COMA - A system for flexible combination of schema matching approaches

Hong-Hai Do
University of Leipzig
hong@informatik.uni-leipzig.de

Erhard Rahm
University of Leipzig
rahm@informatik.uni-leipzig.de

Abstract

Schema matching is the task of finding semantic correspondences between elements of two schemas. It is needed in many database applications, such as integration of web data sources, data warehouse loading and XML message mapping. To reduce the amount of user effort as much as possible, automatic approaches combining several match techniques are required. While such match approaches have found considerable interest recently, the problem of how to best combine different match algorithms still requires further work. We have thus developed the COMA schema matching system as a platform to combine multiple matchers in a flexible way. We provide a large spectrum of individual matchers, in particular a novel approach aiming at reusing results from previous match operations, and several mechanisms to combine the results of matcher executions. We use COMA as a framework to comprehensively evaluate the effectiveness of different matchers and their combinations for real-world schemas. The results obtained so far show the superiority of combined match approaches and indicate the high value of reuse-oriented strategies.

1 Introduction

Schema matching is the task of finding semantic correspondences between elements of two schemas [11, 12, 15]. It is a critical operation in many schema and data translation and integration applications, such as integration of web data sources, data warehouse loading, XML message mapping and XML-relational data mapping. Currently, schema matching is largely performed manually by domain experts, and therefore a time-consuming and tedious process. In web-based applications and services, such a manual approach is a major limitation due to the rapidly increasing number of data sources, XML message and document schemas, and web service interfaces to be dealt with. Hence, approaches for automating the schema matching tasks as much as possible are badly needed to simplify and speed up the development, maintenance and use of such applications.

**Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002**

Numerous researchers have addressed the schema matching problem either for specific applications [1, 4, 5, 7, 8, 9, 11, 15, 16] or in a more generic way for different applications and schema languages [12, 13, 14]. The proposed techniques for automating schema matching exploit various types of schema information, e.g. element names, data types and structural properties [2, 12, 15, 16, 9] as well as characteristics of data instances [7, 8, 14, 11, 9]. Some approaches utilize auxiliary sources, such as taxonomies, dictionaries and thesauri [2, 9]. To achieve high match accuracy for a large variety of schemas, a single technique (e.g., name matching) is unlikely to be successful. Hence, it is necessary to combine different approaches in an effective way. For this purpose, previous prototypes have followed either a so-called *hybrid* or *composite* combination of match techniques [18]. So far the hybrid approach is most common where different match criteria or properties (e.g., name and data type) are used within a single algorithm. Typically these criteria are fixed and used in a specific way. By contrast, a composite match approach combines the results of several independently executed match algorithms, which can be simple (based on a single match criterion) or hybrid. This allows for a high flexibility, as there is the potential for selecting the match algorithms to be executed based on the match task at hand. Moreover, there are different possibilities for combining the individual match results. We know of only three recent systems following such a composite approach [7, 8, 9]. They are all limited to match techniques based on machine learning and do not fully utilize the flexibility offered by the composite approach (see Section 2).

To investigate the effectiveness of composite match approaches more comprehensively we have developed the COMA system for combining match algorithms in a flexible way. COMA represents a *generic match system* supporting different applications and multiple schema types such as XML and relational schemas. It provides an *extensible library* of match algorithms and supports different ways for combining match results. New match algorithms can be included in the library and used in combination with other matchers. COMA thus allows us to tailor match strategies by selecting the match algorithms and their combination for a given match problem. Moreover, we use COMA as an *evaluation platform* to system-

atically examine and compare the effectiveness of different matchers and combination strategies. In the design of COMA we observed that in general fully automatic solutions to the match problem are not possible due to the potentially high degrees of semantic heterogeneity between schemas. We thus allow an *interactive* and *iterative* match process during which the user can provide feedback, e.g. to manually provide match correspondences or to confirm or reject proposed matches.

As another contribution we propose a new match approach that aims at reusing previously obtained match results, motivated by the observation that many schemas to be matched are very similar to previously matched schemas. Reusing the previous match results may thus result in significant savings of manual effort. A simple form of such an approach is the use of synonym tables indicating match correspondences at the level of single schema elements. Our new approach tries to reuse match results at the level of entire schemas or schema fragments. The flexibility of COMA is made possible by the use of a DBMS-based repository for storing schemas, intermediate similarity results of individual matchers, and complete (possibly user-confirmed) match results for later reuse.

The paper is organized as follows. In Section 2 we discuss some related work. Section 3 provides an overview of COMA. In Sections 4 and 5 we present the supported matchers including the reuse-oriented approach. Section 6 outlines the strategies for matcher combination. Section 7 presents the results of using COMA for evaluating different strategies for matching real-world schemas. Finally, we conclude and discuss some future work.

2 Related work

A recent survey on automatic schema matching proposed a solution taxonomy differentiating between schema- and instance-level, element- and structure-level, and language- and constraint-based matching approaches [18, 12]. Furthermore, the distinction between hybrid and composite combination of matchers is introduced and previous match prototypes such as Cupid [12], SemInt [11], LSD [7], Dike [16], SF [13], TranScm [15], and Momis [2] are reviewed.

Cupid [12] represents a sophisticated hybrid match approach combining a name matcher with a structural match algorithm, which derives the similarity of elements based on the similarity of their components hereby emphasizing the name and data type similarities present at the finest level of granularity (leaf level). In a comparative evaluation Cupid was generally more effective than two earlier match prototypes (Dike and Momis).

LSD (Learning Source Description) [7] and its extension GLUE [8] represent powerful composite approaches to combining different matchers. Both use machine-learning techniques for individual matchers and an automatic combination of match results. Machine learning is a promising technique especially for evaluating data instances to predict element similarity. On the other hand, the accuracy of the predictions depends on a suitable training which can incur a substantial manual effort. The predictions of individual matchers are combined by a so-called meta-learner, which weights the predictions from a matcher according to its accuracy shown during the training phase. In various experiments LSD and GLUE showed promising results, albeit based on a not well-defined accuracy metric apparently not taking into account wrongly proposed match correspondences.

In [9], Embley et al. describe another composite approach based on machine learning. In addition to instance-level matchers a name matcher is supported requiring an external dictionary (WordNet). The predictions of the individual matchers are combined using an average function. Like LSD and GLUE, a training phase is needed.

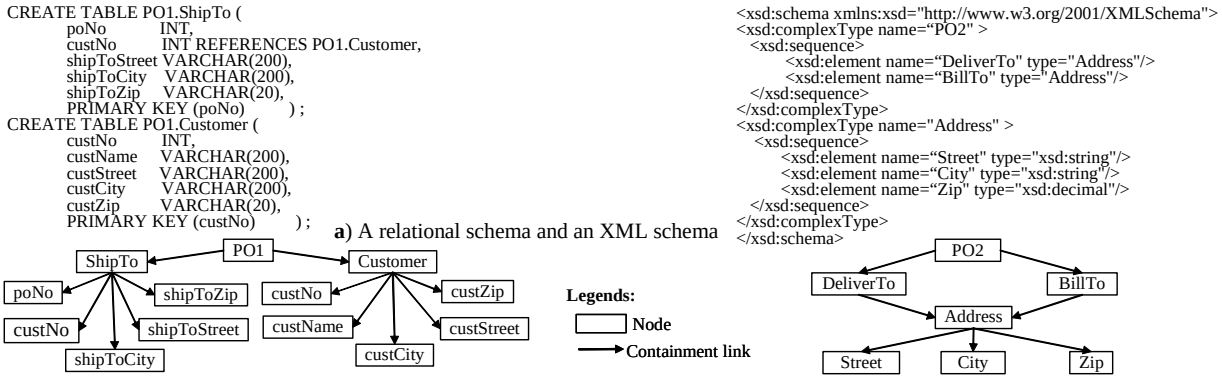
The evaluation of the structural match algorithm SF (Similarity Flooding) in [13] used a more realistic metric for measuring the match accuracy than previous studies. It takes into account both the share of correctly proposed match candidates and wrongly suggested match candidates. In our evaluation we will also use this refined metric (Section 7).

