill this goal. The first part of this section describes these different strategies. Several tools have been designed to enhance the interactivity with users for selecting the best strategy. These tools are presented in the second part of this section. Finally, we focus on a specific strategy which is widely used by matchers: the linear regression. It mainly encompasses weights to reflect each similarity measure's influence when combining them.

5.1 Strategy for Combining Similarity Measures

As many schema matchers use different similarity measures (based on string-matching, semantic, linguistics, structure, etc.), they need to adopt a strategy for combining these measures. In most cases, schema matchers combine the results computed by similarity measures, after a normalization step of the values (e.g., in the range $[0, 1]$).

One of the simplest method to combine similarity values is the average function, used by tools like ASID (Bozovic and Vassalos, 2008) or BMatch (Duchateau et al, 2008b). Aggregating the similarity values using weights reflects the impact of each measure in the matching process. In other words, it is possible to promote measures that are based on reliable resources (e.g., dictionaries, ontologies) by assigning them a high weight (see Section 5.2 for more details).

More complex strategies are found within COMA++ (Aumueller et al, 2005) in which similarity measures, types of nodes along with context, and fragments (parts) of the schema can be tuned. These strategies are then applied to the matrix built by

COMA++ to deduce the correspondences that are displayed to the user. The three main combination steps are aggregation (e.g., *weighted*, *max* or the default *average*), direction (e.g., *unidirectional* or *stable marriage*) and selection (e.g., *threshold*, *maxN*) (Do and Rahm, 2002). A strategy is built by choosing a value for each of these three steps. Figure 4 depicts an overview of COMA++ graphical interface for selecting a strategy. We also notice that a strategy may be performed on specific elements of the schemas (nodes, leaves, etc.).

MatchPlanner is a schema matcher based on decision trees to combine similarity measures. Although it has been extended by machine learning techniques to generate these decision trees (Duchateau, 2009), users can provide their own decision trees to the system (Duchateau et al, 2008a). There is no weight on the measures, but their order and position in the decision tree are crucial. Manually designing such decision trees is interesting when one wants to promote time performance or use a specific similarity measure.

In SMB (Anan and Avigdor, 2008), the output of a weak similarity measure (called *first-line matcher*) is combined with a decision maker (or *second-line matcher*) to discover correspondences. The combination strategy depends on the decision maker, which can be *Maximum Weighted Bipartite Graph* algorithm, *Stable Marriage*, etc.

