A Flexible Approach for Planning Schema Matching Algorithms

Fabien Duchateau, Zohra Bellahsene, Remi Coletta

A Flexible Approach for Planning Schema Matching Algorithms *

Fabien Duchateau and Zohra Bellahsene and Remi Coletta

LIRMM - Université Montpellier 2
161 rue Ada 34000 Montpellier
{firstname.name}@lirmm.fr

Abstract. Most of the schema matching tools are assembled from multiple match algorithms, each employing a particular technique to improve matching accuracy and making matching systems extensible and customizable to a particular domain. The solutions provided by current schema matching tools consist in aggregating the results obtained by several match algorithms to improve the quality of the discovered matches. However, aggregation entails several drawbacks. Recently, it has been pointed out that the main issue is how to select the most suitable match algorithms to execute for a given domain and how to adjust the multiple knobs (e.g. threshold, performance, quality, etc.). In this article, we present a novel method for selecting the most appropriate schema matching algorithms. The matching engine makes use of a decision tree to combine the most appropriate match algorithms. As a first consequence of using the decision tree, the **performance** of the system is improved since the complexity is bounded by the height of the decision tree. Thus, only a subset of these match algorithms is used during the matching process. The second advantage is the improvement of the **quality of matches**. Indeed, for a given domain, only the most suitable match algorithms are used. The experiments show the effectiveness of our approach w.r.t. other matching tools.

1 Introduction

Schema matching studies has been initially motivated by schema integration applications in distributed database systems. The task is to produce a global schema from independently constructed schemas [4, 34]. The research highlighted the issues in schema integration of relational schemas, the integrity of integrated schema and different possible techniques to integrate schemas (binary or n-ary). From the artificial intelligence view point, it is the task of integrating multiple ontologies into a single global ontology [18]. Today, this application has been expanded to XML schemas on the Web [27].

Schema matching is the task of discovering correspondences between semantically similar elements of two schemas or ontologies [8, 23, 24, 28]. In this paper, our emphasis is on schema matching, if otherwise explicitly mentioned e.g., ontology alignment. Basic syntax based match definition has been discussed in the survey by Rahm and

* Supported by ANR Research Grant ANR-05-MMSA-0007

Bernstein [31], extended by Shvaiko and Euzenat in [33] with respect to semantic aspect. A plethora of match algorithms have been proposed in the context of schema matching (refer to [31] for a survey of the different measures). And none of these measures outperforms all the others on all existing benchmarks. Therefore, most matching tools [2, 15, 13] aggregate the results obtained by several similarity measures to improve the quality of discovered matches. However, the aggregation function entails several major drawbacks: computing all match algorithms decreases time performance and it can negatively influence the matching quality. And adding new match algorithms mainly implies to update the aggregation function. Finally, a threshold is applied on the aggregated value. Yet, each match algorithm has its own value distribution, thus they should have their own threshold.

In this paper, we present a novel method for combining schema matching algorithms, which enables to avoid the previously mentioned drawbacks. Thus, the matching engine makes use of a decision tree to combine most appropriate match algorithms. As a first consequence of using the decision tree, the **performance** of the system is improved since the complexity is bounded by the height of the tree. Thus, only a subset of these match algorithms is used for matching from a large library of match algorithms. The second advantage is the improvement of the **quality of matches**. Indeed, for a given domain, only the most suitable match algorithms are used. Moreover, the decision tree is flexible since new match algorithms can be added, whatever their output (discrete or continuous values). Finally, MatchPlanner is also able to **tune automatically** the system for providing the optimal configuration for a given matching scenario.

Contributions. We designed a flexible and efficient method for the schema matching problem. The main interesting features of our approach are:

- Introducing the notion of planning in the schema matching process by using a decision tree.
- A tool has been designed based on the planning approach.
- Experiments demonstrate that our tool provides good performance and quality of matches w.r.t. the main matching tools.

Outline. The rest of the paper is organised as follows. Section 2 describes the drawbacks of traditional matching tools. Section 3 focuses on the decision tree to combine match algorithms. Section 4 contains an overview of our prototype. The results of experiments for showing the effectiveness of our approach are presented in section 5. Related work is described in section 6. Finally, we conclude in section 7.

2 Motivations

Most matching tools are assembled from multiple match algorithms, which are then aggregated to improve matching accuracy and making matching systems extensible and customizable to a particular domain. Thus, the aggregation function can be seen as the kernel of a matching tool. However, as pointed out in [22], the main issues are how to select and combine the most suitable match algorithms to execute for a given

domain and how to adjust the multiple knobs (e.g. threshold, performance, quality, etc.). Two other important issues can be added: the thresholds and the precision versus recall problem.

2.1 A Brutal Aggregation Function

Lots of semantic similarity measures have been proposed in the context of schema matching (refer to [31] for a survey of the different measures). And none of these measures outperforms all the others on all existing benchmarks. Therefore, most matching tools [2, 15, 13] aggregate the results obtained by several similarity measures to improve the quality of discovered matches. However, the aggregation function entails major drawbacks on three aspects.

