Improving Quality and Performance of Schema Matching in Large Scale

Fabien Duchateau, Mathieu Roche, Zohra Bellahsene

Improving Quality and Performance of Schema Matching in Large Scale

Fabien Duchateau* — **Zohra Bellahsene*** — **Mathieu Roche***

* *LIRMM - Université Montpellier 2*
161 rue Ada 34000 Montpellier, France
prenom.nom@lirmm.fr

ABSTRACT. *Schema matching is a crucial task to gather information of the same domain. However, this process is still largely performed manually or semi-automatically, discouraging the deployment of large-scale mediation systems. Indeed, these large-scale scenarii need a solution which ensures both an acceptable matching quality and good performance. In this article, we present the BMatch approach to efficiently match a large number of schemas. The quality aspect is based on the combination of terminological and contextual methods. The performance aspect relies on a B-tree indexing structure to reduce the search space. Finally, experiments with real sets of schemas show that our approach is scalable and outperforms the most referenced matching tools both in quality of matches and performance time.*

RÉSUMÉ. *La découverte de correspondances entre schémas est une étape importante lorsque l'on intégre des informations d'un même domaine. Cependant, ce processus est encore trop souvent effectué manuellement ou au moyen d'approches semi-automatiques. Notre approche Bmatch s'appuie sur une combinaison de mesures terminologiques et d'informations contextuelles pour découvrir des correspondances entre schémas. Par ailleurs, pour être efficace dans un contexte large échelle, nous nous appuyons sur une structure d'indexation B-tree pour réduire l'espace de recherche. Des expérimentations sur des données réelles montrent que notre approche passe bien à l'échelle tout en obtenant globalement une meilleure qualité et de meilleures performances comparativement aux outils de découverte de correspondances de référence.*

KEYWORDS: *semantic similarity, schema matching, BMatch, B-tree index structure, node context, terminological and structural measures.*

MOTS-CLÉS : *similarité semantique, découverte de correspondances entre schmémas, BMatch, structure d'indexation B-tree, contexte d'un noeud, mesures terminologiques et structurelles.*

RSTI - ISI – 13/2008. Modèles et langages pour les bases de données, pages 59 à 82

1. Introduction

Interoperability among applications in distributed environments, including today's World-Wide Web and the emerging Semantic Web, depends critically on the ability to map between them. Unfortunately, automated data integration, and more precisely matching between schema, is still largely done by hand, in a labor-intensive and error-prone process. As a consequence, semantic integration issues have become a key bottleneck in the deployment of a wide variety of information management applications. The high cost of this bottleneck has motivated numerous research activities on methods for describing, manipulating and (semi-automatically) generating schema mappings.

The schema matching problem consists in identifying one or more terms in a schema that match terms in a target schema. The current semi-automatic matchers (Madhavan *et al.*, 2001; Aumueller *et al.*, 2005; Melnik *et al.*, 2002; Euzenat *et al.*, 2004a; Rahm *et al.*, 2001; Yatskevich, 2003) calculate various similarities between elements and they keep the couples with a similarity above a certain threshold. The main drawback of such matching tools is the performance: although the matching quality provided at the end of the process is acceptable, the elapsed time to match implies a static and limited number of schemas. Yet, in many domain areas, a dynamic environment involving large sets of schema is required. Nowadays matching tools must combine both an acceptable quality of matches and good time performance.

In this paper, we present our matching tool, BMatch. It supports both the semantic aspect by ensuring an acceptable matching quality and good performance by using an indexing structure. Contrary to similar works, our approach is generic; it does not use any dictionnary or ontology and is both language and domain independent. The semantic aspect is specifically designed for schemas and consists in using both terminological algorithms and structural rules. Indeed, the terminological approaches enable to discover elements represented by close character strings. On the other hand, the structural rules are used to define the notion of context of a node. This context includes some of its neighbours, each of them is associated a weight representing the importance it has when evaluating the contextual node. Vectors composed of neighbour nodes are compared with the cosine measure to detect any similarity. Finally, the different measures are aggregated for all couples of nodes. Like most of the matchers, this semantic aspect lacks to provide good performance in terms of time. Indeed, comparing each node from one schema to each node from the other schemas is a time-consuming process. To overcome this lack, the second aspect of our approach aimed at improving the performance by using an indexing structure to accelerate the schema matching process. The B-tree structure has been chosen to reach this goal, as it has been designed to search and find efficiently an index among a large quantity of data. Indeed, we assume that two similar labels share at least a common token, so instead of parsing the whole schema, we just search for the tokens indexed in the B-tree. Furthermore, we performed some experiments based on large sets of schema and the results show that our approach is scalable.

Our main contributions are:

– we designed a generic and flexible approach, BMatch, to discover matches between two schemas,

– we introduced the notion of context for a schema node. Our approach is based on both terminological measures and structural measure using this context,

– an indexing structure provides good performance by clustering label tokens,

– an experiment section allows to judge on the results provided by BMatch on both aspects, the matching quality and the time performance.

Outline. The rest of the paper is structured as follows: first we briefly define the main concepts in Section 2. In Section 3, the semantic aspect of our approach is detailed; and Section 4 covers the time performance aspect; in Section 5, we present the results of our experiments; an overview of related work is given in Section 6. Finally, we conclude and outline some future work in Section 7.

2. Preliminary aspects

In this section, we define the main notions used in this paper. Our approach deals with semi-structured data. Schema in this context are trees.

Definition 1 (Schema): a schema is a labeled unordered tree *S = (*V_S, E_S, r_S, *label)* where V_S is a set of nodes; r_S is the root node; $G_S \subseteq V_S \times V_S$ is a set of edges; and *label* $E_S \rightarrow \Lambda$ where Λ is a countable set of labels.

Definition 2 (Semantic similarity measure): let E_1 be a set of elements of schema 1, and E_2 be a set of elements of schema 2. A semantic similarity measure between two elements $e_1 \in E_1$ and $e_2 \in E_2$, denoted as $S_m(e_1, e_2)$, is a metric value based on the likeness of their meaning / semantic content, defined by:

$$S_m : E_1 \text{x} E_2 \rightarrow [0, 1]$$

$(e_1, e_2) \rightarrow S_m(e_1, e_2)$ where a zero value means total dissimilarity and 1 value stands for total similarity. In the rest of the paper, we refer to $S_m(e_1, e_2)$ as the similarity value.