To sum up, the composite approach has so far only been studied in the context of machine learning approaches focusing on instance-level matchers and using a specific combination of match results. By contrast we want to support and evaluate a spectrum of matchers not confined to machine learning as well as the customizable combination of their results. A systematic comparative evaluation of different match algorithms and their combinations based on well-defined accuracy metrics does not exist so far. To our knowledge, beyond the use of simple synonym tables the reuse of previous match results has not yet been studied.

3 Overview of COMA

A schema consists of a set of elements, such as relational tables and columns or XML elements and attributes. In COMA we represent schemas by rooted *directed acyclic graphs*. Schema elements are represented by graph nodes connected by directed links of different types, e.g. for containment and referential relationships. Schemas are imported from external sources, e.g. relational databases or XML files, into the internal format on which all match algorithms operate. Figure 1 shows our running examples, a relational and an XML schema for purchase orders (PO), and their internal graph representation.

The match operation takes as input two schemas and determines a mapping indicating which elements of the input schemas logically correspond to each other, i.e. match. The match result is a set of mapping elements specifying the matching schema elements together with a similarity value between 0 (strong dissimilarity) and 1 (strong similarity) indicating the plausibility of their correspondence. Similar to previous work, we focus on one-to-one (1:1) match relationships. However, match algorithms may determine multiple match candidates with



a) A relational schema and an XML schema

b) Their corresponding graph representation

Figure 1. External and internal schema representation

different similarities for a schema element and finally select one of them or leave the final choice to the user.

Figure 2 illustrates match processing in COMA on two input schemas *S1* and *S2*. Match processing either takes place in one or multiple iterations depending on whether an automatic or interactive determination of match candidates is to be performed. Each match iteration consists of three phases: an optional user feedback phase, the execution of different matchers and the combination of the individual match results. In interactive mode, the user can interact with COMA for each iteration to specify the match strategy (selection of matchers, of strategies to combine individual match results), define match or mismatch relationships, and accept or reject match candidates proposed in the previous iteration. The interactive approach is useful to test and compare different match strategies for specific schemas and to continuously refine and improve the match result. In automatic mode, the match process consists of a single match iteration for which a default strategy is applied or strategy specified by input parameters. This mode is especially useful for applications already knowing their most suitable match strategy or implementing their own user interaction interface.

We now describe the steps of the match process in more detail. After being converted to the internal graph format introduced above, the schemas are traversed to determine all schema elements for which the match algorithms calculate the similarity values. We represent schema elements by their *paths*, i.e. sequences of nodes following the containment links from the root to the corresponding nodes. Shared schema fragments or elements, such as *Address* in *PO2*, will result in multiple paths for which we can independently determine match candidates.

COMA supports user interaction by a so-called User-Feedback matcher to capture match and mismatch information provided by the user including corrected match results from the previous match iteration. This matcher ensures that approved matches (and mismatches) are assigned the maximal (and minimal) similarity and that these values remain unaffected by the other matchers during the matcher execution step. The user-provided similarity values influence the similarity computations for the neighbourhood of the respective elements and can thus improve the match accuracy of structural matchers.

A main step during a match iteration is the execution of multiple independent matchers chosen from the matcher library. The matchers currently supported fall into three classes: *simple*, *hybrid* and *reuse-oriented matchers*. They exploit different kinds of schema information, such as names, data types, and structural properties, or auxiliary information, such as synonym tables and previous match results. Each matcher determines an intermediate match result consisting of a similarity value between 0 and 1 for each combination of *S1* and *S2* schema elements. The result of the matcher execution phase with *k* matchers, *m S1* elements and *n S2* elements is a *k* x *m* x *n cube of similarity values*, which is stored in the repository for later combination and selection steps. Table 1 shows a sample extract from the similarity cube for the purchase order schemas of Figure 1.

Matcher	*PO1* Elements	*PO2* Elements	Sim
Type-Name	*PO1.ShipTo.shipToCity*	*PO2.DeliverTo.Address.City*	0.65
	PO1.ShipTo.shipToStreet		0.3
	PO1.Customer.custCity		0.80
Name-Path	*PO1.ShipTo.shipToCity*	*PO2.DeliverTo.Address.City*	0.78
	PO1.ShipTo.shipToStreet		0.73
	PO1.Customer.custCity		0.53

Table 1. Similarity values computed for *PO1* and *PO2*

The final step in a match iteration is to derive the combined match result from the individual matcher results stored in the similarity cube. This is achieved in two substeps: *aggregation of matcher-specific results* and *selection of match candidates*. First, for each combination of schema elements the matcher-specific similarity values are aggregated into a combined similarity value, e.g. by taking the average or maximum value. Table 2 shows the result of this step for the example of Table 1 using the average strategy. Second, we apply a selection strategy to choose the match candidates for a schema element, e.g. by selecting the elements of the other schema with the best similarity value exceeding a certain threshold. For the example in Table 2 we could thus determine

PO1 elements	*PO2* elements	Combined sim
PO1.ShipTo.shipToCity	*PO2.DeliverTo.Address.City*	0.72
PO1.Customer.custCity		0.67
PO1.ShipTo.shipToStreet		0.52

Table 2. Similarity values combined from Table 1

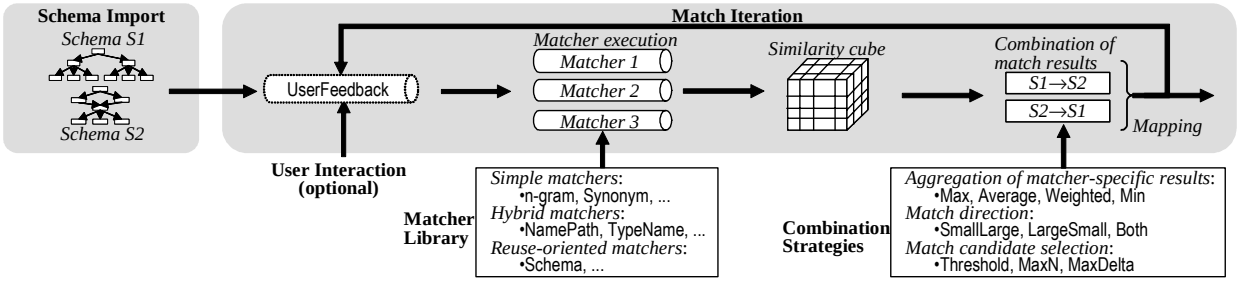


Figure 2. Match processing in COMA

PO1.ShipTo.shipToCity as the match candidate of *PO2.DeliverTo.Address.City*.

COMA supports the determination of *undirectional* or *directional* match results. In the former case, match candidates are determined for both input schemas. Moreover, an *S1* element *s1* is only accepted as a match candidate for an *S2* element *s2* if *s2* is also a match candidate of *s1*. For instance, in the above example we would accept *PO1.ShipTo.shipToCity* as the match candidate of *PO2.DeliverTo.Address.City* only if there are no better *PO2* match candidates for *PO1.ShipTo.shipToCity* than *PO2.DeliverTo.Address.City*. In the case of a directional match, the goal is to find all match candidates only with respect to one of the schemas, say *S2*. Hence, it is only tried to find match candidates for *S2* elements while accepting that *S1* elements remain unmatched. This approach has been followed by most previous studies and is motivated by the fact that many applications require such a directional match (e.g., to integrate a new data source with schema *S1* into a data warehouse or mediator with global schema *S2*). If the target schema *S2* is small compared to *S1* the match problem is substantially simplified.