Performance. A first drawback is to apply useless measures, involving a costly time and resource consumption. Indeed, let consider matching two schemas with *n* and *m* elements thanks to a matcher which uses *k* measures. Then $n \times m \times k$ similarities will be computed and aggregated. Yet, there are many cases for which applying the *k* measures is not necessary. The following example shows that even a reliable match algorithm, like the use of a dictionary, may fail to discover even simple matches. Consider the two elements *name* and *named*. Applying a string matching technique like 3-grams between them provides a similarity value equal to 0.5. On the contrary, a dictionary technique (based on Wordnet [35] for example) would result in a very low similarity value since no relationship between the two elements can be inferred. Thus, some techniques can either be appropriate in some cases or they can reveal totally useless. Applying **all measures** between **every couple of elements** involves a costly time and resource consumption.

Quality. The aggregation function may negatively influence the quality. First, it might give more weight to closely-related match algorithms: using several string matching techniques between the polysemous labels *mouse* and *mouse* leads to a high similarity value, in spite of other techniques, like context-based, which could have identified that one label represents a *computer device* and the other an *animal*. Besides, the quality due to the aggregation does not necessarily increase when the number of similarity measures grows. Matching *mouse* and *mouse* with one or two string matching algorithms already results in a high similarity value. Thus using more string matching algorithms would not have an interesting impact.

Flexibility. The aggregation function often requires manual tuning (thresholds, weights, etc.) in the way of combining the measures. This does not make it really flexible w.r.t. new similarity measures contributions. For instance, if a new measure is considered as reliable for a specific domain (based on an ontology for example), how would it be aggregated easily by an expert ?

2.2 The Threshold Applied to the Match Algorithms

To decide whether a couple of schema elements should be considered as a match, a global similarity value is first computed by aggregating the similarity values of several match algorithms. Then this global similarity value is compared with a threshold. Yet the value distribution is very different from a match algorithm to another. Thus, the matching tool should have one threshold for each match algorithm.

2.3 Recall vs Precision

In [19], the author underlines the problem of recall vs precision. Matching tools like COMA++ focus on a better precision, but this does not seem to be the best choice for an end-user in terms of post-effort: consider two schemas containing 100 elements each, there is potentially 10,000 matching possibilities (considering only 1:1 matches), and the number of relevant matches is 25. Let us assume a first matcher discovers 10 relevant matches and achieves 100% precision, then the expert would have to find manually the 15 missing matches among 81,000 possibilities. On the contrary, another matcher returns a set of 300 matches, and it achieves a 100% recall. As all the relevant matches have been discovered, the expert has to remove the 275 irrelevant matches among the 300 ones. Thus, favouring the recall seems a most appropriate choice. And note that technically speaking, it is easier to validate (or not) a discovered mapping than to manually browse two large schemas for adding new matches. The overall metric (also known as accuracy), proposed in [26], was designed to illustrate the post-match effort. However, it does not differentiate matching elimination from matching insertion.

2.4 Another Way to Design Matching Tools ?

To solve previously mentioned drawbacks, our approach aims at replacing the current kernel of matching tools by a decision tree, and it favours the recall to reduce user post-match effort.

3 A Decision Tree to Combine Match Algorithms

The kernel of traditional matching tools is the aggregation function, which combines the similarity values computed by different match algorithms. As it suffers from several drawbacks (see section 2), our idea consists in replacing the aggregation function by a decision tree. We first explain the notion of decision tree, illustrated by an example, and give its interesting features for the matching context. We conclude with a discussion about ongoing work.

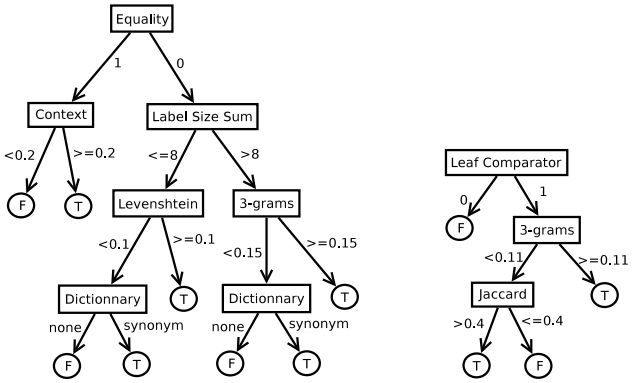
3.1 Decision Trees

The idea is to determine and apply, for a matching scenario, the most suitable matching techniques, by means of a decision tree [29]. In our context, a decision tree is a tree

whose internal nodes represent the similarity measures, and the edges stand for conditions on the result of the similarity measure. Thus, the decision tree contains plans (i.e ordered sequences) of match algorithms. All leaf nodes in the tree are either *true* or *false*, indicating if there is a match or not. We use well-known match algorithms from Second String [32], i.e. Levenshtein, trigrams, Jaro-Winkler, etc. We added the neighbour context from [15], an annotation-based measure, a restriction measure and some dictionary-based [35] techniques.

The similarity value computed by a measure must satisfy the condition (continuous or discrete) on the edges to access a next node. Thus, when matching two schema elements with our decision tree, the first similarity measure, at the root node, is computed and returns a similarity value. According to this value, the edge for which its condition is satisfied leads to the next tree node. This process will iterate until a leaf node is reached, indicating whether the two elements should match or not. The final similarity value between two elements is the last one which has been computed, since we consider that the previous similarity values have only been computed to find the most appropriate measure.