Definition 3 (Automatic schema matching): given two schema element sets E_1 and E_2 and a threshold t, we define automatic schema matching the algorithm to obtain the set of discovered matches M={(e_1, e_2,$S_m(e_1, e_2)$)}, such that between two elements $e_1 \in E_1$ and $e_2 \in E_2$, $S_m(e_1, e_2) \geq t$.

Threshold t may be adjusted by an expert, depending upon the strategy, domain or algorithms used by the schema matching tools.

Example: if $S_m(adresse, address)$ is calculated using the Levenshtein distance algorithm, the similarity value is 0.857 and if the 3-gram algorithm is used, then the result is 0.333 (see Section 3.2 for more details). For another example $S_m(dept, department)$, the Levenshtein distance value is 0 and the 3-gram result is 0.111. These examples show that the threshold has to be adjusted by an expert depending upon the properties of strings being compared and the match algorithms applied. Our approach, presented in Sections 3 and 4, is based on these definitions.

3. BMatch: semantic aspect

In the context of a large scale scenario, performing the matching with similarity measures which use external resources or are based on instances might be a costly process. Besides, an external resource might be inappropriate for matching a given domain: it can be either too general (e.g. Wordnet) or too specific (e.g. an ontology). Thus, our approach is generic and it favours measures which directly rely on the schemas and do not require too much processing. It combines three semantic similarity measures: two of them are terminological and the other is based on the structure. All of these measures are combined with different thresholds to produce a set of matches. In the rest of this section, we first give some motivations for the choice of similarity measures. We describe some important notions of our approach, the terminological measures and the context. Then, the BMatch's semantic aspect is explained in detail. Finally, more precision is given about the parameters.

3.1. *Motivations*

In this section, we explain the motivations underlying our work, especially why we have chosen to combine both terminological and structural approaches.

– Terminological measures are not sufficient, for example:

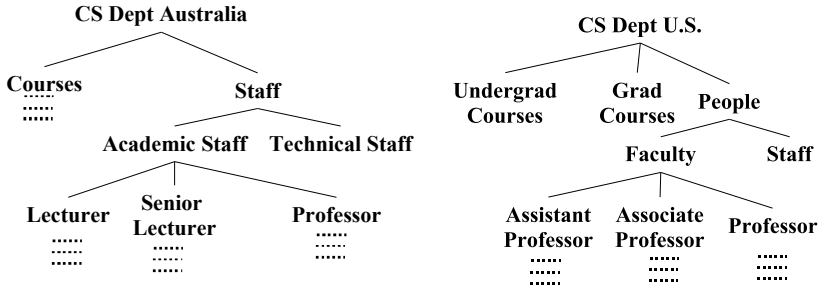
- mouse (computer device) and mouse (animal) lead to a polysemia problem,
- university and faculty are totally dissimilar labels.

– Structural measures have some drawbacks:

- propagating the benefit of irrelevant discovered matches to the neighbour nodes increases the discovery of more irrelevant matches,
- not efficient with small schemas.

Example of schema matching: consider the two following schemas used in (Doan *et al.*, 2004). They represent the organization in universities from different countries and have been widely used in the literature. In the rest of the paper, we will refer to these schemas as the university scenario. With these schemas, the ideal set of

matches given by an expert is {*(CS Dept Australia, CS Dept U.S.), (courses, undergrad courses), (courses, grad courses), (staff, people), (academic staff, faculty), (technical staff, staff), (lecturer, assistant professor), (senior lecturer, associate professor), (professor, professor)*}.



(a) Organization of an Australian university (b) Organization of a US university

Figure 1. *The two university scenario schemas*

Let's imagine that we try to determine a similarity between *Courses* and *GradCourses*. Using terminological measures, namely 3-grams and Levenshtein distance, we discover a high similarity between these labels. *StringMatching* denotes the average between 3-grams and Levenshtein distance values and it represents the similarity obtained by terminological measures. All of these measures are defined in Section 3.2.

– 3grams(Courses, GradCourses) = 0.2

– Lev(Courses, GradCourses) = 0.42

⇒ **StringMatching(Courses, GradCourses) = 0.31**

Now if we consider the nodes *Academic Staff* and *Faculty*, the terminological measures are not useful for discovering a match between these labels, since the labels are totally dissimilar, implying a StringMatching value of 0.002. However, the structural measure enables us to match the labels with a similarity value of 0.37. They are based on the notion of *context*, which represents, for a given node, its semantically most important neighbours. And the contexts of two nodes are compared using the *cosine measure*. This structural measure thus reveals semantic relationships. A detailed explanation of the context and the cosine measure is given in Section 3.4.

– StringMatching(Academic Staff, Faculty) = 0.002

– Context(Academic Staff) = Academic Staff, Lecturer, Senior Lecturer, Professor

– Context(Faculty) = Faculty, Assistant Professor, Associate Professor, Professor

⇒ **CosineMeasure(Context(Academic Staff), Context(Faculty)) = 0.37**

In our approach, we thus combine both terminological and structural measures to avoid the previously described problems.

3.2. *Terminological measures*

To calculate the semantic similarity between two labels, there are many measures which are often cited in the literature (Euzenat *et al.*, 2004a; Cohen *et al.*, 2003; Maedche *et al.*, 2002). This section only presents the two terminological measures used in the BMatch semantic aspect. They are described in more detail in (Duchateau *et al.*, 2007). Both measures compute a value in [0, 1], with the 0 value denoting dissimilarity and 1 denoting total similarity.

3.2.1. *n-grams*

An n-gram (Shannon, 1948) is a similarity measure dealing with subsequences of n items from a given string. n-grams are used in various areas of statistical natural language processing to calculate the number of n consecutive characters in different strings. In general, n value ranges from 2 to 5 and is often set at 3 (Lin, 1998; Kefi, 2006).

