

MatchPlanner: A Self Tuning Tool for Planning Schema Matching Algorithms*

Fabien Duchateau
LIRMM - Univ. Montpellier 2
161 rue Ada 34000 Montpellier
duchatea@lirmm.fr

Remi Coletta
LIRMM - Univ. Montpellier 2
161 rue Ada 34000 Montpellier
coletta@lirmm.fr

Zohra Bellahsene
LIRMM - Univ. Montpellier 2
161 rue Ada 34000 Montpellier
bella@lirmm.fr

ABSTRACT

To improve the matching accuracy, most of the schema matching tools aggregate the results obtained by several matching algorithms. The quality of matches depends on the adequacy and of the number of match algorithms used, and their combination and aggregation strategy. However, this aggregation entails several drawbacks on the performance, quality and tuning aspects. In this paper, we present a novel method for combining schema matching algorithms, which enables to avoid these drawbacks. Unlike other composite matchers, it is able to learn the most appropriate match algorithms for a given schema matching scenario. 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 decision tree. For this purpose, for a given domain, only the most suitable match algorithms are used from a large library of match algorithms. The second advantage is the improvement of the **quality of matches**.

1. INTRODUCTION

There are many algorithms that have been designed for schema matching (refer to [10, 12] for a survey of the different approaches). Most of the 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. It has been pointed out in [6] 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.). The solutions provided by current schema matching tools [1, 4] consist in aggregating the results obtained by several match algorithms to improve the quality of the discovered matches. However, aggregation entails three main drawbacks.

The first one is about **performance** in terms of matching processing time. Indeed, **all match algorithms** are applied against

*This work was partially supported by ANR Research Grant ANR-05-MMSA-0007.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '08 Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

every couple of schema elements. Yet, a match algorithm may be very efficient for a certain set of schema but really unsuitable with another set. The following example shows that even a reliable match algorithm, like the use of dictionary, may fail to discover even simple mappings.

Example: Consider the two couples (*author, writer*) and (*name, named*). Applying a string matching technique like 3-grams between (*author, writer*) would result in a very low similarity value close to 0. On the contrary, such a string matching technique applied to (*name, named*) provides a similarity value equal to 0.5. Then, if we apply a dictionary technique (based on Wordnet for example), the labels (*author, writer*) are considered synonyms, implying a high similarity value. While the same technique failed to discover any match between (*name, named*). Thus, some techniques can either be appropriate in some cases or they can be totally useless. Furthermore, applying all match algorithms on every couple of elements involves a costly time and resource consumption.

The second drawback is related to the **quality of matches**. Indeed, the aggregation function may influence the quality. For example, the way of combining the match algorithms is performed by means of a brutal aggregation function, for instance a linear combination. Furthermore, the aggregation 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 of the label represents a *computer device* and the other an *animal*.

Finally, most sophisticated methods often require **manual tuning**, which makes them not really flexible w.r.t. new match algorithms contributions. To the best of our knowledge, there is a few tools like eTuner [6], which have been designed to automatically tune schema matching tools. In eTuner, a given matching tool (e.g. COMA++ [1] or Similarity Flooding [7]) is applied against a set of expert matches until an optimal configuration is discovered for the matching tool. However, eTuner heavily relies on the capabilities of the matching tool, especially for the available match algorithms and its aggregation function. Besides, it does not improve the performance since all match algorithms are computed for every couple of elements.

In this paper, we present a new tool, MatchPlanner, which enables to avoid the previously mentioned drawbacks. Indeed, unlike other composite matchers, it is able to learn the most appropriate match algorithms for a given domain. For this purpose, the matching engine makes use of a decision tree to combine most appropriate match algorithms. As a first advantage, the **performance** of the

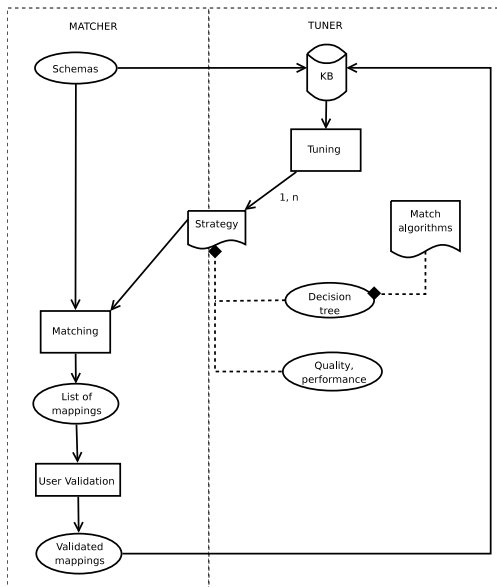


Figure 1: Architecture of MatchPlanner

system is improved since the complexity in $\mathcal{O}(h)$ is bounded by the height h 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 of our approach 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 and implemented MatchPlanner, a new tool for the schema matching task. The main interesting features of our approach are:

- Introducing the notion of planning in the schema matching process by using a decision tree.
- Learning the best strategy (decision) for a given domain
- A tool has been designed based on the planning approach which also is self tuning.