3.2 Matching with Decision Trees



(a) quality-based (b) performance-based

Fig. 1. Examples of decision tree

Figure 1 illustrates two examples of decision tree. The first one (1(a)) focuses on the quality, because it includes some costly measures (context, dictionary). The tree depicted by figure 1(b) aims at discovering some matches quickly, by using mainly string matching measures. Now, let us illustrate with 3 examples how the matching occurs using a decision tree. We want to match the three couples of elements *(author, writer)*, *(name, named)*, and *(title, title)* with the quality-based decision tree (figure 1(a)):

- ***(author, writer)*** is first matched by *equality* which returns 0, then the *label sum size* is computed (value of 12), followed by the *3-grams* algorithm. The similarity value obtained with *3-grams* is low (0.125), implying the *dictionary* technique to be finally used to discover a synonym relationship.
- on the contrary, ***(name, named)*** is matched using *equality*, then *label sum size*, and finally *3-grams* which provides a sufficient similarity value (0.5) to stop the process.
- finally, the couple ***(title, title)*** owns similar labels, implying the *equality* algorithm to return 1. The *context* measure must then be computed to determine if there is a match or not.

Thus, only 9 match algorithms have been computed (4 for *(author, writer)*, 3 for *(name, named)* and 2 for *(title, title)*) instead of 18 (if all distinct match algorithms from the decision tree would have been used). In traditional matching tools based on an aggregation function, all match algorithms would have been applied for each couple of elements.

3.3 Advantages of Using a Decision Tree

This section describes the advantages of using a decision tree in the schema matching context:

- First of all, decision trees are simple to understand or interpret. If a given situation is observable in a model, then the explanation for the condition is easily explained by boolean logic. An example to illustrate this assumption is shown figure 1(a): if two labels are similar, the equality measure returns 1.0 and the context measure is then computed. Indeed, there is no meaning computing more string matching measures like 3-grams or Levenshtein.
- The decision trees are able to handle both numerical and categorical data. This feature is crucial since some similarity measures returns either a number (3-grams, Levenshtein, etc.) or categories (dictionnary-based technique). Other techniques are usually specialised in analysing datasets that have only one type of variable. For example, relation rules can be only used with nominal variables while neural networks can be used only with numerical variables.
- Matching quality does not decrease because of the decision tree. On the contrary, it tends to improve since many related match algorithms (for example string matching, or dictionnary-based, etc.) cannot have a very strong impact on a similarity value. For instance, using several string matching algorithms (3-grams, Levenshtein, Jaro, etc.) for matching very similar labels (e.g. *power* and *tower*) has as much weight as another similarity measure which would discriminate the labels.
- Another advantage is the threshold, which is specific for each match algorithm. Besides, the decision tree enables to consider several cases because a node does not have a limited number of children. Here is an example with three cases: if a similarity value is less than 0.3, then we consider there is no match. If it above 0.7, we consider there is a match. And between 0.3 and 0.7, another match algorithm is computed.

– Finally, using a decision tree does not have a significant impact on the performance. It handles large data in a short time. Besides, in the schema matching context, we show in section 5 that it improves performance by applying only a subset of the match algorithms. Indeed, the complexity in the worst case depends on the maximum height of the decision tree. We can add the fact that the time-costly measures, like the dictionary measure in decision tree 1(a), might appear at the bottom of the tree: they are only computed if necessary, as illustrated by the example with the couple *(name, named)*.

3.4 Ongoing Work for Learning Appropriate Decision Trees

The main drawback of such an approach is that a decision tree may work fine for a given domain, but it can reveal completely inappropriate for another one. This weakness is also true for traditional approaches with an aggregation function. However, our decision tree is more flexible since we are not only able to tune the parameters, but also to design a totally new decision tree.

A first solution is to provide a large set of decision trees, enhanced by an editor to easily allow the creation or modification of decision trees. However, this heavily relies on the user who might not have all the abilities to design a new decision tree. Thus, we are currently working on an automatic solution, based on machine learning techniques and the expert feedback. Indeed, the idea consists in using an existing tree from the library to quickly generate a set of matches. The expert then validates or not some of these matches. This feedback is used to learn a new decision tree thanks to machine learning techniques, for instance C4.5 [30]: the match algorithms which enabled to obtain the most number of validated matches are selected and added in a new decision tree. This process is in charge of setting up the thresholds for each match algorithm too. The user can finally refine the set of matches by using this generated decision tree. This idea is still a preliminary work and will not be discussed anymore in the rest of the paper.

4 Implementation

Our approach has been implemented in Java as a prototype named MatchPlanner. Its architecture is depicted by figure 2. Input of the matching process consists of a set of schemas and a decision tree, composed of match algorithms. Every couple of schema elements is matched against the decision tree. MatchPlanner then outputs a list of mappings which can be validated or not by an expert. Note that our tool is provided with several decision trees. Some have been manually designed while a few others have been generated using machine learning techniques as described in section 3.4. New decision trees can also been added. Each decision tree is associated with a performance and quality value. The first one is relative to the number of discarded algorithms. The second is related to the machine learning techniques. It represents the minimum quality that should be obtained according to the validated matches. Both values only aim at giving an idea about the decision tree: for instance, if the tree only has 10% of discarded match

algorithms, then the matching process should be quite long (the worst case being the one for which the 90% of match algorithms are computed for every couple).

MatchPlanner can also simply be used as a benchmark for testing match algorithms: by creating a decision tree which includes the match algorithms which need to be tested, it is possible to show the effectiveness of each match algorithm for specific schemas.