3.2.2. *Levenshtein distance*

The Levenshtein distance (Levenshtein, 1966) between two strings is given by the minimum number of operations needed to transform one source string into the target string, where an operation is an insertion, deletion, or substitution of a single character.

3.3. *Node context*

A specific feature of our approach is to consider the neighbour nodes. We have called this notion the context, which represents, given a current node n_c, the nodes denoted n_i in its neighbourhood. In fact, all nodes in the schema may be considered in the neighbourhood of n_c. However, it is quite obvious that the closest nodes n_i are semantically closer to the node n_c. Given this assumption, we calculate the weight of each node n_i according to the node n_c, which evaluates how semantically close the context node n_i is to the node n_c. First we calculate $\Delta\ d$, which represents the difference between the n_c level and the n_i level:

$$\Delta d = |lev(n_c) - lev(n_i)| \quad [1]$$

where $lev(n)$ is the depth of node n from the root. Then we can calculate the weight denoted $\omega(n_c, n_i)$ between the nodes n_c and n_i:

$$\omega(n_c, n_i) = \begin{cases} \omega_1(n_c, n_i), & if\, Anc(n_c, n_i) \text{ or } Desc(n_c, n_i) \\ \omega_2(n_c, n_i), & otherwise \end{cases} \quad [2]$$

where $Anc(n, m)$ (resp. $Desc(n, m)$) is a boolean function indicating whether node n is an ancestor (resp. descendant) of node m. This weight formula is divided into two cases, according to the relationship between the two related nodes. If n is an ancestor or a descendant of m, the Formula 3 is applied. Otherwise we apply Formula 4. The idea behind this weight formula is based on the fact that the closer two nodes are in the tree, the more similar their meaning is.

$$\omega_1(n_c, n_i) = 1 + \frac{K}{\Delta d + |lev(n_c) - lev(n_a)| + |lev(n_i) - lev(n_a)|} \quad [3]$$

$$\omega_2(n_c, n_i) = 1 + \frac{K}{2 \times (|lev(n_c) - lev(n_a)| + |lev(n_i) - lev(n_a)|)} \quad [4]$$

where n_a represents the lowest common ancestor to n_c and n_i, and K is a parameter to allow some flexibility with the context. This is described in more detail in Section 3.5. The value of this weight is in the interval]1,2] for K = 1. Note that this formula, for a given node *n*, gives the same weight to all descendants and ancestors of this node *n* which are at the same level.

Example: let us consider the node *Academic Staff* from schema 1(a). We look for the importance of *Staff* for the node *Academic Staff*. As *Staff* is an ancestor of *Academic Staff*, we apply Formula *3*. Δd, the difference between their levels in the tree hierarchy, is equal to 1. Their lowest common ancestor is *Staff*, and the difference in level between this common ancestor with itself is 0, while it is equal to 1 with the node *Academic Staff*, thus giving us the following result:

$$\omega(\text{Academic Staff}, Staff) = 1 + \frac{1}{1 + 1 + 0} = 1.5 \quad [5]$$

Now we compute the weight of the node *Courses* with regards to *Academic Staff*. They have no ancestor or descendant relationship, so Formula 4 is applied. Their lowest common ancestor is the root node, namely *CS Dept Australia*. *Academic Staff* is 2 levels away from the common ancestor, and *Courses* is 1 level away from it. The weight of *Courses* for the node *Academic Staff* gives:

$$\omega(\text{Academic Staff}, Courses) = 1 + \frac{1}{2 \times (2 + 1)} = 1.17 \quad [6]$$

We can then generalize to obtain the following set of pairs (neighbour, associated weight), also called context vector, which represents the context of the node *Academic Staff*. *{(CS Dept Australia, 1.25), (Courses, 1.17), (Staff, 1.5), (Technical Staff, 1.25), (Lecturer, 1.5), (Senior Lecturer, 1.5), (Professor, 1.5) }* Note that some parameters (described later in this section) have an impact on the context.

3.4. *Semantic match algorithm*

The semantic aspect of BMatch is based on two steps: first we replace terms in the context vectors when they have close character strings. This step uses the Levenshtein distance and 3-gram algorithms (see Section 3.2). Secondly, we calculate the cosine measure between two vectors to determine if their context is close or not.

3.4.1. *Step one: terminological measures to replace terms*

The following describes in detail the first part of the semantic aspect. The two schemas are traversed in preorder traversal and all nodes are compared two by two with the Levenshtein distance and the 3-grams. Both measures are processed and, according to the adopted strategy[1], the highest one or the average is kept. The obtained value is denoted *SM* for *String Measure*. If **SM** is above a certain threshold, which is defined by an expert, then some replacements may occur. The threshold will be discussed in Section 5. We decided to replace the term with the greater number of characters by the term with the smaller number of characters. Indeed, we consider that the smaller sized term is more general than the larger sized one. This assumption can be checked easily since some terms may be written in singular or plural. After this first step, we finally obtain the initial schemas that have possibly been modified with character string replacements.

We have also noted polysemia or synonymy problems. The typical polysemous example is *mouse*, which can represent both an animal and a computer device. In those cases, the string replacement obviously occurs but has no effect since the terms are similar. On the contrary, two synonyms are mainly detected as dissimilar by string matching algorithms. However, the second part of our algorithm, by using the context, enables us to avoid these two problems.

3.4.2. *Step two: cosine measure applied to context vectors*

In the second part of our algorithm, the schemas - in which some string replacements may have occurred by means of step 1 - are traversed again. And the context vector of a current element is extracted in each schema. The neighbour elements composing this vector may be ancestors, descendants, siblings or further nodes of the current element, but each of them has a weight, illustrating the importance of this neighbour with regards to the current node. The two context vectors are compared using the cosine measure, in which we include the node weight. Indeed, when counting the number of occurrences of a term, we multiply this number by its weight. This processing enables us to calculate **CM**, the cosine measure between two context vectors, and thus also the similarity between the two nodes related to these contexts.

1. The maximum and average strategies are reported to be a good tradeoff in the literature.

The cosine measure (Wilkinson *et al.*, 1991) is widely used in Information Retrieval. The cosine measure between the two context vectors, denoted **CM**, is given by the following formula:

$$CM(v_1, v_2) = \frac{v_1 \cdot v_2}{\sqrt{(v_1 \cdot v_1)(v_2 \cdot v_2)}} \quad [7]$$

CM value is in the interval [0,1]. A result close to 1 indicates that the vectors tend to be in the same direction, and a value close to 0 denotes total dissimilarity between the two vectors.