4 Matcher library

Table 3 gives an overview of the matchers we have implemented and tested so far. We characterize the kinds of schema and auxiliary information they exploit. In the following we first describe the simple matchers followed by the hybrid matchers. The more complex reuse-oriented matcher Schema is discussed in Section 5.

4.1 Simple matchers

Element names represent an important source for assessing similarity between schema elements. This can be done syntactically by comparing the name strings or semantically by comparing their meanings. Approximate string matching techniques [10] have already been employed in other fields, such as record linkage [20] and data cleaning [19], to detect duplicate database records concerning the same real-word entity, i.e. matching at the instance level. In COMA, we have implemented four simple approximate string matchers:

Affix: This matcher looks for common affixes, i.e. both prefixes and suffixes, between two name strings.

n-gram: Strings are compared according to their set of *n-grams*, i.e. sequences of *n* characters, leading to different variants of this matcher, e.g. Digram (2), Trigram (3).

EditDistance: String similarity is computed from the number of edit operations necessary to transform one string to another one (the Levenshtein metric [10]).

Soundex: This matcher computes the phonetic similarity between names from their corresponding soundex codes.

Further simple matchers are UserFeedback (Section 3), a semantic matcher, Synonym, and a DataType matcher:

Synonym: This matcher estimates the similarity between element names by looking up the terminological relationships in a specified dictionary. Currently, it simply uses relationship-specific similarity values, e.g., 1.0 for a synonymy and 0.8 for a hypernymy relationship.

DataType: This matcher uses a synonym table specifying the degree of compatibility between a set of predefined generic data types, to which data types of schema elements are mapped in order to determine their similarity.

Matcher Type	Matcher	Schema Info	Auxiliary Info
Simple	*Affix*	Element names	-
	n-gram	Element names	-
	Soundex	Element names	-
	EditDistance	Element names	-
	Synonym	Element names	Extern. dictionaries
	DataType	Data types	Data type compatibility table
	UserFeedback	-	User-specified (mis-) matches
Hybrid	*Name*	Element names	-
	NamePath	Names+Paths	-
	TypeName	Data types+Names	-
	Children	Child elements	-
	Leaves	Leaf elements	-
Reuse-oriented	*Schema*	-	Existing schema-level match results

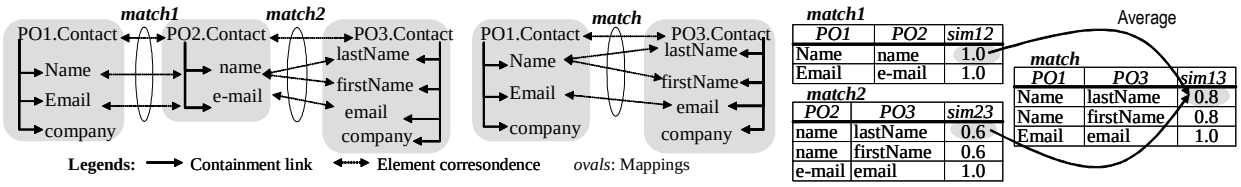
Table 3. Implemented matchers in the matcher library

4.2 Hybrid matchers

The hybrid matchers use a fixed combination of simple matchers and other hybrid matchers to obtain more accurate similarity values. The approach applied for combining the results of the constituent matchers follows the same principles used for combining the matcher results in the final phase of the match process (or iteration). The details of how matchers are combined within a hybrid matcher are explained in Section 6.

We currently support two hybrid element-level matchers, Name and TypeName, and three hybrid structural matchers, NamePath, Children and Leaves. All approaches rely to different degrees on similarities derived from element names for which combinations of the simple matchers discussed above can be utilized (e.g. Synonym, etc.).

Name: This matcher only considers the element names but is a hybrid approach because it combines different



a) ***match1***: *PO1*↔*PO2* and ***match2***: *PO2*↔*PO3* **b) *match***=MatchCompose(***match1, match2***) **c)** relational representation for MatchCompose

Figure 3. MatchCompose example

simple name matchers. It performs some pre-processing steps, in particular a tokenization to derive a set of components (tokens) of a name, e.g. *POShipTo* → {*PO*, *Ship*, *To*}. Moreover it expands abbreviations and acronyms, e.g. *PO* → {*Purchase*, *Order*}. The Name matcher then applies multiple simple matchers, such as Affix, Trigram, and Synonym, on the token sets of the names and combines the obtained similarity values for tokens to derive similarity values between element names (see Section 6).

NamePath: This matcher matches elements based on their hierarchical names, i.e. both structural aspects and element names are considered. It first builds a long name by concatenating all names of the elements in a path to a single string. It then applies Name to compute the similarity between these long names. Considering the complete name path of an element provides additional tokens for name matching which may improve match accuracy. For instance, this can be helpful to find match candidates at different schema levels, e.g. *PurchaseOrder.ShipTo.Street* and *PurchaseOrder.shipToStreet*. On the other hand, it is possible to distinguish between different contexts of the same element, e.g. *ShipTo.Street* and *BillTo.Street*.

TypeName: This element matcher combines the DataType and Name matcher, i.e. it matches elements based on a combination of their name and data type similarity.

Children: This structural matcher is used in combination with a leaf-level matcher. It determines the similarity between two inner elements based on the combined similarity between their child elements, which in turn can be both inner and leaf elements. The similarity between the inner elements needs to be recursively computed from the similarity between their respective children. The similarity between the leaf elements is obtained from the leaf-level matcher, for which TypeName is used as the default.

Leaves: This structural matcher is also used in combination with a leaf-level matcher, for which TypeName is set as the default. In contrast to the Children strategy, this matcher only considers the leaf elements to estimate the similarity between two inner elements. This strategy aims at more stable similarity in cases of structural conflicts. In Figure 1, for example, elements *shipToStreet*, *shipToCity*, etc., are children of *ShipTo* in *PO1*, while in *PO2*, their matching elements are not children of *DeliverTo*, but of *Address*. Children will therefore only find a correspondence between *ShipTo* and *Address*, while Leaves can also identify a correspondence between *ShipTo* and *DeliverTo*.

5 Reuse of previous match results

The consideration of reuse-oriented matchers is motivated by our expectation that many schemas to be matched are similar (or identical) to previously matched schemas. The use of auxiliary information such as synonym dictionaries, thesauri, already represents such a reuse-oriented approach utilizing confirmed correspondences at the level of schema elements (names or data types). Our goal is to generalize this idea and reuse multiple match correspondences at the same time at the levels of schema fragments or entire schemas.

As a first step, we have implemented two simple reuse-oriented matchers that can be invoked and combined like other matchers. One of them, Schema, tries to reuse match results for entire schemas, the other, Fragment, operates on schema fragments. In both cases we use a special compose operation, MatchCompose, to derive a new match result from existing ones. We first introduce MatchCompose. Due to lack of space, we then only describe Schema.

5.1 The MatchCompose operation

Given two match results, *match1*: *S1*↔*S2* and *match2*: *S2*↔*S3* sharing schema *S2*, MatchCompose derives a new match result, *match*: *S1*↔*S3*, between *S1* and *S3*. The operation assumes a transitive nature of the similarity relation between elements, i.e. if *a* is similar to *b* and *b* to *c*, then *a* is (very likely) also similar to *c*. Of course wrong match candidates may be determined in cases where the transitivity property does not hold.