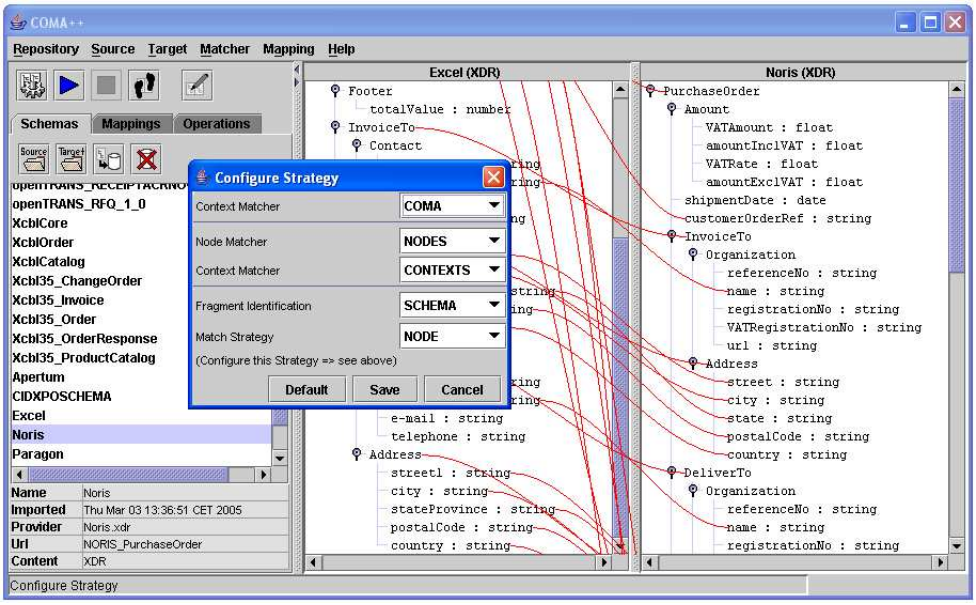


Fig. 4 COMA++ User Interface for Selecting Combination Strategy

In YAM (Duchateau et al, 2009a,b), the combination of similarity measures is performed by a machine learning classifier. Authors consider that any classifier is a matcher since it classifies pairs of schema elements as relevant or not. Thus, the combination of the similarity measures depends on the type of classifier (decision tree, Bayes network, neural network, etc.).

To sum up, many tools have designed their own strategies to combine similarity measures. However, most of them are based on weighted functions that the users may have to tune.

5.2 Weights in Formulas

Previously, we have detailed different types of strategies. One of the most common strategy in the matching community is the linear regression to aggregate values computed by similarity measures. In that case, the weights given to each measure is important according to the domain and the schemas to be matched. For instance, if a domain ontology is available, one may decide to give a high weight to the measures which are able to use this ontology. However, manually tuning these these weights still requires user expertise.

A simple example of aggregation function is demonstrated with BMatch (Duchateau et al, 2008b) or Cupid (Madhavan et al, 2001). Their authors aggregate the results of terminological measure with the ones computed by a structural measure by varying the weights applied to each measure ($\frac{1}{2}$ and $\frac{1}{2}$, $\frac{1}{3}$ and $\frac{2}{3}$, etc.).

In most tools, default values are given to these weights. They are mainly the results of intensive experiments. For example, the default weights of COMA++'s *name* and *data type* similarity measures are 0.7 and 0.3 respectively (Do and Rahm, 2002). As explained in Glue (Doan et al, 2003) or APFEL (Ehrig et al, 2005), it is possible to automatically tune the weights of an aggregation function thanks to machine learning techniques.

To help tuning the weights in aggregating functions, we discuss the iMAP approach (Dhamankar et al, 2004). This matcher mainly provides a new set of machine-learning based measures for discovering specific types of complex mappings (e.g., *name* is a concatenation of *firstname* and *lastname*). It also includes an explanation module to clarify why a given correspondence has been discovered to the detriment of another candidate. For instance, this module is able to describe that a string-matching classifier has a strong influence for a discovered correspondence. Thus, user can use this feedback to decrease the weight of this classifier.

5.3 Supporting Users to Revise Strategies

Although most matchers simply provide a graphical user interface to visualize the results, recent works have pointed out a need for selecting the best strategy. For instance, including some mechanisms to easily update the weights of a function so that users can directly analyse impacts of these changes.

Here, we describe recent works that aim at supporting users during the tasks of selecting appropriate similarity measures and combining them. To combine them efficiently, weights have to be efficiently tuned. To support users during these tasks, two tools have been designed : AgreementMaker and Harmony. Whatever the technique they use (interactions with users or strategy filters), they enable a revision of the current strategy by adding, removing or modifying parameters and similarity measures involved in the combination. We further describe each of these tools in the rest of this part.

5.3.1 AgreementMaker

The originality of AgreementMaker (Cruz et al, 2007, 2009) is the capability of matching methods combination. Moreover, it provides facilities for tuning manually the quality of matches. Indeed, one of interesting features of AgreementMaker is a comprehensive user interface supporting both advanced visualization techniques and a control panel that drives the matching methods. This interface, depicted by Figure 5, provides the user facilities to evaluate the matching process, thus enabling the user to be directly involved in the loop and evaluation strategies.

AgreementMaker provides a combination strategy based on the linear interpolation of the similarity values. The weights can be either user assigned or evaluated through automatically-determined quality measures. The system allows for serial and parallel composition where, respectively, the output of one or more methods can be used as input to another one, or several methods can be used on the same input and then combined.