Example: during step 2, the following replacement occurred: Faculty $\leftrightarrow$ Academic Staff. Now consider the two current nodes *Staff* and *People* respectively from schemas 1(a) and 1(b). Their respective and limited[2] context vectors, composed of pairs of a neighbour node and its associated weight, are *{(CS Dept Australia, 1.5), (Faculty, 1.5), (Technical Staff, 1.5) }* and *{(CS Dept U.S., 1.5), (Faculty, 1.5), (Staff, 1.5) }*. As the only common term between the two vectors is *Faculty* with a weight of 1.5, the cosine measure between those context vectors is 0.44.

The context enables to discover or disambiguate a match between polysemous or synonymous pairs. It also enables to discover matches which share other kind of relationships. In the previous example, *Staff* is a "subclassOf" of *People* while in Section 3.1, *Academic Staff* is a synonym of *Faculty*. Note that our approach is not able to discover the kind of relationship between the pair elements, it simply indicates whether it should be a match with regards to the computed similarity.

Finally, we obtain two similarity measures, *SM* and *CM*, with the first one based on terminological algorithms while the second takes the context into account. Here again, a strategy must be adopted to decide how to aggregate those similarity measures. In our approach, the maximum and average were chosen because they generally give better results in experiments than other formulas where one of the measures is favoured. At the end of the process, BMatch deals with a set of matches consisting of all element pairs whose similarity value is above a threshold given by an expert.

3.5. *Tuning the parameters*

Like most matchers, our approach includes some parameters. Although this may be seen as a drawback, since a domain expert is often required to tune them, this is offset by the fact that our application is generic and works with no dictionary regardless of the domain or language.

2. To clarify the example, the context has been voluntarily limited in terms of number of neighbours thanks to the parameters.

– NB_LEVELS: this parameter is used to know the number of levels, both up and down in the hierarchy, to search in order to find the context nodes.

– MIN_WEIGHT: combined with NB_LEVELS, it represents the minimum weight to be accepted as a context node. This is quite useful to avoid having many cousin nodes (that do not have a significant importance) included in the context.

– REPLACE_THRESHOLD: this threshold is the minimum value to be reached to make any replacement between two terms.

– SIM_THRESHOLD: this threshold is the minimal value to be reached to accept a similarity between two schema nodes based on terminological measures.

– K: this coefficient used in formula 2 allows more flexibility. Indeed, it represents the importance we give to the context when measuring similarities.

Given that the number of parameters is important, such an application needs to be tuned correctly to give acceptable results. In (Duchateau *et al.*, 2007), several experiments show the flexibility of BMatch by testing different configurations. This enabled us to set some of the parameters at default values. Besides, we note that some tools like eTuner (Lee *et al.*, 2007) aim at automatically tuning matching tools.

Semantic aspect is hampered by the same drawback as the other matchers, *i.e.* low time performance. This is due to the large number of possibilities, *i.e.* each element from one schema is tested with each element of another schema. The next section presents an indexing structure to accelerate the schema matching process by reducing the search space.

4. BMatch: performance aspect

The first part of this section introduces the B-tree, an indexing structure. Then we explain how this structure is integrated with the semantic part to improve the performance.

4.1. *An indexing structure: the B-tree*

In our approach, we use the B-tree as the main structure to locate matches and create matches between schemas. The advantage of searching for matches using the B-tree approach is that B-trees have indexes that significantly accelerate this process. For example, if you consider the schemas 1(a) and 1(b), they have 8 and 9 elements respectively, implying 72 matching possibilities with an algorithm that tries all combinations. And those schemas are small examples, but in some domains, schemas may contain up to 6 000 elements. By indexing in a B-tree, we are able to reduce this number of matching possibilities, thus providing better time performance.

As described in (Comer, 1979), B-trees have many features. A B-tree is composed of nodes, with each of them having a list of indexes. A B-tree of order *M* means that each node can have up to *M* children nodes and contain a maximum of *M-1* indexes. Another feature is that the B-tree is balanced, meaning that all the leaves are at the same level - thus enabling fast insertion and fast retrieval since a search algorithm in a B-tree of *n* nodes visits only $1+log_{M}n$ nodes to retrieve an index. This balancing involves some extra processing when adding new indexes into the B-tree, however its impact is limited when the B-tree order is high.

The B-tree is a structure widely used in databases due to its efficient capabilities of retrieving information. As schema matchers need to quickly access and retrieve a lot of data when matching, an indexing structure such as B-tree could improve the performances. The B-tree has been preferred to the B+tree (which is commonly used in database systems) since we do not need the costly delete operation. Thus, with this condition, the B-tree seems more efficient than the B+tree because it stores less indexes and it is able to find an index quicker.

4.2. *Principle of our matching algorithm*

Contrary to most other matching tools, BMatch does not use a matrix to compute the similarity of each pair of elements. Instead, a B-tree, whose indexes represent tokens, is built and enriched as we parse new schemas, and the discovered matches are also stored in this structure. The tokens reference all labels that contain it. For example, after parsing schemas 1(a) and 1(b), the *courses* token would hold three labels: *courses* from schema 1(a), *grad courses* and *undergrad courses* from schema 1(b). Note that the labels *grad courses* and *undergrad courses* are also stored under the *grad* and the *undergrad* tokens respectively.

For each input schema, the same algorithm is applied: the schema is parsed element by element following a preorder traversal. This enables us to compute the context vector of each element. The label is split into tokens. We then fetch each of those tokens in the B-tree, resulting in two possibilities:

– no token is found, so we just add it in the B-tree with a reference to the label,

– or the token already exists in the B-tree, and then we try to find semantic similarities between the current label and those referenced by the existing token. We assume that in most cases, similar labels have a common token (and, if not, they may be discovered with the context similarity).