In the context of information retrieval, transitive similarity estimations have been applied to derive the similarity of words based on terminological relationships, such as synonymy and hypernymy [4, 17]. A common approach to determine the transitive similarity is to multiply the individual similarity values [2]. This approach, however, may lead to rapidly degrading similarity values. For instance, for

$$contactFirstName \xleftrightarrow{0.5} Name \xleftrightarrow{0.7} firstName$$

the similarity between *contactFirstName* and *firstName* would become 0.5*0.7=0.35, which is unlikely to reflect the similarity, which we would expect for the two names. We thus prefer the alternatives for combining the results of different matchers, such as Average (Section 6.1), for calculating transitive similarities, resulting in similarity value 0.6 in the last example.

Figure 3a and b illustrate the approach for the match *PO1*↔*PO3* derived from composing the two match re-

sults *match1*: *PO1*↔*PO2* and *match2*: *PO2*↔*PO3*. To efficiently calculate the MatchCompose result, *match*, we use a relational representation for the input match results. Figure 3c shows the tables representing *match1*, *match2* and *match*. In these tables each tuple specifies a 1:1 correspondence between elements of the respective schemas together with their similarity. MatchCompose then corresponds to the natural join between the two input tables.

The example in Figure 3 also shows that MachCompose and thus Schema may miss some correspondences, e.g. between *company* of *PO1* and *PO3*, due to the absence of a match counterpart in *PO2*. Furthermore, MatchCompose may return undesirable correspondences when elements of the "intermediate" schema are related to several elements of the other schemas to be matched. In Figure 4, the composition of two mappings returns all possible matches, i.e. *ShipTo.Contact* is matched to both *DeliverTo.Contact* and *InvoiceTo.Contact*, while only the former match is likely to be correct. However, these negative effects can be limited, as shown in our evaluation in Section 7, by combining MatchCompose results with the results of other matchers.

Figure 4. MatchCompose with undesirable m:n matches

5.2 The Schema reuse matcher

Figure 5 illustrates the schema-level reuse approach implemented in the Schema matcher. All previous match results are maintained within the repository and can be exploited for reuse. Given two schemas *S1* and *S2* to match, Schema identifies all schemas *S*, i.e. *Si*, *Sj*, *Sk*, for which there is a pair of match results relating *S* with both *S1* and *S2* in any order. For each such pair MatchCompose is applied to produce a *S1*↔*S2* match result. If there are multiple such results, they in turn can be combined using any strategy for aggregation (e.g., Average) and selection (Section 6), and the combined result is stored in the similarity cube for further processing in the match process.

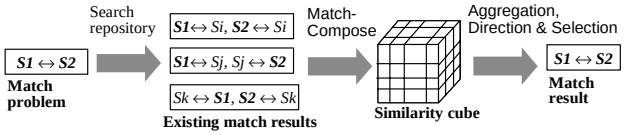


Figure 5. Schema-level reuse in the Schema matcher

Despite the high level of reuse in Schema (schema level), we believe that there is a high probability to find the necessary match result pairs for MatchCompose in an environment where many schemas are managed and matched to each other. Furthermore, schemas from the same application domain usually contain many similar elements, which are typical to this domain, so that their mappings can provide good reusable candidates.

6 Combination of similarity values

In this section we describe how similarity results from different matchers are combined in COMA to derive a combined match result. Such a combination of similarity values is used in two main cases: within the implementation of our hybrid matchers to combine the results of the constituent matchers and in the final step of a match process (or iteration) to combine the results of independent matchers to obtain a complete match result. Both cases are implemented by a series of aggregation and selection operations on the similarity cube containing the similarity values calculated by a set of matchers *M* (Figure 6). To determine the complete match result for two input schemas two main steps are needed; step 3 is optional:

- *Aggregation of matcher-specific results*: In the first substep, similarity values computed by multiple matchers are aggregated to a combined similarity value for each pair of schema elements. With *m S1* elements and *n S2* elements we obtain an *m* x *n* matrix of combined similarity values.
- *Selection of match candidates*: To determine the best match candidate(s), we rank the correspondences according to their similarity values per schema element and apply a filter strategy to determine the most plausible ones. The result of this step is a combined match result with 0, 1 or more match candidates per schema element. In the case of an undirectional match, the match candidates for both schemas are determined.
- *Computation of combined similarity*: The match result from the previous step can be aggregated into a single similarity value for the two schemas, called *schema similarity*. It depends on the chosen matchers and their combination strategy.

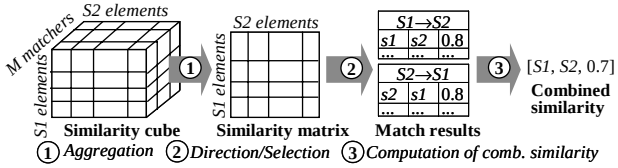


Figure 6. Combination of match results

These three steps are also needed for hybrid matchers to combine the similarity values for its constituent matchers. However, in this case these steps are not applied to similarity values of schema elements but to similarity values for the components of schema elements. For instance, a name matcher determines the similarity of names from the similarities of the name tokens, and a structural matcher can derive the similarity of inner nodes from the similarity values of their children or leaves. As a result, the sets *S1* and *S2* in Figure 6 for which similarity values are processed now refer to these components of schema objects. For hybrid matchers, these similarity values of the components can be determined by different matchers resulting in a similarity cube which has to be aggregated. Now, the third step is no longer optional but required to derive a single similarity value, the *element similarity*, for a pair of schema objects (names, inner nodes) by combining the similarity values of the match candidates determined in step 2.

To sum up, we use steps 1 and 2 for combining similarity values to obtain the complete match result. For hybrid matchers we need the additional step 3. In the following we present the approaches for these steps that COMA

currently supports; additional approaches can easily be added. We finally discuss the default combinations used for the various hybrid matchers.

6.1 Aggregation of matcher-specific results

One of the following strategies can be used to aggregate matcher-specific similarity values for every element pair:

1. **Max:** This strategy returns the maximal similarity value of any matcher. It is optimistic, in particular in case of contradicting similarity values. Furthermore, matchers can maximally complement each other.
2. **Weighted:** This strategy determines a weighted sum of similarity values of the individual matchers and needs relative weights which should correspond to the expected importance of the matchers.
3. **Average:** This strategy represents a special case of Weighted and returns the average similarity over all matchers, i.e. considers them equally important.
4. **Min:** This strategy chooses the lowest similarity value of any matcher. As opposed to Max, it is pessimistic.

6.2 Direction and selection of match candidates

As discussed in Section 3, COMA supports determination of directional or undirectional match results. Given two schemas *S1* and *S2* with $|S2| \leq |S1|$, match candidate selection can be performed in the following directions:

1. **LargeSmall:** In this directional approach, we match the larger schema *S1* against the smaller target *S2*, i.e. elements from *S1* are ranked and selected with respect to each *S2* element.
2. **SmallLarge:** As opposed to LargeSmall, match candidate selection is performed based on ranking *S2* elements for each *S1* element.
3. **Both:** This strategy considers the results from both match directions, LargeSmall and SmallLarge. Furthermore, an *S1* and an *S2* element are only accepted as a matching pair if it is identified as such in both directions.