5.3.2 Harmony

Harmony schema matcher (Mork et al, 2008; Smith et al, 2009) combines multiple matcher algorithms by using a vote merger. The vote merging principle is a weighted average of the match scores provided by each match voter. A match voter provides a confidence score for each pair of schema elements to be matched. Then, Similarity Flooding strategy (Melnik et al, 2002) is applied to adjust the confidence scores based on structural information. Thus, positive confidence scores propagate up the graph. An interesting feature of Harmony lies in its graphical user interface for viewing and modifiying the discovered schema correspondences. This allows to assist the users to focus their attention on different ways. This assistance is done through a filter, which is a predicate that is evaluated against each candidate corre-

spondence. Harmony supports two kinds of filters. The first kind named *link filters* depends on the characteristics of a candidate correspondence. For example, applying the confidence filter will have as effect to graphically display those correspondences whose match score falls within the specific range of values. The second one named *node filters* is related to a schema element characteristics. This kind of filters includes a depth and a sub-tree filter. For example, in Entity Relationship schemas, entities appear at level 1 whereas attributes are at level 2. In this case, by using the depth filter with value *1*, the user may focus on matching entities. While the sub-tree filter is useful in tree based model like XML schemas.

5.4 Discovering the Best Configuration

The previous section gathers tools that support users to manually find the best strategy, i.e. the best method for combining similarity measures and its optional parameters such as weights. This last part is dedicated to the tools that automatically discover the best strategy : eTuner and YAM.