Let us illustrate this case. When *courses* is parsed in schema 1(a), the label is first tokenized, resulting in the following set of tokens: *courses*. We search the B-tree for this single token, but it does not exist. Thus, we create a token structure whose index is *courses* and which stores the current label *courses* and it is added to the B-tree. Later on, we parse *grad courses* in schema 1(b). After the tokenization process,

we obtain this set of tokens: *grad*, *courses*. We then search the B-tree for the first token of the set, but *grad* does not exist. A token structure with this *grad* token as index is inserted in the B-tree, and it stores the *grad courses* label. Then the second token, *courses*, is searched in the B-tree. As it already exists, we browse all the labels it contains (here only the *courses* label is found) to calculate the String Measure (denoted SM) between them and *grad courses*. BMatch can replace one of the labels by another if they are considered similar (depending on the parameters). Whatever happens, *grad courses* is added in the *courses* structure. The next parsed element is *undergrad courses*, which is composed of two tokens, *undergrad* and *courses*. The first one results in an unsuccessful search, implying that an *undergrad* token structure can be created. The second token is already in the B-tree, and it contains the two previously added labels: *courses* and *grad courses*. The String Measures are computed between *undergrad courses* and the two labels, involving replacements if SM reaches a certain threshold. *undergrad courses* is added in the label list of the *courses* token structure. In this way, the index enables us to quickly find the common tokens between occurrences, and to limit the String Measure computation with only a few labels.

At this step, some string replacements might have occurred. Then the parser recursively performs the same action for the descendant nodes, so the children nodes can then be added to the context. Once all descendants have been processed, similarities might be discovered by comparing the label with token references using the cosine and the terminological measures. A parameter can be set to extend the search to the whole B-tree if no matches have been discovered. Let us extend our example. After processing *undergrad courses*, we should go on to its children elements. As it is a leaf, we then search the B-tree again for all tokens which compose the label *undergrad courses*. Under the *undergrad* token, we find only one label, so nothing happens. Under the *courses* token, only one of the three existing labels, namely *courses*, is interesting (one is *undergrad courses* and the other, *grad courses*, is in the same schema). The String Measure is thus applied between *courses* and *undergrad courses*. The Cosine Measure is also performed between their respective contexts, and aggregation of these two measures results in the semantic measure between those labels. If this semantic measure reaches the given threshold, then a match may be discovered.

5. Experiments

As our matching tool deals with both quality and performance aspects, this section is organized in two parts. The first one shows that BMatch provides an acceptable quality of matching regarding the existing matching tools. The second part deals with the performance. For this purpose, large schemas are matched to evaluate the benefit of the B-tree. These experiments were performed on a 2 Ghz Pentium 4 laptop running Windows XP, with 2 Gb RAM. Java Virtual Machine 1.5 is the current version required to launch our prototype. To evaluate our matching tool, we have chosen five

real-world scenarios, each composed of two schemas. These are widely used in the literature. The first one describes a *person*, the second is related to *university courses* from Thalia (Hammer *et al.*, 2005), the third one on *business order* extracted from OAGIS[3] and XCBL[4]. Finally, *currency* and *sms* are popular web services[5]. Their main features are given in Table 1.

Table 1. *Features of the different scenarios*

	Person	University	Order	Currency	SMS
Number of nodes (S_1/S_2)	11/10	8/9	20/844	12/35	46/64
Avg number of nodes	11	9	432	24	55
Max depth (S_1/S_2)	4/4	4/4	3/3	3/3	4/4
Number of mappings	5	9	10	6	20

5.1. *Matching quality*

We present three metrics to evaluate the quality of our matching tool, namely precision, recall and F-measure, that we use to compare BMatch results with those of COMA++ and Similarity Flooding (SF). To the best of our knowledge, these tools are the only ones publicly available.

5.1.1. *Precision, recall and F-measure*

Precision, recall and F-measure formulas are based on Table 2. Precision is an evaluation criterion that is very appropriate for an unsupervised approach. Precision calculates the proportion of relevant pairs extracted among extracted pairs. 100% precision means that all pairs extracted by the system are relevant. Using the notations of Table 2, the precision is given by the Formula 8.

$$Precision = \frac{TP}{TP + FP} \quad [8]$$

Table 2. *Contingency table at the base of evaluation measures*

	Relevant pairs	Irrelevant pairs
Pairs evaluated as relevant by the system	TP (True Positive)	FP (False Positive)
Pairs evaluated as irrelevant by the system	FN (False Negative)	TN (True Negative)

Another typical measure of the machine learning approach is recall, which computes the proportion of relevant pairs extracted among relevant pairs. 100% recall

3. http://www.oagi.org
4. http://www.xcbl.org
5. http://www.seekda.com

means that all relevant pairs of elements have been found. The recall is given by Formula 9.

$$Recall = \frac{TP}{TP + FN} \quad [9]$$

It is often important to find a tradeoff between recall and precision. We can use a measure that takes these two evaluation criteria into account by calculating the F-measure (Van-Risbergen, 1979) :

$$F - measure(\beta) = \frac{(\beta^2 + 1) \times Precision \times Recall}{(\beta^2 \times Precision) + Recall} \quad [10]$$

The β parameter of Formula 10 regulates the respective influence of precision and recall. It is often set at 1 to give the same weight to these two evaluation measures. This measure is widely used in the research field (e.g. INEX[6], TREC[7] or the evaluation of schema matching (Do *et al.*, 2002)).

5.1.2. *Comparison with other matching tools*

In this part, the quality of BMatch is compared with two matching tools known to provide an acceptable matching quality: COMA++ and Similarity Flooding (SF). COMA++ (Aumueller *et al.*, 2005) uses 17 similarity measures to build a matrix between every pair of elements to finally aggregate the similarity values and extract matches. Conversely, SF (Melnik *et al.*, 2002) builds a graph between input schemas and then processes some initial matches with a string matching measure. The matches are refined thanks to the propagation mechanism. Both matching tools are described with more detail in Section 6.