The rest of the paper is organised as follows. Section 2 contains an overview of our prototype. Section 3 focuses on the decision tree to combine match algorithms. Section 4 describes the capacity of MatchPlanner to learn the best sequence of match algorithms. Finally, we conclude in section 5.

2. OVERVIEW OF MATCHPLANNER

Figure 1 shows the architecture of our MatchPlanner prototype, with two main parts: (i) the schema matcher and (ii) the learner. Both parts share a main component, the strategies, since the learner is in charge of generating them while the matcher uses them as a kernel.

As a schema matcher, it takes as input schemas and a strategy. Note that the schemas are also stored in the knowledge base (KB). This KB can be seen as a repository for schemas and expert mappings. A strategy is composed of a decision tree and a ratio between performance and quality. The decision tree is a plan (i.e. an

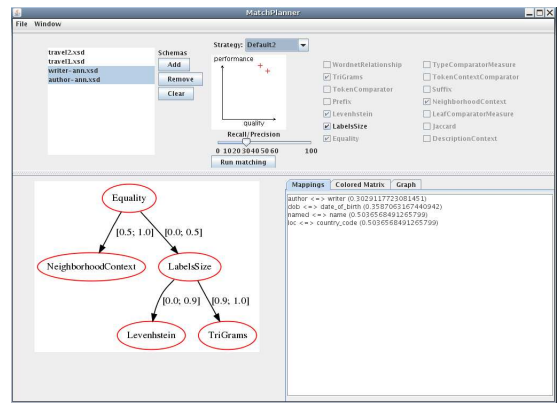


Figure 2: MatchPlanner Menu

ordered sequence) of match algorithms. We use well-known match algorithms from [11], and we add the neighbour context from [3], an annotation-based measure and some dictionary-based technique ([13]). We explain in section 4 how our approach is flexible to allow the integration of new algorithms. The matching process mainly relies on the decision tree (see section 3) to generate a list of mappings. The expert might decide to validate some discovered mappings, which are then stored in the KB and used by the learner.

The learner part aims at generating one or more strategies from the information stored in the KB: the schemas and the expert feedback (validated mappings). It is based on the C4.5 machine learning algorithm as described in section 4. The learned strategies can then be used by the matching process to improve the quality and/or performance.

Figure 2 shows the MatchPlanner's main menu. The top part enables to add schemas and to choose an appropriate strategy. Note that some strategies are already provided with our tool. Once a strategy has been selected, the plot enables to select one decision tree according to expert requirements (quality and performance). In the bottom part is displayed the selected decision tree and the list of discovered mappings.

We would like to demonstrate two scenarios for showing the following features of MatchPlanner: (i) its self tuning capability in section 3 and (ii) its capability as a schema matcher that has been enhanced with incremental mapping discovery in section 4.

3. PLANNING & SELF-TUNING

To avoid previously mentioned drawbacks, we propose to use machine learning techniques for discovering the best combination of match algorithms. We then describe how an expert can intuitively select a decision tree which fulfills her needs.

3.1 Decision Tree for Combining Similarity Measures

The idea is to determine and apply, for a matching scenario, the most suitable matching techniques, by means of a decision tree. Decision tree [8] is a machine learning method, especially well-suited for the task of combining similarity measures. Indeed, it is able to handle both numerical (3-gram, Levenshtein,...) and categorical (data restriction, being synonyms, ...) attributes (i.e. measures in our context). In addition, decision trees are robust to irrelevant attributes. Thus the quality of the similarity measure produced by a decision tree is strictly growing up with the number of similarity measure taken as input. We can also point out that the

tree structure avoids testing some potentially costly similarity measures. Moreover, a learned decision tree is computed only once, and the learning process is performed in less than 1 second.