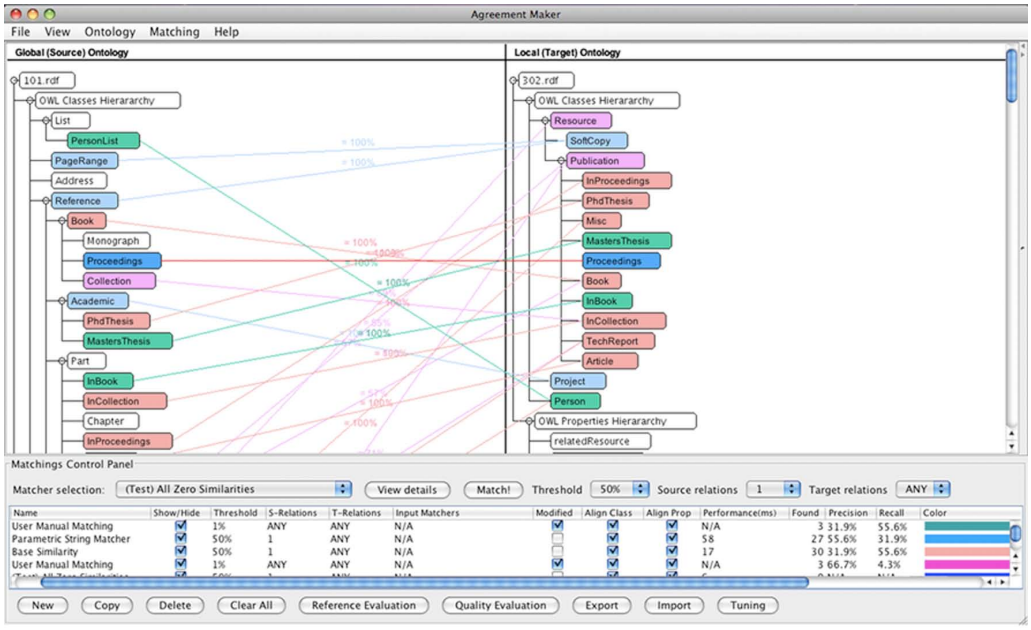


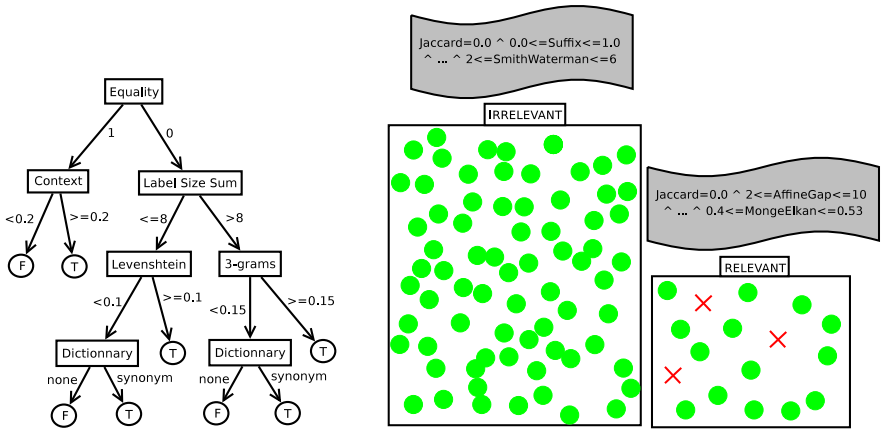
Fig. 5 AgreementMaker User Interface for Testing and Comparing Combination Strategies

5.4.1 eTuner

eTuner (Lee et al, 2007) is not a schema matching tool, but it aims at automatically tuning them. It proceeds as follows: from a given schema, it derives many schemas which are semantically equivalent. The mappings between the initial schema and its derivations are stored. Then, a given matching tool (e.g., COMA++ or Similarity Flooding) is applied against the schemas and the results is compared with the stored set of mappings. This process is repeated until an optimal parameters configuration of the matching tool is found, i.e., the mappings discovered by the matching tool are mostly similar to those stored. eTuner strongly relies on the capabilities of the matching tool that it tunes. In most experiments, eTuner is able to improve matching quality by 1 to 15% compared to the tools with their default configuration.

5.4.2 YAM

Similarly to MatchPlanner, YAM (Duchateau et al, 2009a,b) takes some user inputs and it uses them to produce a schema matcher. Although MatchPlanner is limited to combine the similarity measures with a decision tree, YAM is able to combine them thanks to any machine learning classifier. All low-level parameters such as weights and thresholds are therefore automatically tuned during the learning process. The combination of similarity measures only depends on the type of classifier selected by YAM. For instance, Figures 6(a) and 6(b) depict two techniques for combining similarity measures. The first one is based on a decision tree while the second one uses NNge. With the decision tree, each pair of schema elements is matched with similarity measures from the root until a leaf node is reached, indicating whether the pair is a correspondence (T) or not (F). The value of the previously computed similarity measure is used to decide which edge (and consequently which similarity measure) should be executed next. Combining with a decision tree enables a sparing of resources since all similarity measures may not be computed for a given pair of schema elements. On the contrary, NNge classifier builds groups of nearest neighbour pairs of schema elements and then finds the best rule, expressed by boolean logic, for each group. YAM currently includes 20 classifiers from the Weka library (Garner, 1995). According to (Duchateau, 2009), experiments show that the tuned matchers produced by YAM are able to improve F-measure by 20% over traditional approaches. Datasets mainly include average schemas from various domains, but also two datasets involving large schemas. Similarly to most machine learning based approaches, authors have noticed the fact that the results may vary according to training data, hence the need to perform different runs during experiments.



(a) With a Decision Tree (b) With Nearest Neigbour Generalized Examples

Fig. 6 YAM : Examples of Combination of Similarity Measures

5.5 Conclusion

In this section, we have described the **different strategies** to combine similarity measures and to tune them, mainly their **weights**. Fortunately, there exist several tools to help users revising or selecting the strategies. **Visual tools** support users for manually configuring these strategies, mainly thanks to state-of-the-art GUI. Finally, we have explored **automatic approaches** that are able to discover and tune the best strategy. In the following section, we are still heading one level higher. Indeed, the first choice of a user deals with the matching tool.