To determine the match candidates from *S1* for an element *s2* in *S2* we use the similarity matrix to rank the *S1* correspondences in descending order of their similarity value. For selecting the match candidates one of the following strategies can be used:

1. **MaxN:** The *n S1* elements with maximal similarity are selected as match candidates. *n*=1, i.e. Max1, represents the natural choice for 1:1 correspondences. Generally, *n*>1 is useful in interactive mode to allow the user to select among several match candidates.
2. **MaxDelta:** The *S1* element with maximal similarity is determined as match candidate plus all *S1* elements with a similarity differing at most by a tolerance value *d*, which can be specified either as an absolute or relative value. The idea is to return multiple match candidates when there are several *S1* elements with the same or almost the same similarity value.
3. **Threshold:** All *S1* elements showing a similarity exceeding a given threshold value *t* are selected.

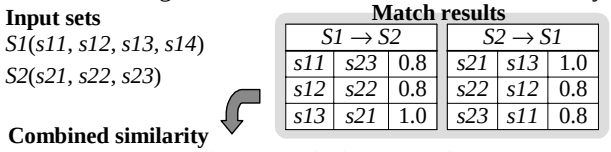
A single approach may return imprecise match candidates. While Threshold may return too many match candidates, MaxN and MaxDelta may return match candidates with too little similarity. Thus, we support considering several criteria at the same time, in particular MaxN or MaxDelta in combination with a low threshold, e.g. 0.5.

6.3 Computation of combined similarity

As discussed above, hybrid matchers require an additional step to obtain a combined similarity value for sets of element components. For this purpose we support two strategies, namely Average and Dice. They work on the output of step 2 consisting of a list of match candidates for sets *S1* and *S2*. Assuming at most one match candidate per *S1* and *S2* element we determine the combined similarity as follows:

1. **Average:** The average similarity is determined by dividing the sum of the similarity values of all match candidates of both sets *S1* and *S2* by the total number of set elements, $|S1|+|S2|$.
2. **Dice:** This strategy is based on the Dice coefficient [6] and returns the ratio of the number of elements which can be matched over the total number of set elements.

Figure 7 illustrates the two approaches. Unlike Average, the individual similarity values in Dice do not influence the overall similarity of the sets. Hence, Dice returns a higher similarity value than Average and thus is more optimistic. Average and Dice can also be applied to manually derived match results to compute the similarity between two schemas. With all element similarities set to 1.0, both strategies will return the same schema similarity.



Combined similarity

- Average $(S1, S2) = \frac{(0.8+0.8+1.0)+(1.0+0.8+0.8)}{4+3} = 0.74$
- Dice $(S1, S2) = \frac{3+3}{4+3} = 0.86$

Figure 7. Examples for computing combined similarity

6.4 Construction of hybrid matchers

To determine the combination strategy for a hybrid matcher, a tuple of 4 sub-strategies is to be specified, e.g. (Max, Both, Max1, Average), one for each step in our combination scheme. Table 4 shows the default constituent matchers and combination strategies used in our hybrid matchers to compute similarity values for single pairs of schema elements. While the Name matcher covers all three steps, other approaches either require only the first step (TypeName) or the last two steps (Children, Leaves). Note that any strategy specified for computing combined similarity will presuppose Both as direction strategy in step 2.

Hybrid matcher	Default matchers	Default combination strategy		
		(1) Aggreg.	(2) Direct. & Select.	(3) Comb. sim
Name	*Trigram, Synonym*	*Max*	*Both, Max1*	*Average*
TypeName	*DataType, Name*	*Weighted* (0.3, 0.7)	-	-
Children	*TypeName*	-	*Both, Max1*	*Average*
Leaves	*TypeName*	-	*Both, Max1*	*Average*

Table 4. Construction of hybrid matchers

Name computes element similarities by combining the similarity values for the names' token sets. Token similarities are determined using multiple simple matchers, such as Trigram and Synonym. In step 1, we use Max for

aggregating the matcher-specific similarity values from the cube, motivated by the fact that tokens are typically similar according to only some simple matchers. For example, string matchers such as Trigram find no similarity for *Ship* and *Deliver*, while a semantic matcher such as Synonym can detect the synonymy and assign a high similarity value. In step 2, we consider both directions and apply Max1 to the similarity matrix to obtain two sets of directional token correspondences. Finally, in step 3, the name similarity between the token sets is then computed using the Average strategy.

TypeName combines DataType and Name, each of which produces a single similarity value for a pair of schema elements. For step 1, we use the Weighted strategy. Steps 2 and 3 are not needed because we already have a single similarity value after step 1. The default weights of the name and data type similarity, 0.7 and 0.3, respectively, permit to match attributes with similar names but different data types. When several attributes exhibit about the same name similarity, candidates with higher data type compatibility are preferred.

In Children and Leaves we compare two sets of elements, which are either the children or leaves of two inner elements. Thus steps 2 and 3 are necessary. Because only one matcher is used for determining the leaf similarities, we do not need step 1 but directly obtain the similarity matrix from the results of the single leaf matcher.

7 Evaluation on real world schemas

We performed a comprehensive evaluation of the match processing strategies supported by COMA on several complex real world schemas. The main goal was to investigate the impact of different combination strategies, i.e. aggregation, direction, selection, computation of combined similarity, on match quality, and to compare the effectiveness of different matchers, i.e. single matchers and their combinations, with and without reuse. We first describe the design and methodology of our evaluation. We then present the results for the combination strategies, the single matchers and matcher combinations, and the match sensitivity in different match tasks.

7.1 Experimental design

For our evaluation we used 5 XML schemas for purchase orders, CIDX, Excel, Noris, Paragon, and Apertum, taken from www.biztalk.org. For short, we refer to them as 1, 2, etc., respectively. Table 5 summarizes the characteristics about the test schemas. Except for schema 1, the number of paths is different from the number of nodes, indicating the use of shared fragments in the schemas. Previous match studies mostly used smaller schemas [9, 13]. The size of schemas can impact match accuracy because it determines the search space for match candidates.

Schemas	1	2	3	4	5
Max depth	4	4	4	6	5
#Nodes / paths	40 / 40	35 / 54	46 / 65	74 / 80	80 / 145
#Inner nodes / paths	7 / 7	9 / 12	8 / 11	11 / 12	23 / 29
#Leaf nodes / paths	33 / 33	26 / 42	38 / 54	63 / 68	57 / 116

Table 5. Characteristics of test schemas

We defined 10 match tasks, each matching two different schemas. To provide a basis for evaluating the quality of different automatic match strategies we first manually performed the match tasks. Figure 8 gives an impression about the problem size in each match task. *#Matches* indicates the number of correspondences to be identified. The similarity between the schemas (*Schema Similarity*) is computed using the Dice strategy (Section 6.3) as the ratio between *#Matched Paths* and *#All Paths*.[1] This similarity is mostly around 0.5, showing that the schemas are much different even though they are from the same domain.

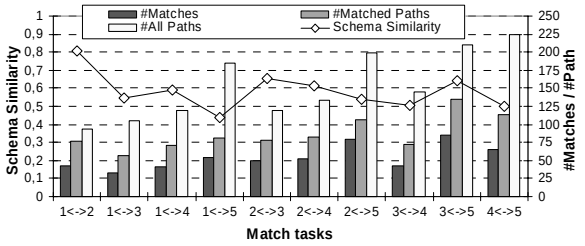


Figure 8. Problem size in schema matching tasks

Measures for match quality: To evaluate the quality of our match operations, we compare the manually determined real matches (*R*) for a match task with the matches *P* returned by automatic match processing. We determine the true positives, i.e. correctly identified matches, *I*, as well as the false positives, i.e. false matches, *F=P\I*, and the false negatives, i.e. missed matches, *M=R\I*. Based on the cardinalities of these sets, the following quality measures are computed:

- $Precision = \frac{|I|}{|P|} = \frac{|I|}{|I|+|F|}$ estimates the reliability of the match predictions
- $Recall = \frac{|I|}{|R|}$ specifies the share of real matches that is found
- $Overall = 1 - \frac{|F|+|M|}{|R|} = \frac{|I|-|F|}{|R|} = Recall * \left(2 - \frac{1}{Precision}\right)$ represents a combined measure for match quality [13], taking into account the post-match effort needed for both removing false and adding missed matches.