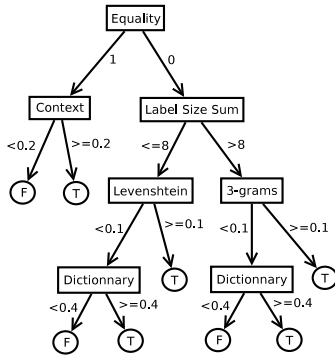


Figure 3: Example of a learned decision tree

Example: Figure 3 shows an example of a decision tree. Each internal node in the tree represents a similarity measure while a leaf node indicates if the couple should be a mapping (T) or not (F). According to the value computed by a similarity measure, another similarity measure might then be used if the constraint on the edge linking both measures is satisfied. Such a decision tree enables to efficiently combine match algorithms: only a subset of the match algorithms are computed when matching 2 nodes. For instance, consider the two couples of elements (*author, writer*) and (*name, named*). When matching (*author, writer*), *equality* measure is first used and it returns a 0 value, implying the *label sum size* to be performed next. This measure computes the total size of the labels (12). Thus, the next algorithm is *3-grams* algorithm, which returns a low similarity(0.11), implying the *dictionary* technique to be finally used to discover a synonym relationship between (*author, writer*). On the contrary, (*name, named*) is matched using *equality* which returns 0, then *label sum size* computes a size of 9. Finally, *3-grams* is performed and it provides an acceptable similarity value (0.5). Thus, only 4 match algorithms have been selected to match the couples of elements ((*author, writer*) and 3 for (*name, named*)). While, in aggregation approaches, all the match algorithms of the library would have been applied for each couple of elements.

In conclusion, using such a decision tree optimizes both time and resources, reducing the impact of previously mentioned problems. Another advantage is that the time-costly measures, like the dictionary measure, appear at the bottom of the decision tree, avoiding to be used if not necessary. Finally, decision trees are able to handle both numerical (3-gram, Levenshtein, etc.) and categorical (data restriction, being synonyms, etc.) match algorithms.

The main drawback of such an approach is that a decision tree may provide good results for a given domain, but it can reveal completely inappropriate for another one. In order to avoid such drawback, our approach is able to compute the best decision tree for a given domain by using machine learning techniques as it will be explained in more details in next section.

3.2 Selecting a Decision Tree

The first demonstration scenario aims at showing the self tuning capability. Given some input schemas, the expert chooses an appropriate decision tree according to her requirements for discovering the matches between the input schemas. To demonstrate this capability, we developed a very intuitive interface, which uses plots

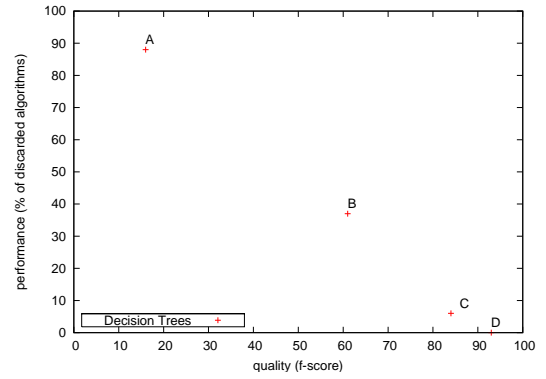


Figure 4: Intuitive interface to select a decision tree

like the one illustrated in figure 4. This figure shows some decision trees (labelled points from A to D), each with different quality and performance ratios. For example, the tree A ensures good performance, but a low quality, since this tree only uses a few string matching measures. On the contrary, the tree D emphasizes on the quality to the detriment of performance, because it includes all measures, including the costly ones (dictionary-based or context). Trees B and C can be seen as trade-offs between quality and performance.

By clicking on a decision tree in the plot, the system will use the corresponding decision tree to discover the matches. This scenario demonstrates that MatchPlanner provides smarter mechanisms to capture the expert requirements: the expert knows the impact of this strategy on the performance and/or quality of the matches. Whereas the others matchers allows a very poor tuning mechanism.

4. A MACHINE LEARNING APPROACH TO CAPTURE THE EXPERT FEEDBACK

In spite of the recent improvement in automatic matching tools, the matching process often remains a semi-automatic process. Indeed, after executing a tool on her schemas to match, the expert has to discard part of the proposed mappings and to look for the ones missed by the tool. Depending on the schemas size and complexity, this manual post matching may be a very long and difficult process, and this effort is never re-used in standard matching tools. We propose here an approach to capture this expert feedback, in order to improve our tool quality and to learn dedicated decision trees.

We propose to formulate the problem of determining, for a given schema matching scenario, the most appropriate decision tree as a machine learning task. The machine learning classification problem consists in predicting the class of an object from a set of its attributes. In the context of schema matching, an object is a couple of elements and its class represents its validity in terms of mapping relevance. The match algorithms represent the attributes of this couple. And as training data, we use the mappings validated or rejected by the expert.