6 Matcher Selection

The selection of a schema matcher is obviously not a parameter: it does not fit with the definitions provided in Section 2. But this is likely meta-tuning since one first needs to choose a schema matcher before tuning its parameters and using it. Furthermore, some recent challenges directly refer to this issue (Shvaiko and Euzenat, 2008). The selection of a schema matcher may be guided by the results that it obtains using some benchmarking tools. In addition, a few recent works have been proposed to automatize this matcher selection. We describe each of them in the rest of this section.

6.1 AHP

Authors of (Malgorzata et al, 2006) have proposed to select a relevant and suitable matcher for ontology matching. They have used Analytic Hierarchical Process (AHP) to fulfill this goal. They first define characteristics of the matching process divided into six categories (inputs, approach, usage, output, documentation and costs). Users then fill in a requirements questionnaire to set priorities for each defined characteristic. Finally, AHP is applied with these priorities and it outputs the most suitable matcher according to user requirements. This approach suffers from two drawbacks: (i) there is no experiment demonstrating its effectiveness and (ii) currently there does not exist a listing of all characteristics for all matching tools. Thus, the user would have to manually fill in these characteristics.

6.2 RiMOM

RiMOM (Li et al, 2009) is a multiple strategy dynamic ontology matching system. Different matching strategies are applied to a specific type of ontology information. Based on the features of the ontologies to be matched, RiMOM selects the best strategy (or strategy combination) to apply. When loading the ontologies, the tool also compute three feature factors. The underlining idea is that if two ontologies share similar feature factors, then the strategies that use these factors should be given a high weight when computing similarity values. For instance, if the *label meaningful* factor is low, then the *Wordnet-based strategy* will not be used. Each strategy produces a set of correspondences, and all sets are finally aggregated using a linear interpolation method. A last strategy dealing with ontology structure is finally performed to confirm discovered correspondences and to deduce new ones. Contrary to other approaches, RiMOM does not rely on machine learning techniques. It is quite similar to the AHP work by selecting an appropriate matcher based on input's features. RiMOM participated to the 2009 OAEI campaign (Zhang et al, 2009). Results show that the tool performed well in different tracks (anatomy, benchmark, instance matching). For instance, it achieves F-measures above 75% for all datasets in the instance matching track.

6.3 YAM

YAM (Yet Another Matcher) (Duchateau et al, 2009a,b) enables the generation of *à la carte* schema matchers according to user requirements. It uses a knowledge base that includes a (possibly large) set of similarity measures and machine learning classifiers. All classifiers are trained with scenarios from this knowledge base (and optionally provided by the users). Their individual results (precision, recall and F-measure) are computed and according to the adopted strategy, the classifier which

achieves the best quality is selected as schema matcher. The strategy mainly depends on user inputs. For instance, if (s)he wants to promote recall, then the classifier with the best recall value is returned. If the user has provided expert mappings, then YAM selects as the schema matcher the classifier that obtains the best F-measure on this set of expert mappings.

6.4 SMB

In (Anan and Avigdor, 2008), the authors propose a machine learning approach, SMB. It uses the Boosting algorithm to classify the similarity measures, divided into first line and second line matchers. The Boosting algorithm consists in iterating weak classifiers over the training set while re-adjusting the importance of elements in this training set. Thus, SMB automatically selects a pair of similarity measures as a matcher by focusing on harder training data. An advantage of this algorithm is the important weight given to misclassified pairs during the training. Although this approach makes use of several similarity measures, it mainly combines a similarity measure (first line matcher) with a decision maker (second line matcher). Empirical results show that the selection of the pair does not depend on their individual performance.