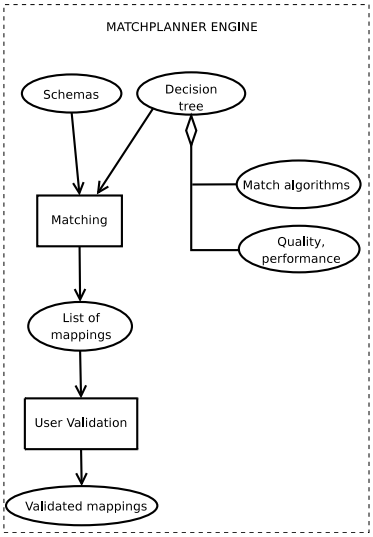


Fig. 2. Architecture of MatchPlanner

5 Experiments

In this section, we demonstrate the benefit of MatchPlanner's decision tree when compared to other schema matching tools, reputed to provide an acceptable matching quality: COMA++ and Similarity Flooding. To the best of our knowledge, these tools are the only ones available for experiments. We first evaluate and compare the matching tools w.r.t the quality aspect, which is crucial in schema matching. Then, we show that MatchPlanner ensures good time performance, an important aspect when dealing with large and / or numerous schemas.

The 3 matching tools have been tested against 7 scenarios:

- **book** and **university** have been widely used in the literature [15, 12]. Both are available in XBenchMatch benchmark [14].
- **thalia** [21] is another benchmark with 40 schemas describing the courses offered by some American universities.
- **travel** are schemas extracted from airfare web forms [1].
- **person** schemas describes people. However, they have been manually designed for the schema matching evaluation, and are available in XBenchMatch too.

– **currency** and **sms** are popular web services which can be found at *http://www.seekda.com*.

Note that the decision trees used by MatchPlanner have been specifically generated for each scenario.

Quality Measures : To evaluate the matching quality of MatchPlanner, we use common measures in the literature, namely precision, recall and F-measure. Precision calculates the proportion of relevant mappings extracted among extracted mappings. Another typical measure is recall which computes the proportion of relevant mappings extracted among relevant mappings. F-measure is a tradeoff between precision and recall.

5.1 Quality Aspect

The first figure 3 illustrates the precision obtained by the matching tools for each scenario. On 5 scenarios, COMA++ achieves the best precision. However, it is also the only tool which is not able to discover any match for one of the scenarios (**travel**). Similarity Flooding obtains twice the best precision, but it achieves the lowest score on the 5 others scenarios. Although MatchPlanner does not emphasize on the precision, it is ranked second in terms of precision for all scenarios.

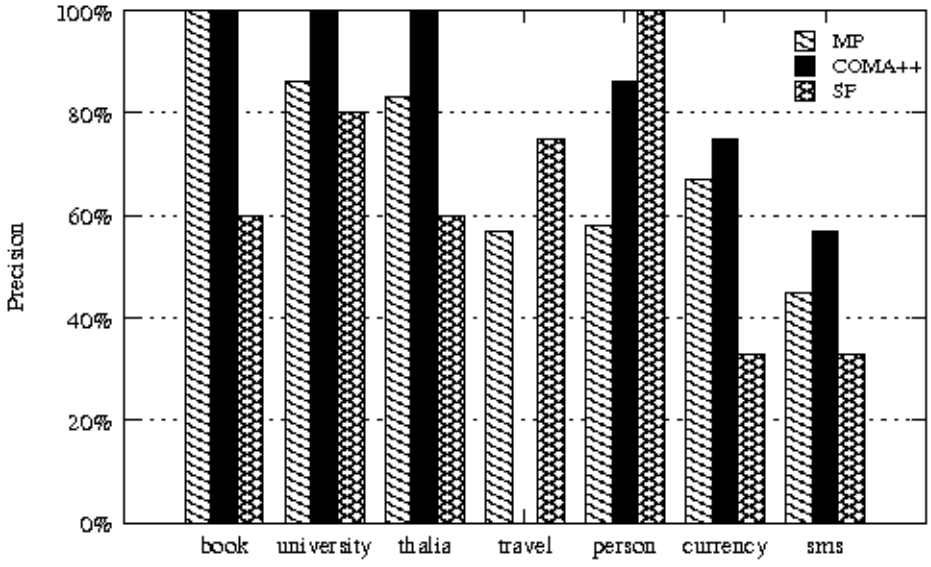


Fig. 3. Precision obtained by the matching tools on the 7 scenarios

The next figure 4 depicts the recall obtained by the tree matching tools for each scenario. For the 7 scenarios, MatchPlanner obtains the highest recall (mostly above 60%), and it discovers all the relevant matches for 3 scenarios. We remind that our tool favours the recall since we believe that a high recall reduces the post-match effort of the user. Leaning towards recall is possible thanks to the numerical conditions on the edges

of the decision trees: they have sufficiently low thresholds to get several matchings, and these results are refined when going down in the tree or when a categorical measure is encountered. Both Similarity Flooding and COMA++ mainly achieve a lower recall (less than 60%), which indicates that an expert would have to browse the input schemas to manually discover the lacking matches.