Precision and Recall originate from the information retrieval field and have also been used in other match studies [11, 1][2]. Note that neither Precision nor Recall alone can accurately assess the match quality. Recall can easily be maximized at the expense of a poor Precision by returning all possible correspondences, i.e. the cross product of two input schemas. Similarly, a high Precision can be achieved at the expense of a poor Recall by returning only few (correct) correspondences. On the other side, Overall proportionally depends on both Recall and Precision and thus represents a single metric suitable for comparing match quality. Unlike Precision and Recall, Over-

[1] In our manually derived match results, all element similarities are set to 1.0 so that Average and Dice yield the same schema similarity

[2] In [1] they are called soundness and completeness, respectively

all can have negative values if the number of the false positives exceeds the number of the true positives, i.e. Precision<0.5. In this case, the post-match effort can be regarded as higher than the gain from the automatic match operation, which is then rather useless. In the ideal case, when all and only the real matches are returned, i.e. $I=P=R$, $F=M=\emptyset$, the measures reach their highest values Precision=Recall=Overall=1. In all other cases, Overall is smaller than both Precision and Recall making it difficult to achieve higher values than, say, 0.5.

Experiment methodology and execution: To determine the effectiveness of the match operations we used COMA only in automatic mode in this study, i.e. we did not consider possible improvements by user feedback and iterative refinements of the generated match result.

Matchers		Aggregat.	Direction	Selection	CombSim
No reuse	5 single		-LargeSmall -SmallLarge -Both	-MaxN(1-4) -Delta(0.01-0.1) -Thr(0.3-1.0) -Thr(0.5)+ MaxN(1-4) -Thr(0.5)+ Delta(0.01-0.1)	-Average -Dice
	11 combinations	-Max -Average -Min			
Reuse	2 single				
	12 combinations	-Max -Average -Min			-Average
Σ = 16 + 14		3	3	36	2

Table 6. Tested matchers and combination strategies

We performed a systematic evaluation for all relevant matchers and combination strategies (see Table 6). Our evaluation encompassed a number of *series*, in each of which we applied a different choice of matchers and combination strategies. Altogether, we conducted 12,312 series. Each series in turn consisted of 10 *experiments* dealing with the (10) predefined match tasks. The quality measures were first determined for single experiments and then averaged over all experiments in each series (hereafter average Precision, etc.).

Due to their limited focus, the simple matchers alone do not perform well on real world schemas. Therefore, we only tested the more powerful hybrid matchers and their combinations. Each matcher was tested with all possible combinations of the strategies for aggregation, direction, selection, and computation of combined similarity, whereby the different strategies for aggregation and for computation of combined similarity are not relevant for the single matchers and the single reuse matchers, respectively. The Weighted aggregation strategy was not considered because we did not want to make any assumption about the importance of the individual matchers. For each selection strategy, we chose a generous parameter range, in which the best result is to be expected. In particular, we tested MaxN with up to 4 candidates, MaxDelta (or short Delta) with relative deltas between 0.01 and 0.1 (i.e., 1-10%), Threshold (or short Thr) with threshold values between 0.3 and 1.0. Furthermore, we tested the combinations of MaxN and MaxDelta, respectively, with Threshold using a low threshold of 0.5.

For semantic name matching in the Synonym matcher, we constructed a synonym file with some trivial abbreviations, such as, *No*, *Num*, and domain-specific synonyms, such as (*ship*, *deliver*), (*bill*, *invoice*). This auxiliary information was used uniformly in all experiments.

7.2 Combination strategies

We first performed series with the 16 no-reuse matchers: the 5 single hybrid matchers, all their pair-wise combinations, and the combination of all 5 hybrid matchers (hereafter All). In the no-reuse case the best average Overall obtained for the 10 match tasks is 0.73, and the worst is -88.0, i.e. Precision is almost 0 and Recall almost 1. Figure 9 shows the distribution of the series with respect to different Overall ranges. The x-axis covers the entire value range of Overall. The distribution of the series with average Overall>0 is shown in successive ranges 0.0-0.1, ..., 0.7-0.8, while the series with average Overall<0 are aggregated in one single range Min-0.0. Most series have negative average Overall, indicating poor matchers and/or combination strategies. Only 3 series yield average Overall in the range of 0.7-0.8.

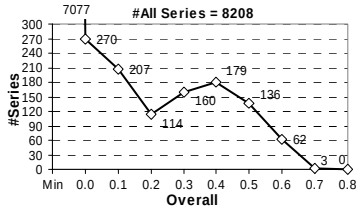


Figure 9. Distribution of series in different Overall ranges

In order to identify the best matchers and combination strategies, we concentrate on the best series having average Overall>0. Figure 10 shows the share of the series belonging to single strategies of aggregation, direction and selection, respectively, with respect to all series encountered in each Overall range. Note that in each diagram, the number of the series belonging to the single strategies stays equal due to the exhaustive evaluation. For example in Figure 10a, over the entire x-axis, we have #Max series = #Min series = #Average series = 2376 despite their unproportional appearance. Hence, the distribution of the series indicates the quality of the respective strategy: a strategy of high quality should show a high presence in higher Overall ranges. We now discuss the quality of the single combination strategies.

- **Aggregation:** Max is only represented in Overall ranges below 0.1. This is because of, first, its optimistic nature, and second, the inaccuracy of several hybrid matchers (Section 7.3). In such cases, the pessimistic Min and compensating strategy Average are more stable. In particular, series with Min and Average are equally represented in the Overall range 0.0-0.6; however only Average yields greater average Overall. Thus, while Max is only suitable for combining accurate matchers, Min and Average are also able to cope with inaccurate ones.
- **Direction:** SmallLarge is only represented in Overall ranges below 0.3, while LargeSmall can reach average Overall of up to 0.6. This shows the impact of the different size between the source and the target schema on the quality of directional matching. SmallLarge tries to find match candidates for the larger target schema from the smaller source schema, resulting in a larger match

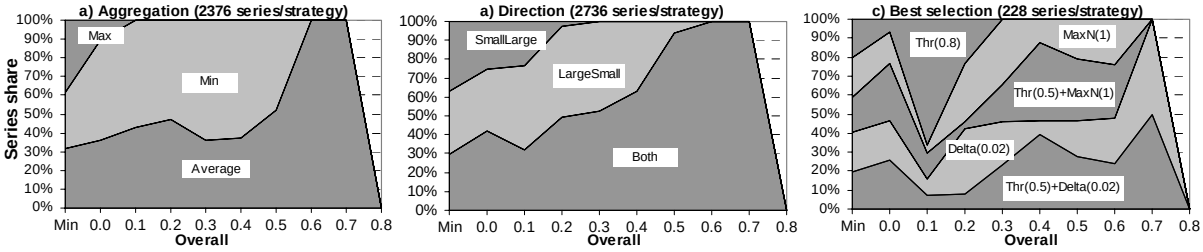


Figure 10. Distribution of series with respect to combination strategies

result apparently with a greater inaccuracy than for LargeSmall. On the other side, Both can produce much better results than both LargeSmall and SmallLarge and is less dependent on the size of the input schemas.