6.5 STEM

In a broader way, the STEM framework (Köpcke and Rahm, 2008) identifies the most interesting training data set which is then used to combine matching strategies and tune several parameters such as thresholds. First, training data is generated, either manually (i.e., an expert labels the entity pairs) or automatically (at random, using static-active selection or active learning). Then, similarity values are computed using pairs in the training data set to build a similarity matrix between each pair and each similarity measure. Finally, the matching strategy is deduced from this matrix thanks to supervised learned algorithm. The output is a a tuned matching strategy (how to combine similarity measures and tune their parameters). The framework enables a comparative study of various similarity measures (e.g., Trigrams, Jaccard) combined with different strategies (e.g., decision tree, linear regression) whose parameters are either manually or automatically tuned.

6.6 Conclusion

This last section underlines the fact that selecting an appropriate schema matching tool is the first issue to be considered. A few works have been proposed in this do-

main, which is recognized as one of the ten matching challenges for the next decade (Shvaiko and Euzenat, 2008). If we exclude the *AHP* approach, for which no experiment is provided, the remaining tools are all based on machine learning techniques. This is an interesting feature since more datasets with correct correspondences are becoming available. However, discovering the features of a dataset to determine the most appropriate tool could be a challenging task.

7 Conclusion

In this chapter, we have provided an overview about what has been done for tuning schema matchers. At first, schema matchers enabled users to configure some of their low-level parameters (e.g., thresholds). They mainly allow to filter or select the output (the set of mappings). The next step deals with parameters for combining similarity measures. They add more flexibility and the set of discovered mappings depends on the configuration of these parameters. More recently, some works went up one level further by selecting the appropriate matcher for a given matching task. These tools lessen the burden of the user by automatically tuning most of the low-level parameters.

In the meanwhile, much effort has also been spent to integrate user preferences or input data parameters. Most of them are based on machine learning techniques so that schema instances or expert feedback can be used in the process. The integration of such parameters is often an extra means for improving matching quality. User preferences such as the promotion of precision or recall let users choose how they intend to manage post-match effort. These options are also interesting in contexts where high dynamicity leads to a quick evolution of data sources, thus implying that a high precision is prefered. On the contrary, recall can be promoted when data sources are going to be fully integrated and manually checked.

Although a default configuration should still be proposed with a matcher, we believe that we are heading towards a specific configuration of a schema matcher for a given matching task. Namely, various properties of the matching scenario can be computed by the tool. The latter can then deduce, based on previous experiments or properties values, the best configuration. Visual tools have a strong impact on the manual post-match effort. By displaying the results of different matching strategies, one has sufficient information to check and (in)validate the mappings. Combined with user preferences, these tools would clearly reduce manual post-match effort. To the best of our knowledge, there are currently no works which study the impact of the tuning (during pre-match effort) over matching quality (and post-match effort). A balanced effort between parameters that would bring significant impact on the matching quality given a matching task might be further investigated.

Acknowledgements We would like to thank our reviewers for their comments and corrections on this chapter. We are also grateful to colleagues who have accepted the publication of pictures from their tools.

References

Anan M, Avigdor G (2008) Boosting schema matchers. In: OTM '08: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems, Springer-Verlag, Berlin, Heidelberg, pp 283–300, DOI http://dx.doi.org/10.1007/978-3-540-88871-0_20

Aumueller D, Do HH, Massmann S, Rahm E (2005) Schema and ontology matching with COMA++. In: ACM SIGMOD, pp 906–908

Avesani P, Giunchiglia F, Yatskevich M (2005) A Large Scale Taxonomy Mapping Evaluation. In: Intl. Semantic Web Conf., pp 67–81

Avigdor G (2005) On the cardinality of schema matching. In: OTM Workshops, pp 947–956

Berlin J, Motro A (2001) Automated discovery of contents for virtual databases. In: CoopIS, pp 108–122

Berlin J, Motro A (2002) Database schema matching using machine learning with feature selection. In: CAiSE

Bonifati A, Bellahsene Z, Duchateau F, Velegrakis Y (To appear in 2011) Schema Matching and Mapping by Z. Bellahsene, A. Bonifati, E. Rahm, Data-Centric Systems and Applications, Springer, chap On Evaluating Schema Matching and Mapping