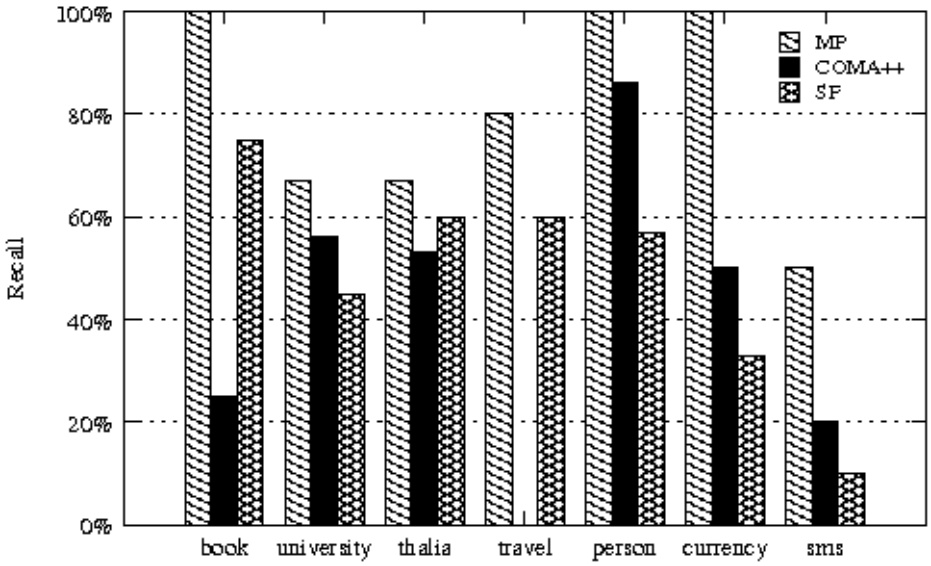


Fig. 4. Recall obtained by the matching tools on the 7 scenarios

Figure 5 depicts the F-measure that each matching tool experimented on the 7 scenarios. Our tool performs best on 6 scenarios, while COMA++ is better on the **person** scenario. We notice that COMA++ might give poor results in 2 cases (**book** and **travel**). Similarity Flooding obtains the lowest F-measure in 4 scenarios. The string matching measures combined with a propagation process may not reveal flexible enough to achieve better results, even with average-sized schemas (**thalia** and **currency**). And we note that the web-based scenarios (**travel**, **currency**, **sms**) are more difficult to match than the others: the F-measure of the matching tools slightly decreases for these scenarios, probably due to their strong heterogeneity.

F-measure represents a tradeoff between precision and recall and it is calculated with the formula

$$F - measure(\beta) = \frac{(\beta^2 + 1) \times Precision \times Recall}{(\beta^2 \times Precision) + Recall} \quad (1)$$

in which the β parameter represents the influence of recall and precision. It is mainly tuned to 1, implying precision and recall to be both as much important. However, we do not believe that recall and precision should be given the same weight. Let us give an example with the **sms** scenario, composed of two schemas with 45 and 64 elements:

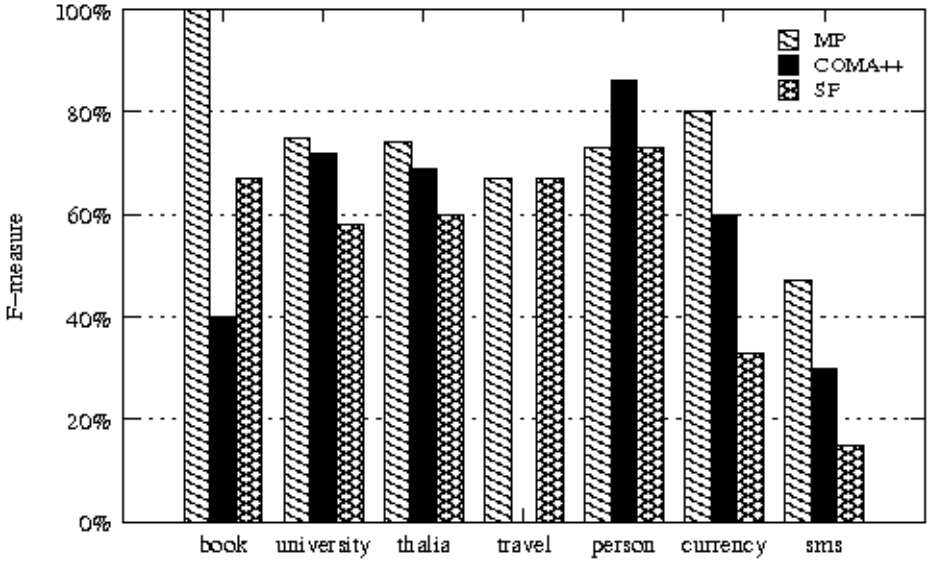


Fig. 5. F-measure ($\beta = 1$) obtained by the matching tools on the 7 scenarios

there are 20 relevant matches between the schemas among 2880 possibilities (we are only considering 1:1 match for sake of clarity). MatchPlanner discovered 10 relevant matches and 12 irrelevant ones. Thus, during the post-match effort, the expert firstly has to manually remove the irrelevant matches. This step mainly consists in validating or not the discovered matches, which can be done quickly. Then she has to find the 10 other relevant matches among the 2880. With COMA++, which discovered 4 relevant matches and 3 irrelevant ones, the expert would have to manually find 16 forgotten matches among 2880 possibilities. As for Similarity Flooding, it discovered 2 relevant matches and 4 irrelevant ones, thus implying the expert to manually find the 18 forgotten matches. Based on this assumption, the recall should be given more weight. By setting α to 2 in the F-measure formula, we generated the figure 6.