- **Selection:** From the many selection strategies tested, we first determined the most effective parameter settings for each approach. Figure 10c shows the share of series for these best selection strategies. Surprisingly, the Threshold approach, which is often used in previous work, shows the worst result. The average Overall values for its best variant Threshold(0.8) are always below 0.3. MaxN(1) and Threshold(0.5)+MaxN(1) can produce average Overall of up to 0.7. Most successful are Delta(0.02) and Threshold(0.5)+Delta(0.02) with average Overall beyond 0.7.
- **Computation of combined similarity**:[3] We observe in general some degradation of match quality using Dice, compared to Average, for computing combined similarity in the hybrid matchers. The best average Overall of Dice and Average is 0.67 and 0.73, respectively.

The results show that in the no-reuse cases the best single strategies for aggregation, direction, selection and computation of combined similarity are Average, Both, Threshold(0.5)+Delta(0.02) or Delta(0.02), and Average, respectively. To verify whether the combined use of these strategies can also yield the best match results we identify the best series for the matcher combinations (10 pair-wise plus All) and examine their combination strategies used. Among these 11 combinations, 3 do not produce any positive average Overall and are ignored. We observe that

- all 8 remaining combinations achieve their best result with Average for aggregation and Both for direction
- all 8 employ a combined selection strategy of Threshold(0.5) and a Delta strategy: 4 use Delta(0.02) and 4 remaining utilize different delta values (0.03 and 0.04)
- 7 use Average for computing combined similarity

Thus we choose (Average, Both, Threshold(0.5)+Delta(0.02)) as the default combination strategies, while Average is set as the default strategy for computing combined similarity in the hybrid matchers. Using these default strategies, All (among 8 above) achieves the highest average Overall, 0.73, in the no-reuse test and thus is chosen as the default matcher combination.

7.3 Single matchers and matcher combinations

For all match tasks, we used the default match operation to automatically derive the corresponding match results, which were stored in the repository in addition to the manually derived match results. We then performed further series for 14 reuse strategies: two variants of the Schema matcher called SchemaM and SchemaA, their pair-wise combinations with the 5 single hybrid matchers, and their combination with all hybrid matchers (hereafter named All+Schema). While SchemaM reuses the *manually* determined match results of different match tasks, SchemaA is based on the reuse of the *automatically* derived match results of the default match operation. Considering these two cases illustrates the improvements possible by reusing manually confirmed match results over unconfirmed results. For every match task, say 1↔3, SchemaM and SchemaA apply MatchCompose to 3 pairs of match results from *other match tasks*, namely (1↔2, 2↔3), (1↔4, 3↔4), and (1↔5, 3↔5).

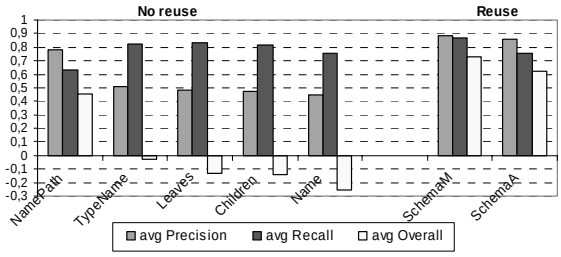


Figure 11. Quality of single matchers

Figure 11 shows the quality of the single matchers, distinguished between the no-reuse and reuse-oriented ones and sorted on their average Overall. We make the following observations about the quality of the hybrid matchers:

- As expected, redundancy in schemas, i.e. shared elements, causes instability of some matchers, such as Name, TypeName, Children, Leaves. These matchers are not able to distinguish between different element contexts, resulting in a high number of false positives in several match tasks (negative values of Overall).
- NamePath shows a better quality than Name and is among the no-reuse matchers the best one in terms of Precision and Overall. Using hierarchical names represents an effective method for distinguishing between different contexts of a shared element. However, it is also more restrictive in predicting match candidates than considering single element names. Usually, NamePath results in lower Recall but higher Precision than Name.
- TypeName shows a slightly better quality than Name, indicating that incorporating data type information can be valuable. For the considered PO schemas the improvements are small because most leaf elements in our test schemas are either of type String or Number.
- TypeName, Leaves and Children show approximately the same Recall, because first, both Leaves and Children use

[3] Due to lack of space, we do not present a figure for this comparison.

TypeName as the leaf matcher, and second the number of inner element matches is much lower than that of leaf matches. Leaves yields a slightly better quality than Children for the PO schemas.

The Schema reuse matchers show the best quality among the single matchers. While the best no-reuse matcher, NamePath, achieved an average Overall of only 0.45, SchemaA and SchemaM achieved significantly better average Overall values of 0.62 and 0.73, respectively. This is because both the automatically and manually derived match results have provided many candidates for reuse. As expected, the manually derived match results allow for a substantial improvement over the reuse of automatically determined match results. Compared to the no-reuse schemes, SchemaA and SchemaM achieve only slightly better Recall but significantly better average Precision of 0.85 and 0.88, respectively. This shows that the problem of false n:m matches has been compensated to a great extent by combining multiple MatchCompose results.

We now discuss the quality of different matcher combinations considering both no-reuse and reuse matchers. When evaluating the combination of the reuse schemes with other matchers, we noted that the above identified strategies that worked best for combining no-reuse matchers were also effective for SchemaA but not so much for SchemaM. Instead, SchemaM combinations mostly achieve their best quality with combination strategies (Min, LargeSmall, Delta(0.1), Average). This apparently traces back to the fact that the similarity is uniformly set to 1.0 for all element correspondences in the manually derived match results. We will conduct further research in order to understand this behavior in more detail.

Figure 12 shows the quality of the best matcher combinations sorted on their average Overall. Due to lack of space we only present the results for SchemaM for the reuse combinations. We observe that the matcher combinations in general allow much better match quality than the single hybrid matchers. The best no-reuse combination, All, achieves average Overall of 0.73, the best reuse combination, All+SchemaM, reaches the best average Overall observed in our entire evaluation, 0.82. This is because the combined matchers inherit the high stability of NamePath and SchemaM, while eliminating many false matches of Name, TypeName, Leaves, or Children.

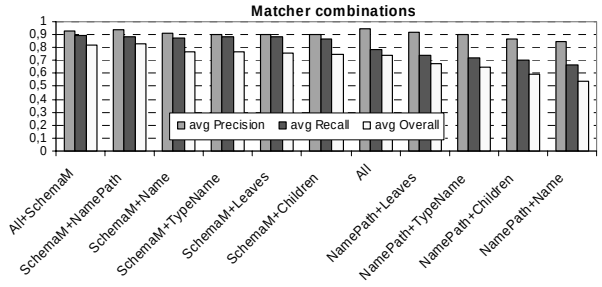


Figure 12. Quality of best matcher combinations

Among the no-reuse combinations, All performs best because many aspects are examined at the same time to contribute to the overall similarity between elements. NamePath+Leaves also represents an effective no-reuse combination with an improvement of 20% and 80% of average Overall compared to NamePath and Leaves, respectively. The combined scheme exploits three kinds of schema information, element names, data types, and structural information, in an intelligent way: examining paths to identify the context of shared elements while considering leaves to cope with structural conflicts. Because of the restrictiveness of NamePath in match prediction, its combinations achieve very high Precision. Combinations with Leaves yield better quality that those with Children.

The reuse combinations are always better than the no-reuse combinations. They exhibit only small differences in match quality with very high average Precision (>0.90). Their maximal average Recall observed is 0.89, indicating that previous match results cannot always provide all reuse candidates required for a new match task.