We analysed the set of matches returned by the matching tools to compute the precision, recall, and F-measure. We first focus on our running scenario which describes two universities. BMatch obtained matches for this scenario are shown in Table 3. Our application was tuned with the following configuration, according to the experiments from (Duchateau *et al.*, 2007): the adopted strategy is $avg - max$, the replacement threshold is 0.2, the similarity threshold is 0.15. The number of levels in the context is limited to 2, K and the $minimum\_weight$ are respectively set at 1 and 1.5. For COMA++, all its strategies have been tried and the best obtained results are shown in the following Table 4. Similarity Flooding matches are listed in Table 5. Both matching tools are responsible for their thresholds. Note that in these three tables, a + in the relevance column indicates that the match is relevant.

In Table 3, the fourth match between *CS Dept Australia* and *People* is irrelevant. However, the relevant matches are also noted on line 3 and 6. BMatch is currently not able to determine if one of the matches should be removed or not. Indeed, some complex matches can be discovered, for instance the label *Courses* with both *Grad*

6. http://xmlmining.lip6.fr
7. http://trec.nist.gov

Courses and *Undergrad Courses* on line 2 and 5. Applying a strategy to detect complex matches and remove irrelevant ones is the focus of ongoing research. Similarity Flooding also discovers an irrelevant match.

Table 3. *Matches obtained with BMatch for university scenario*

Element from schema 1	Element from schema 2	Relevance
Professor	Professor	+
Courses	Grad Courses	+
CS Dept Australia	CS Dept U.S.	+
CS Dept Australia	People	
Courses	Undergrad Courses	+
Staff	People	+
Academic Staff	Faculty	+
Technical Staff	Staff	+

Table 4. *Matches with COMA++ for university scenario*

Element from schema 1	Element from schema 2	Relevance
Professor	Professor	+
Technical Staff	Staff	+
CS Dept Australia	CS Dept U.S.	+
Courses	Grad Courses	+
Courses	Undergrad Courses	+

Table 5. *Matches with SF for university scenario*

Element from schema 1	Element from schema 2	Relevance
Professor	Professor	+
Staff	Staff	
CS Dept Australia	CS Dept U.S.	+
Courses	Grad Courses	+
Faculty	Academic Staff	+

After this detailed example for the university scenario, we give the results in terms of precision, recall and F-measure for all scenarios. Figure 2 depicts the precision obtained by the matching tools for the five scenarios. COMA++ is a tool that favours the precision, and achieves a score higher than 70% in three scenarios (university, person and currency). However, we also note that COMA++ obtains the lowest score for the order scenario. BMatch achieves the best precision for two scenarios (order and sms), but the experiments also show that in the other cases, the difference between BMatch and COMA++ precisions is not very significant (10% at most). Although Similarity Flooding scores a 100% precision for the person scenario, it obtains low precision for the others, thus discovering many irrelevant matches.

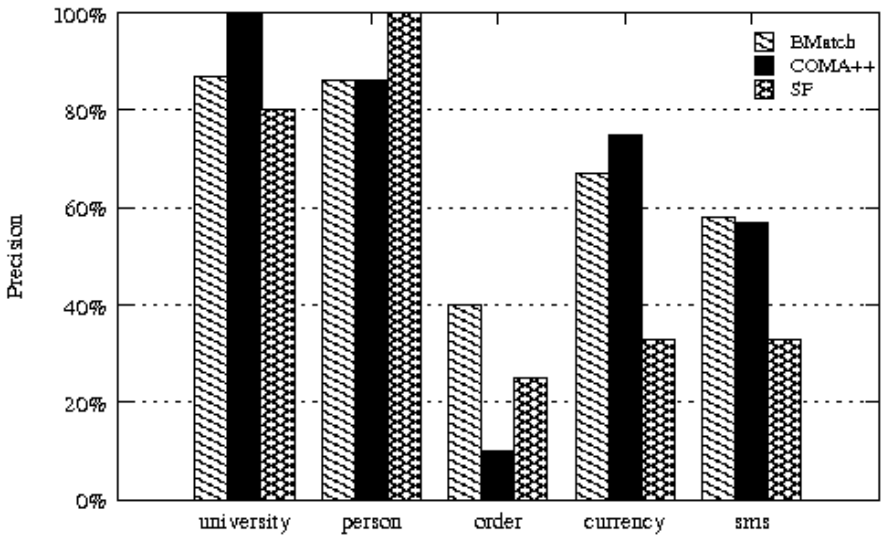


Figure 2. *Precision obtained by the matching tools in the five scenarios*

Figure 3 depicts the recall for the five scenarios. We first note that the matching tools do not discover many relevant matches for the order and sms scenarios (recall less than 40%). BMatch performs best in four scenarios, but it misses many relevant matches for the person scenario (recall equals 32%). This poor recall is mainly due to the numerous tokens in the person schemas and the parameters configuration: with a replacement threshold set at 0.2 and a similarity threshold set at 0.15, our approach missed many relevant matches because terminological measures did not return values which reach these thresholds. We have also demonstrated in (Duchateau *et al.*, 2007) that BMatch is able to obtain 100% recall for the university scenario when its parameters are tuned in an optimal configuration. COMA++ is the only matching tool to obtain a recall above 80% for the person scenario. However, in three scenarios, its recall is at least 15% worse than that of BMatch. Similarity Flooding obtains the lowest recall in four scenarios.

F-measure, the tradeoff between precision and recall, is given in Figure 4. Due to its previous results, Similarity Flooding achieves the lowest F-measure for most scenarios, except for a 73% F-measure for person. BMatch obtains the best F-measure for four scenarios and it outperforms COMA++ by almost 10%. However, both BMatch and COMA++ did not perform well in one scenario (person for BMatch and order for COMA++).