On this figure, we notice that MatchPlanner obtains the best F-measure in all scenarios. However, its F-measure slightly decreases for 2 scenarios (**university** and **thalia**) in which the precision is higher than the recall. But it improves for scenarios like **travel**, **person**, **currency**. By tuning the F-measure, Similarity Flooding mainly do not vary its results. This tool is quite balanced enough between precision and recall. On the contrary, COMA++, which favours the precision, has a lower F-measure in most scenarios. When evaluating the result of the matching w.r.t. the post-match effort, MatchPlanner is the tool which reduces most the expert effort.

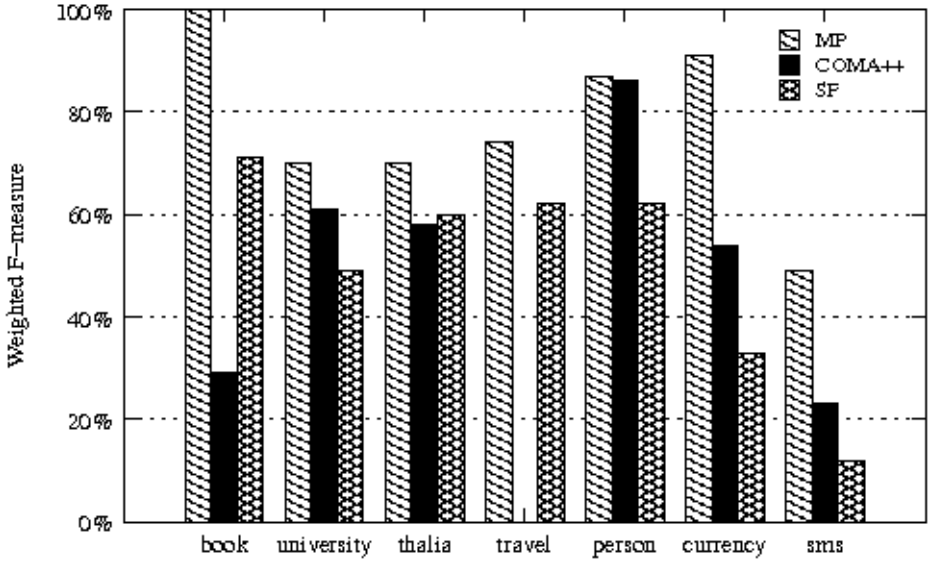


Fig. 6. F-measure ($\beta = 2$) obtained by the matching tools on the 7 scenarios

5.2 Performance Aspect

In this section, we evaluate the time performance of the matching tools. For the scenarios with small schemas (less than 20 elements), the three matching tools performed the matching in a few seconds (less than 3 seconds). With larger schemas (**currency** and **sms**, whose schemas contains more than 50 elements), Similarity Flooding still performs well (less than 6 seconds). MatchPlanner needs 2 more seconds to match the schemas of the **sms** scenario. COMA++ is the slowest matching tool in most cases and it takes nearly 20 seconds to match the **sms** scenario. Although MatchPlanner and COMA++ have the same number of match algorithms in their libraries for the **sms** scenario, MatchPlanner obtains better time performance due to its decision tree, which is able to compute a subset of the match algorithms to match every couple of elements.

5.3 Discussion

In these experiments, Similarity Flooding is the fastest matching tool, probably because it only computes a few initial match algorithms. However, these results are mitigated by the quality of the matches: Similarity Flooding obtains the lowest F-measure for 4 scenarios. We also point out that Similarity Flooding's match algorithms are not efficient from the quality point of view with heterogeneous schemas. On the contrary, COMA++ computes 17 match algorithms for each couple of schema elements. Thus its time performance strongly decreases w.r.t the number of schema elements. Besides, the good precision is obtained to the detriment of the recall. Thus, COMA++ does not achieve the best F-measure in most scenarios. MatchPlanner can be seen as a tradeoff between time performance and quality: although we do no compute all the match algorithms from

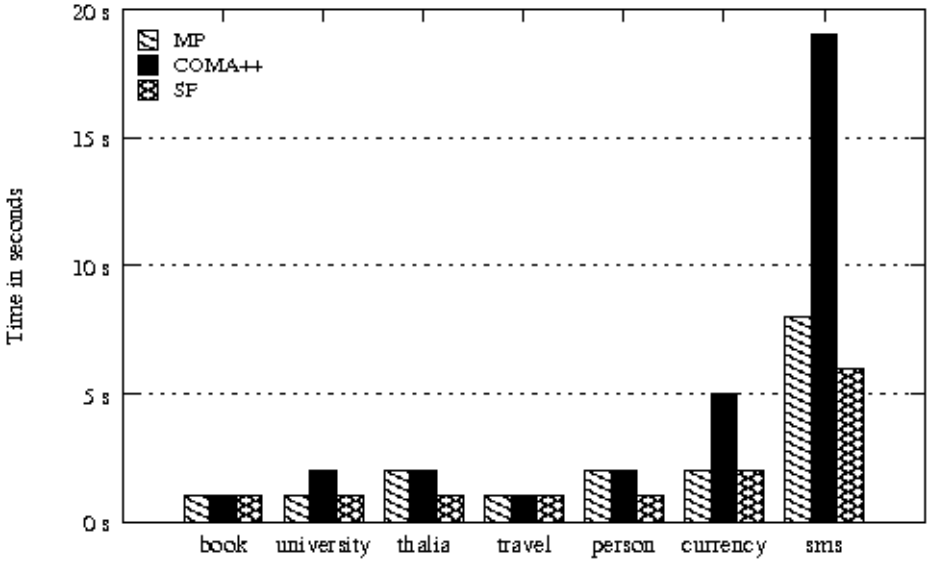


Fig. 7. Time performance for matching each scenario

our library due to the decision tree, the matching quality is mainly better than the one produced by the other matching tools. And it enables to spare some resources, reducing the time execution of the matching process. Finally, our approach shows robustness in terms of quality (highest F-measure in most scenarios) while ensuring acceptable time performance.