Bozovic N, Vassalos V (2008) Two-phase schema matching in real world relational databases. In: ICDE Workshops, pp 290–296

Carmel D, Avigdor G, Haggai R (2007) Rank aggregation for automatic schema matching. IEEE Trans on Knowl and Data Eng 19(4):538–553, DOI http://dx.doi.org/10.1109/TKDE.2007.1010

Cohen W, Ravikumar P, Fienberg S (2003) A comparison of string distance metrics for name-matching tasks. In: Proceedings of the IJCAI-2003, URL `http://citeseer.ist.psu.edu/cohen03comparison.html`

Cruz IF, Sunna W, Makar N, Bathala S (2007) A visual tool for ontology alignment to enable geospatial interoperability. J Vis Lang Comput 18(3):230–254

Cruz IF, Antonelli FP, Stroe C (2009) Agreementmaker: efficient matching for large real-world schemas and ontologies. Proc VLDB Endow 2(2):1586–1589

Dhamankar R, Lee Y, Doan A, Halevy A, Domingos P (2004) iMAP: Discovering Complex Semantic Matches between Database Schemas. In: ACM SIGMOD, pp 383–394

Do HH, Rahm E (2002) COMA - A System for Flexible Combination of Schema Matching Approaches. In: VLDB, pp 610–621

Do HH, Melnik S, Rahm E (2002) Comparison of schema matching evaluations. In: Web, Web-Services, and Database Systems Workshop

Doan A, Domingos P, Halevy AY (2001) Reconciling Schemas of Disparate Data Sources - A Machine Learning Approach. In: ACM SIGMOD

Doan A, Madhavan J, Dhamankar R, Domingos P, Halevy AY (2003) Learning to match ontologies on the Semantic Web. VLDB J 12(4):303–319

Drumm C, Schmitt M, Do HH, Rahm E (2007) Quickmig: automatic schema matching for data migration projects. In: CIKM, ACM, pp 107–116, DOI http://doi.acm.org/10.1145/1321440.1321458

Duchateau F (2009) Towards a generic approach for schema matcher selection: Leveraging user pre- and post-match effort for improving quality and time performance. PhD thesis, Université Montpellier II - Sciences et Techniques du Languedoc, URL http://tel.archives-ouvertes.fr/tel-00436547/en/

Duchateau F, Bellahsene Z, Hunt E (2007) Xbenchmatch: a benchmark for xml schema matching tools. In: VLDB, pp 1318–1321

Duchateau F, Bellahsene Z, Coletta R (2008a) A flexible approach for planning schema matching algorithms. In: OTM Conferences (1), pp 249–264

Duchateau F, Bellahsene Z, Roche M (2008b) Improving quality and performance of schema matching in large scale. Ingénierie des Systèmes d'Information 13(5):59–82

Duchateau F, Coletta R, Bellahsene Z, Miller RJ (2009a) (not) yet another matcher. In: CIKM, pp 1537–1540

Duchateau F, Coletta R, Bellahsene Z, Miller RJ (2009b) Yam: a schema matcher factory. In: CIKM, pp 2079–2080

Ehrig M, Staab S, Sure Y (2005) Bootstrapping ontology alignment methods with apfel. In: ISWC

Euzenat J, et al (2004) State of the art on ontology matching. Tech. Rep. KWEB/2004/D2.2.3/v1.2, Knowledge Web

Ferrara A, Lorusso D, Montanelli S, Varese G (2008) Towards a benchmark for instance matching. In: Shvaiko P, Euzenat J, Giunchiglia F, Stuckenschmidt H (eds) OM, CEUR-WS.org, CEUR Workshop Proceedings, vol 431, URL http://dblp.uni-trier.de/db/conf/semweb/om2008.html#FerraraLMV08

Garner SR (1995) Weka: The waikato environment for knowledge analysis. In: Proc. of the New Zealand Computer Science Research Students Conference, pp 57–64