7.4 Match sensitivity

Besides the matchers and combination strategies, external factors, such as schema characteristics, can also impact match quality. Figure 13 shows the relationship between schema size, schema similarity and the best Overall achieved using any no-reuse and (manual) reuse strategy for each match task. Again, the reuse approaches clearly outperform the no-reuse approaches. However, we observe that, regardless of the match strategy used, match quality usually degrades with an increase of schema size. Several reuse strategies achieve optimal or close to optimal Overall values in the smaller match tasks but are limited to an Overall value of 0.7 for the larger problems. Intuitively, a match task of the same size becomes "harder", if the schema similarity drops. For example, quality of automatic matching degrades substantially from task 3↔5 to 4↔5, and from 2↔4 to 3↔4 due to, on the on side, increase of schema size, and on the other side, decrease of schema similarity.

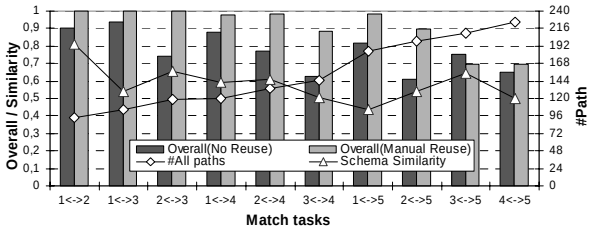


Figure 13. Impact of schema characteristics on match quality

To analyze the stability of the matchers across the match tasks, we identify for each match task the hybrid matcher(s) / combination(s) producing the match result with those maximal Overall values. This is done for reuse and no-reuse strategies, respectively. We observe that All and All+Schema outperform all hybrid matchers and other combinations and yield the best Overall for 5 and 6, respectively, out of 10 match tasks, while showing some minor degradation of at most 10% in the remaining tasks.

Compared to other matchers, All and All+Schema show thus the highest stability across our match tasks.

7.5 Evaluation conclusions

We have systematically investigated and compared the effectiveness of different matchers and combination strategies. We were able to determine a very effective default combination strategy for aggregating matcher-specific results and selecting match candidates. Average proved to be the aggregation method of choice as it could best compensate shortcomings of individual matchers. Our undirectional match approach Both supports very good precision and thus produced usually better match results than directional approaches. Most accurate match predictions can be achieved by selecting match candidates showing the (approximately) highest similarity exceeding a minimal threshold.

We observe that the composite approaches are very effective. Although single matchers may be imprecise, their combination can effectively improve the match quality. In contrast to single matchers, matcher combinations simultaneously analyze schema elements under different aspects, resulting in more stable and accurate similarity for heterogeneous schemas. The stable behavior of the default combination strategy indicates that it can be used for many match tasks thereby limiting the tuning effort.

Despite its simplicity the reuse approach proved to be very successful. The Schema approaches achieve the best quality among the single matchers, showing that the mappings between schemas from the same application domain can provide good candidates for reuse. Furthermore, the reuse-oriented matcher combinations are the best strategies in our entire evaluation and improved the match quality by more than 10% over the best no-reuse approaches.

The best matchers achieve average Precision of 95%, Recall of 80%, and Overall of 70% or better, representing a substantial saving in manual matching effort. For small match tasks, Overall values of close to 1.0 are achieved but for larger match problems, Overall was limited to about 0.6 – 0.7. While this was influenced by a moderate degree of schema similarity, we see potential for improvement by adding further matchers, e.g. those exploiting instance-level data and reusing large-scale dictionaries and standard ontologies. We also want to experiment with more comprehensive strategies for match candidate selection, such as the stable marriage approach [13].

8 Summary and future work

In this paper, we presented the generic schema match system COMA, which provides an extensible library of simple and hybrid match algorithms and supports a powerful framework for combining match results. The user can tailor match strategies by selecting the matchers and their combination for a given match problem. Hybrid matchers can also be configured easily by combining existing matchers using the provided combination strategies. We have developed a novel matcher based on a special compose operation for reusing previous match results. For a flexible combination of independent matchers we store all similarity values determined by matchers within similarity cubes of a DBMS-based repository. COMA can be used in automatic mode or interactively in order to provide user feedback and to continuously improve the match result.

We used COMA to systematically evaluate different aspects of match processing, i.e. aggregation of matcher-specific results, match direction, match candidate selection, and computation of combined similarity, and different matcher usages, i.e. single matchers vs. matcher combinations, no-reuse vs. reuse approaches. We believe that our evaluation insights can be of valuable help for the development and evaluation of further match algorithms.

In future work, we plan to add other match and combination algorithms in order to improve match quality. Furthermore, we will apply COMA to additional schema types and applications, such as in the bioinformatics domain.

Acknowledgements

We thank Sergey Melnik and the anonymous reviewers for many useful comments. This work is supported by DFG grant BIZ 6/1-1.

References

1. Berlin, J., A. Motro: Autoplex: Automated Discovery of Content for Virtual Databases. CoopIS 2001, 108-122
2. Bergamaschi, S., S. Castano, M. Vincini, D. Beneventano: Semantic Integration of Heterogeneous Information Sources. Data & Knowledge Engineering 36: 3, 215–249, 2001
3. Bernstein, P.A., A. Halevy, R. A. Pottinger: A Vision for Management of Complex Models. SIGMOD Record 29: 4, 55–63, 2000
4. Bright, M.W. et al: Automated Resolution of Semantic Heterogeneity in Multidatabase. ACM Trans. Database Systems 19: 2, 1994
5. Castano, S., V. De Antonellis: A Schema Analysis and Reconciliation Tool Environment. IDEAS 1999, 53-62
6. Castano, S, V. De Antonellis, M.G. Fugini, B. Pernici: Conceptual Schema Analysis: Techniques and Applications. ACM Trans. Database Systems 23: 3, 286-333, 1998
7. Doan, A.H., P. Domingos, A. Halevy: Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. SIGMOD 2001
8. Doan, A.H., J. Madhavan, P. Domingos, A. Halevy: Learning to Map between Ontologies on the Semantic Web. WWW 2002
9. Embley, D.W. et al.: Multifaceted Exploitation of Metadata for Attribute Match Discovery in Information Integration. WIIW 2001
10. Hall, P., G. Dowling: Approximate String Matching. Computing Survey 12: 4, 381-402, 1980
11. Li, W., C. Clifton: SemInt: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Network. Data and Knowledge Engineering 33: 1, 49-84, 2000
12. Madhavan, J., P.A. Bernstein, E. Rahm: Generic Schema Matching with Cupid. VLDB 2001
13. Melnik, S., H. Garcia-Molina, E. Rahm: Similarity Flooding: A Versatile Graph Matching Algorithm. ICDE 2002
14. Miller, R.J. et al.: The Clio Project: Managing Heterogeneity. SIGMOD Record 30:1, 78-83, 2001
15. Milo, T., S. Zohar: Using Schema Matching to Simplify Heterogeneous Data Translation. VLDB 1998, 122-133
16. Palopoli, L., G. Terracina, D. Ursino: The System DIKE: Towards the Semi-Automatic Synthesis of Cooperative Information Systems and Data Warehouses. ADBIS-DASFAA 2000, 108–117
17. Rada, R. et al.: Development and Application of a Metric on Semantic Nets. IEEE Trans. Systems, Man, and Cybernetics 19: 1, 1989
18. Rahm, E., P.A. Bernstein: A Survey of Approaches to Automatic Schema Matching. VLDB Journal 10: 4, 2001
19. Rahm, E., Do, H.H.: Data Cleaning: Problems and Current Approaches. IEEE Bulletin on Data Engineering 23:4, 2000
20. Winkler, W.E.: Advanced Methods for Record Linking. Section on Survey Research Methods (American Statistical Association), 1994