6 Related Work

This section covers the related work in schema matching. However, we only mention the works which are the closest of our approach, i.e. they are based on a composition/aggregation approach or they use some machine learning techniques or decision trees.

COMA/COMA++ [2, 9] is a generic, composite matcher with very effective match results. It can process the relational, XML, RDF schemas as well as ontologies. Internally it converts the input schemas as trees for structural matching. For linguistic matching it utilizes a user defined synonym and abbreviation tables like CUPID [24], along with n-gram name matchers. Similarity of pairs of elements is calculated into a similarity matrix. At present it uses 17 element level match algorithms. For each source element, elements with similarity higher then than threshold are displayed to the user for final selection. The COMA++ supports a number of other features like merging, saving and aggregating match results of two schemas. MatchPlanner is able to learn the best combination of match algorithms instead of using the whole set. It outperforms COMA++ in scenario involving large and / or numerous schemas.

Similarity Flooding [26] has been used with Relational, RDF and XML schemas. These schemas are initially converted into labeled graphs and SF approach uses fixpoint computation to determine correspondences of 1:1 local and m:n global cardinality between corresponding nodes of the graphs. The algorithm has been implemented as a hybrid matcher, in combination with a name matcher based on string comparisons. First, the prototype does an initial element-level name mapping, and then feeds these mappings to the structural SF matcher for the propagation process. The weight of similarity between two elements is increased, if the algorithm finds some similarity between the related elements of the pair of elements. In a modular architecture, the components of SF, such as schema converters, the name and structural matchers, and filters, are available as high-level operators and can be flexibly combined within a script for a tailored match operation. One of the main drawback of Similarity Flooding is the matching quality. But this weak point is compensated by the performance. Our approach goes further by using different matching techniques selected by a planner.

S-MATCH/S-MATCH++ [3, 20] takes two directed acyclic graphs like structures e.g. XML schemas or ontologies and returns semantic correspondences between pairs of elements. It uses external oracle Wordnet to evaluate the linguistic matching along with its structural matcher to return a subsumption type match. At present, it uses 13 element level matchers and 3 structural level matchers. It is also heavily dependent on SAT solvers, which decrease its time efficiency.

AUTOMATCH [6] is the predecessor of AUTOPLEX [5], which uses schema instance data and machine learning techniques to find possible matches between two schemas. It explicitly uses Naive Bayesian algorithm to analyze the input instances of relational schemas fields against previously built global schema. The match result consists of 1:1 correspondences and global cardinality. The major drawback of this work is the importance of the data instances. Although this approach is interesting on the machine learning aspect, it seems risky to only have one matching technique.

In [25], the authors propose a model for expressing uncertainty in the schema matching process. A Naive Bayes heuristic, which combines three matchers (term, composition and precedence), is used to discover matches. Given a similarity degree, the Naive Bayes heuristic tries to classify a new match either as correct or incorrect. The matcher independence assumption limits the performance of the approach.

ASID [7] is a 2-step schema matching tool. Reliable matchers (Jaro, TF/IDF applied to descriptions) generate a first set of matches, which is proposed to the user. Then, all non-matched pairs are matched with less credible matchers (Naive Bayes classifier and TF/IDF both applied to data instances). This approach is less flexible than MatchPlanner, since there is only 4 matchers, which are always applied in the same order. Besides, the matchers are combined with an average function.

eTuner [22] aims at tuning schema matching system. It proceeds as follows: a given matching tool (e.g., COMA or Similarity Flooding [26]) is applied against a set of expert matches until an optimal configuration is found for the matching tool. Whereas, MatchPlanner is aimed at learning the best combination of a subset of match algorithms (not matching tools). Moreover, it is able to self tune important features like the performance and quality.

Machine learning techniques have already been used in the context of schema matching. In [10], the authors proposed a full machine learning based approach called LSD. GLUE[11] is the extended version of LSD, which creates ontology/ taxonomy mapping using machine learning techniques. The system is input with set of instances along with the taxonomies. Glue classifies and associates the classes of instances from source to target taxonomies and vice versa. It uses a composite approach, as in LSD, to do so. In this approach, most of the computational effort is spent on the classifiers discovery. As a difference, our approach enables to reuse any existing similarity measures and it focuses on combining them.

Decision trees have been used in ontology matching for discovering hidden mappings among entities [17]. Their approach is based on learning rules for matching terms in Wordnet. In another work [16], decision trees have been used for learning parameters for semi-automatic ontology alignment method. This approach is aimed at optimizing the process of ontology alignment and supporting the user in creating the training examples. However, the decision trees were not used for choosing the best match algorithms.