Giunchiglia F, Shvaiko P, Yatskevich M (2004) S-Match: an Algorithm and an Implementation of Semantic Matching. In: European Semantic Web Symposium, pp 61–75

Giunchiglia F, Shvaiko P, Yatskevich M (2007) Semantic matching: Algorithms and an implementation. Tech. rep., DISI, University of Trento, URL http://eprints.biblio.unitn.it/archive/00001148/

Hernandez MA, Miller RJ, Haas LM (2002) Clio: A semi-automatic tool for schema mapping (software demonstration). In: ACM SIGMOD

Köpcke H, Rahm E (2008) Training selection for tuning entity matching. In: QDB/MUD, pp 3–12

Lee Y, Sayyadian M, Doan A, Rosenthal A (2007) etuner: tuning schema matching software using synthetic scenarios. VLDB J 16(1):97–122

Levenshtein V (1966) Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady 10:707

Li J, Tang J, Li Y, Luo Q (2009) Rimom: A dynamic multistrategy ontology alignment framework. IEEE Trans on Knowl and Data Eng 21(8):1218–1232, DOI http://dx.doi.org/10.1109/TKDE.2008.202

Li WS, Clifton C (2000) Semint: a tool for identifying attribute correspondences in heterogeneous databases using neural networks. Data Knowl Eng 33(1):49–84, DOI http://dx.doi.org/10.1016/S0169-023X(99)00044-0

Madhavan J, Bernstein PA, Rahm E (2001) Generic schema matching with cupid. In: VLDB, pp 49–58

Madhavan J, Bernstein PA, Doan A, Halevy AY (2005) Corpus-based Schema Matching. In: Intl. Conf. on Data Engineering, pp 57–68

Malgorzata M, Anja J, Jérôme E (2006) Applying an analytic method for matching approach selection. In: Shvaiko P, Euzenat J, Noy NF, Stuckenschmidt H, Benjamins VR, Uschold M (eds) Ontology Matching, CEUR-WS.org, CEUR Workshop Proceedings, vol 225, URL `http://dblp.uni-trier.de/db/conf/semweb/om2006.html#MocholJE06`

Melnik S, Garcia-Molina H, Rahm E (2002) Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: ICDE, pp 117–128

Melnik S, Rahm E, Bernstein PA (2003) Developing metadata-intensive applications with rondo. J of Web Semantics I:47–74

Mork P, Seligman L, Rosenthal A, Korb J, Wolf C (2008) The harmony integration workbench. J Data Semantics 11:65–93

Naumann F, Ho CT, Tian X, Haas LM, Megiddo N (2002) Attribute classification using feature analysis. In: ICDE, p 271

Noy N, Musen M (2001) Anchor-prompt: Using non-local context for semantic matching. In: Proc. IJCAI 2001 workshop on ontology and information sharing, pp 63–70

Saleem K, Bellahsene Z (2009) Complex schema match discovery and validation through collaboration. In: OTM Conferences (1), pp 406–413

Saleem K, Bellahsene Z, Hunt E (2008) Porsche: Performance oriented schema mediation. Inf Syst 33(7-8):637–657

Shvaiko P, Euzenat J (2008) Ten challenges for ontology matching. In: OTM Conferences (2), pp 1164–1182

Smith K, Morse M, Mork P, Li M, Rosenthal A, Allen D, Seligman L (2009) The role of schema matching in large enterprises. In: CIDR

Winkler W (1999) The state of record linkage and current research problems. In: Statistics of Income Division, Internal Revenue Service Publication R99/04

Wordnet (2007) http://wordnet.princeton.edu

Yatskevich M (2003) Preliminary evaluation of schema matching systems. Tech. Rep. DIT-03-028, Informatica e Telecomunicazioni, University of Trento
Zhang X, Zhong Q, Shi F, Li J, Tang J (2009) Rimom results for oaei 2009