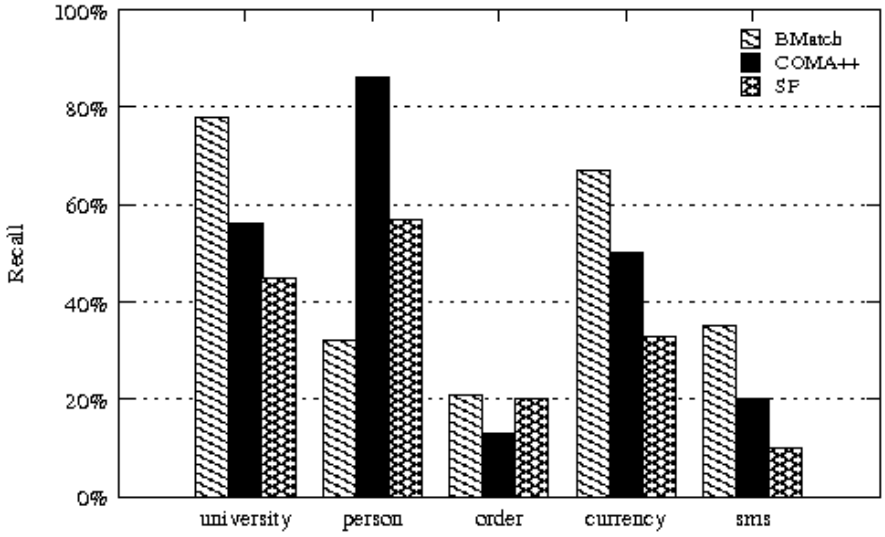


Figure 3. *Recall obtained by the matching tools in the five scenarios*

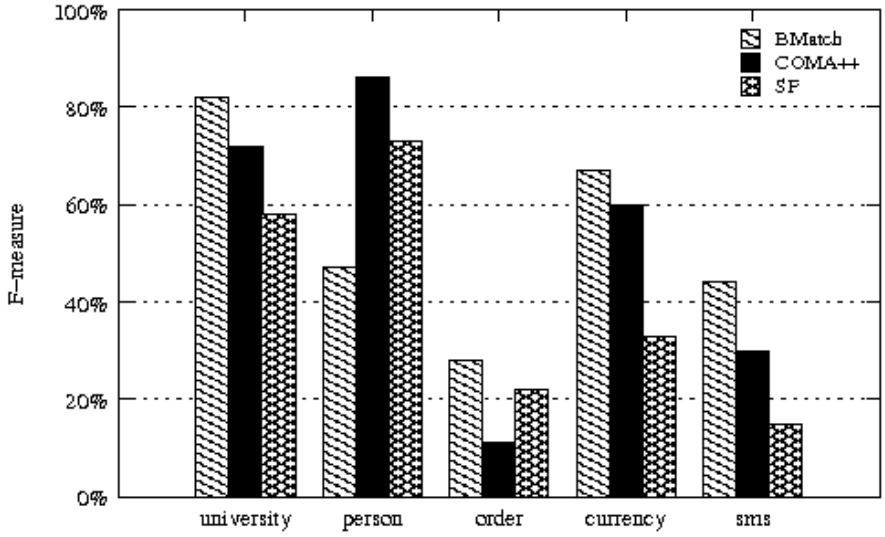


Figure 4. *F-measure obtained by the matching tools in the five scenarios*

5.2. *Time performance aspect*

A matching tool that ensures good performance is required in large scale scenarios, or on the Internet where numerous data sources are available. By focusing on

performance, we mainly mean the time spent to match a large number of schemas. Since the matching tools which are dedicated to large scale scenarios are not available, we compare our BMatch application with a BMatch version without any indexing structure. In this case, the matching algorithm tries to match every pair of nodes for each schema by traversing the trees in preorder.

Table 6 shows the different features of the sets of schemas we used in our experiments. Two large scale scenarios are presented: the first one involves more than a thousand average sized schemas about business-to-business e-commerce taken from the XCBL[8] standards. In the second case, we deal with OASIS[9] schemas which are also business domain related. Note that it is very hard to evaluate the obtained quality when matching large and numerous schemas, because an expert must first manually match them. An example of matching quality with this kind of schema is shown by the order scenario in the previous section.

Table 6. *Characterization of schema sets*

	XCBL set	OASIS set
Average number of nodes per schema	21	2 065
Largest / smallest schema size	426 / 3	6 134 / 26
Maximum depth	7	21

5.2.1. *XCBL Scenario*

Here we compare the performance of BMatch and BMatch without the indexing structure (thus limited to the semantic part) on a large set of average schemas. The results are illustrated by the graph depicted in figure 5. We can see that the version without indexing structure is efficient when there is not a large number of nodes (less than 1600). This is due to the fact that the B-tree requires some maintenance cost to keep the tree balanced. BMatch enhanced with indexing provides good performance with a larger number of nodes, since two thousand schemas are matched in 220 seconds.

5.2.2. *OASIS Scenario*

In this scenario, we are interested in matching 430 large schemas, with an average of 2000 nodes. The graph depicted in figure 6 shows that the version without indexing structure is not suited for large schemas. On the contrary, BMatch is able to match a high number of large schemas in 60 seconds. The graph also shows that BMatch is quite linear. Indeed, it has also been tested for 900 schemas, and BMatch needs around 130 seconds to perform the matching.