Let us detail the learning process. We first apply the default decision tree with a higher recall (to avoid missing mappings) and present a subset of discovered mapping to the expert. We then integrate the expert validation and refutation as new training-data into our KB and re-generate a set of decision trees, using the C4.5 algorithm [9], known to be one of the most efficient algorithm to learn decision trees, especially when combining both continuous and discrete attributes. It ensures some good properties: a high

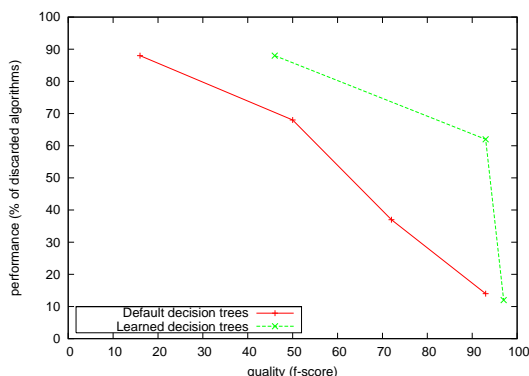


Figure 5: Comparison of default and learned decision trees

classification score, which results in a good matching quality, and it minimizes the height of the generated decision tree, implying better performance. During the learning process, a whole forest of decision trees is generated. The only trees which are kept as strategies are the Pareto optimal [5] trees, which provide an advantage (either on the quality or on the performance). Our tool then uses the learned decision tree to discover the remaining mappings. This capability is very useful in large scale context in which the user should just initiate the schema matching process by giving some mappings.

To evaluate our matching tool, we have chosen four real-world domains composed of two schemas each. They are widely used in the literature. The first one describes a person, the second is related to university courses, the third one on a business order and the fourth one on biology is detailed below. Their main features are given by table 1. The set of relevant mappings for each domain has been processed manually by an expert.

	Person	University	Order	Biology
NB nodes (S_1 / S_2)	11 / 10	18 / 18	20 / 844	719 / 80
Avg NB of nodes	11	18	432	400
Max depth (S_1 / S_2)	4 / 4	5 / 3	3 / 3	7 / 3
NB of Mappings	5	15	10	57

Table 1: Features of the different domains.

To illustrate this scenario, we will take as input large schemas from biology domains (Uni-prot¹ and GeneCards²). We then apply a decision tree previously learned using the three first domains (Person, University, Order) and tuned to provide a ratio $\frac{recall}{precision}$ equals to 100. We then present 100 resulting mappings for validation to the expert. Figure 5 depicts the learned decision trees, compared to the default ones. We observe that the learned trees provide better quality on this biology domain than the default ones. They also provide better performance (they include less similarity measures) to achieve a given quality.

Looking at the learned decision trees, we observe that MatchPlanner is able to detect that *WordNet measure* is not appropriate because of the biology domain specificity. It also discards the *description context* algorithm since the schemas do not include any annotation. Thus, our tool optimizes the matching process by learning appropriate match algorithms for a given domain. Contrary to other matching tools, our approach is flexible since it enables to

¹www.ebi.uniprot.org/support/docs/uniprot.xsd

²www.geneontology.org/GO.downloads.ontology.shtml

easily integrate any new measures: the learner takes this new measure into account by computing all similarities between all couples of elements. Then the measure will be included (or not) in the decision tree according to its effectiveness.

Machine learning techniques have already been used in the context of schema matching. In [2], the authors proposed a full machine learning based approach called LSD, in which 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. Therefore, we avoid wasting time in the learning process and we easily capture any previous and future work on similarity measure.

5. CONCLUSION

In this paper, we present a novel method for improving composite matchers both in terms of time performance and matching accuracy. This method has been implemented as prototype named MatchPlanner and experimented with real world schemas. Unlike other composite matchers, MatchPlanner is able to learn the most appropriate combination of match algorithms for a given domain. Furthermore, MatchPlanner is enhanced with the capability of self tuning the performance and quality parameters. Finally, it can also simply be used as a benchmark for testing schema matching algorithms for a given domain. A major future work is to enrich the KB to improve the robustness. A demo version is available at <http://www.lirmm.fr/~duchatea/MatchPlanner>.

6. REFERENCES

- [1] D. Aumüller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *ACM SIGMOD Conference, Demo paper*, pages 906–908, 2005.
- [2] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *In Proc. of SIGMOD*, pages 509–520, New York, NY, USA, 2001. ACM Press.
- [3] F. Duchateau, Z. Bellahsene, and M. Roche. A context-based measure for discovering approximate semantic matching between schema elements. In *Proceedings of the RCIS*, 2007.
- [4] F. Duchateau, Z. Bellahsene, M. Roche, and M. Roantree. An indexing structure for automatic schema matching. In *Proceedings SMDb Workshop ICDE 2007*, 2007.
- [5] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1983.
- [6] E. Y. Lee, M. Sayyadian, A. Doan, and A. Rosenthal. etuner: Tuning schema matching software using synthetic scenarios. volume 16, pages 97–122, 2007.
- [7] S. Melnik, H. G. Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. of the ICDE*, 2002.
- [8] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, 1987.
- [9] J. R. Quinlan. Improved use of continuous attributes in c4.5. In *JAIR*, volume 4, pages 77–90, 1996.
- [10] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
- [11] Secondstring. <http://secondstring.sourceforge.net/>.
- [12] P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *J. Data Semantics IV*, pages 146–171, 2005.
- [13] Wordnet. <http://wordnet.princeton.edu>, 2007.