7 Conclusion

In this paper, we have presented a flexible and efficient approach for schema matching. Unlike other mathing approaches which try to aggregate a given set of match algorithms, our approach makes use of a decision tree to combine the most appropriate match algorithms. The first advantage of using the decision tree is performance improvement of matching process since we do not compute all match algorithms for a given couple of schema elements: only a subset of these match algorithms is used for matching from a large library of match algorithms. The second advantage is the improvement of the matching quality. Indeed, for a given domain, only the most suitable match algorithms are used. Our approach has been implemented as a matching tool and several decision trees are provided. Experiments with 7 scenarios show that our tool outperforms the existing matching tools on the quality aspect. And MatchPlanner has an acceptable time performance. Thus, it can be seen as a tradeoff between Similarity Flooding and COMA++. Besides, the user can select her configuration: she might emphasize on the time performance aspect, thus using a decision tree like the one depicted by figure 1(b). Or she could favour the quality by choosing a decision tree like the one depicted by figure 1(a).

A major ongoing work deals with the automatic generation of the decision trees. This is based on machine learning techniques and expert feedback. The idea consists in creating a bootstrap (several matches are discovered with a default decision tree), then an expert validates (or not) some of the discovered matches. Finally, a new decision tree is generated and can be used to discover more matches. This greatly reduces the number of parameters that the user needs to tune (thresholds, weights, etc.).

References

1. The UIUC web integration repository. Computer Science Department, University of Illinois at Urbana-Champaign. http://metaquerier.cs.uiuc.edu/repository, 2003.

2. D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *SIGMOD Conference, Demo paper*, pages 906–908, 2005.
3. P. Avesani, F. Giunchiglia, and M. Yatskevich. A large scale taxonomy mapping evaluation. In *International Semantic Web Conference*, pages 67–81, 2005.
4. C. Batini, M. Lenzerini, and S. B. Navathe. A comparitive analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
5. J. Berlin and A. Motro. Automated discovery of contents for virtual databases. In *CoopIS*, pages 108–122, 2001.
6. J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *CAiSE*, 2002.
7. N. Bozovic and V. Vassalos. Two-phase schema matching in real world relational databases. In *Data Engineering Workshop, ICDE*.
8. H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *IWWD*, 2003.
9. H. H. Do and E. Rahm. Coma - a system for flexible combination of schema matching approaches. In *VLDB*, pages 610–621, 2002.
10. A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *SIGMOD*, pages 509–520, 2001.
11. A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Y. Halevy. Learning to match ontologies on the semantic web. *VLDB J.*, 12(4):303–319, 2003.
12. A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Ontology matching: A machine learning approach. In *Handbook on Ontologies, International Handbooks on IS*, 2004.
13. C. Drumm, M. Schmitt, H. H. Do, and E. Rahm. Quickmig: automatic schema matching for data migration projects. In *CIKM*, pages 107–116. ACM, 2007.
14. F. Duchateau, Z. Bellahsène, and E. Hunt. Xbenchmatch: a benchmark for xml schema matching tools. In *VLDB Proceedings*, pages 1318–1321. VLDB Endowment, 2007.
15. F. Duchateau, Z. Bellahsene, and M. Roche. A context-based measure for discovering approximate semantic matching between schema elements. In *RCIS*, 2007.
16. M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with apfel. In *ISWC*, 2005.
17. D. W. Embley, L. Xu, and Y. Ding. Automatic direct and indirect schema mapping: Experiences and lessons learned. *SIGMOD Record journal*, 33(4):14–19, 2004.
18. J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.
19. A. Gal. The generation y of xml schema matching (panel description). In *XSym*, pages 137–139, 2007.
20. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-match: an algorithm and an implementation of semantic matching. In *ESWS*, 2004.
21. J. Hammer, M. Stonebraker, , and O. Topsakal. Thalia: Test harness for the assessment of legacy information integration approaches. In *Proceedings of ICDE*, pages 485–486, 2005.
22. Y. Lee, M. Sayyadian, A. Doan, and A. Rosenthal. etuner: tuning schema matching software using synthetic scenarios. *VLDB J.*, 16(1):97–122, 2007.
23. C. Li and C. Clifton. Semantic integration in hetrogeneous databases using neural networks. In *VLDB*, 1994.
24. J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB*, pages 49–58, 2001.
25. A. Marie and A. Gal. Managing uncertainty in schema matcher ensembles. In H. Prade and V. Subrahmanian, editors, *Scalable Uncertainty Management, First International Conference, SUM 2007*, pages 60–73, Washington, DC, USA, Oct. 2007. Springer.
26. S. Melnik, H. G. Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Data Engineering*, pages 117–128, 2002.

27. P. D. Meo, G. Quattrone, G. Terracina, and D. Ursino. Integration of xml schemas at various "'severity"' levels. *Information Systems*, pages 397–434, 2006.
28. T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *VLDB*, pages 122–133, 1998.
29. J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1987.
30. J. R. Quinlan. Improved use of continuous attributes in c4.5. In *Journal of Artificial Intelligence Research*, volume 4, pages 77–90, 1996.
31. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
32. Secondstring. http://secondstring.sourceforge.net/.
33. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *J. Data Semantics IV*, pages 146–171, 2005.
34. S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of hetrogeneous schemas. *VLDB*, pages 81–126, 1992.
35. Wordnet. http://wordnet.princeton.edu, 2007.