8. http://www.xcbl.org
9. http://www.oagi.org

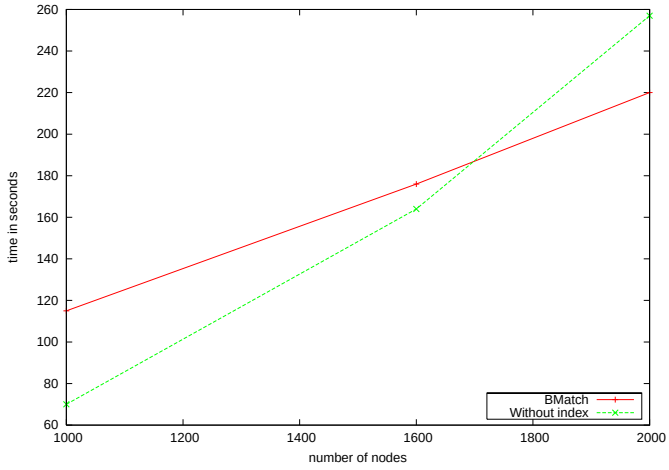


Figure 5. *Matching time with XCBL schemas depending on the number of nodes*

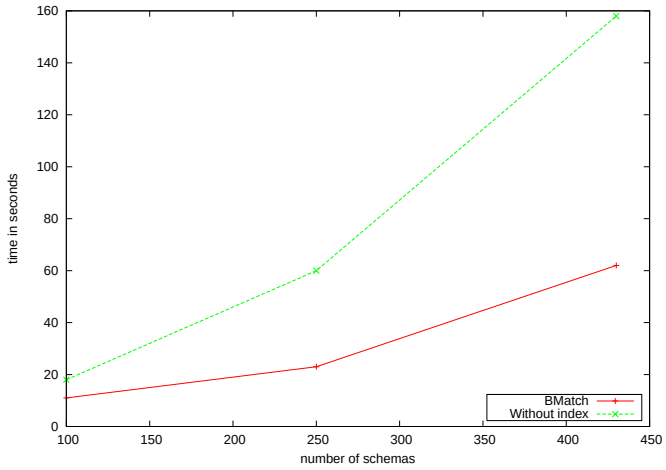


Figure 6. *Matching time with OASIS schemas depending on the number of schemas*

5.2.3. *Comparison with other Matching Tools*

Now, we compare the time performance of the three matching tools for the five scenarios. The time includes both the parsing of the input schemas and the matching process. Table 7 depicts the matching performance of each matching tool for each scenario. All matchers are able to match small schemas (university and order) in less than one second. However, with larger schemas (order, sms), COMA++ and Similarity

Flooding are less efficient. On the other hand, BMatch still ensures good performance while providing the best matching quality (see Section 5.1.2). These matching tools use several methods to store information: COMA++ extracts information from the schemas and stores them in a MySQL database. Then, this information is loaded into memory in directed graphs. The matching matrix, which stores similarities between pairs of elements, is not efficient when the number of pairs is very important. Similarity Flooding stores information in a graph, and then the propagation process runs, and it requires 1 or 2 seconds to perform according to the schemas size. Its time performance are quite similar with that of BMatch. On the contrary, we store all information in the B-tree, and elements are quickly accessed thanks to indexes.

Table 7. *Time performance of COMA++, Similarity Flooding and BMatch on the different scenarios*

	Person	University	Order	Currency	Sms
Average Number of Nodes	11	9	432	24	55
COMA++	$\leq$ 1 s	$\leq$ 1 s	43 s	5 s	19 s
SF	$\leq$ 1 s	$\leq$ 1 s	2 s	1 s	2 s
BMatch	$\leq$ 1 s	$\leq$ 1 s	$\leq$ 1 s	$\leq$ 1 s	$\leq$ 1 s

5.3. *Discussion*

In this section, we conducted some experiments to demonstrate both the quality and time performance of BMatch. We first tuned some parameters to show their influence on the results and to set some of them. Then our matching tool is compared with COMA++ and Similarity Flooding, and we have shown that BMatch provides an acceptable matching quality: in four scenarios out of five, BMatch obtains the highest F-measure. BMatch also performed well in the performance aspect: even with scenarios involving large schemas, there is no impact on BMatch performance, contrary to COMA++ and Similarity Flooding. We also proved the efficiency of the B-tree indexing structure. Indeed, it enables matching of 430 schemas in 50 seconds, while the BMatch version without indexing structure requires 160 seconds. Thus, BMatch is suitable for a large scale scenario.

6. Related work

Many approaches have been devoted to schema matching. Most of them are based on both terminological and structural measures (Melnik *et al.*, 2002; Madhavan *et al.*, 2001; Aumueller *et al.*, 2005; Drumm *et al.*, 2007). However, they mainly aim at matching a small number of schemas. SAT techniques were used in (Avesani *et al.*, 2005) while (Hernandez *et al.*, 2002; Berlin *et al.*, 2002; Doan *et al.*, 2001) deal with instances. Similarly, in the ontology domain, several tools (Ehrig *et al.*, 2004a; Euzenat *et al.*, 2004b; Doan *et al.*, 2003; Ehrig *et al.*, 2004b; Tang *et*

al., 2006) have been designed to fulfill the alignment task. This section only focuses on schema matching tools that we compared BMatch with. Further details about the other approaches are given in surveys (Euzenat *et al.*, 2004a; Rahm *et al.*, 2001; Yatskevich, 2003; Noy, 2004).

6.1. *COMA++*

As described in (Aumueller *et al.*, 2005; Do *et al.*, 2007), COMA++ is a hybrid matching tool that can incorporate many independent matching algorithms. Different strategies, e.g. reuse-oriented matching or fragment-based matching, can be included, offering different results. When loading a schema, COMA++ transforms it into a rooted directed acyclic graph. Specifically, the two schemas are loaded from the repository and the user selects the required match algorithms from the *matcher library*. For each algorithm, each element from the source schema is attributed a threshold value between 0 (no similarity) and 1 (total similarity) with each element of the target schema, resulting in a *cube of similarity values*. The final step involves combining the similarity values given by each matcher algorithm by means of aggregation operators like max, min, $average$, etc. Finally, COMA++ displays all match possibilities and the user checks and validates their accuracy.

The shortcoming of COMA++ is the time required, both for adding files into the repository and matching schemas. In a large scale context, spending several minutes with those operations can entail performance degradation and the other drawback is that it does not support direct matching of many schemas. COMA++ is more complete than BMatch, it uses many algorithms and selects the most appropriate function to aggregate them. However, BMatch is designed for a large scale scenario while COMA++ focuses on the matching quality and is able to match only two schemas at a time. Besides, our approach does not only rely on terminological measures, but it considers the node context too.

6.2. *Similarity Flooding*

Similarity Flooding is a matching tool described in (Melnik *et al.*, 2002) and is based on structural approaches. Input schemas are converted into directed labeled graphs and the aim is to find terminological relationships between those graphs. Then the following structural rule is applied: two nodes from different schemas are considered similar if their adjacent neighbours are similar. When similar nodes are discovered, this similarity is then propagated to adjacent nodes until there are no longer any changes. As in most of matchers, Similarity Flooding generates matches for nodes having a similarity value above a certain threshold.

Our experiment results show that Similarity Flooding does not give good results when labels from the same schema are quite similar (e.g they share several common tokens) or with small schemas. BMatch uses the same structural rule which states

that two nodes from different schemas are similar if most of their neighbours are similar. But BMatch is a combination of terminological and structural measures while Similarity Flooding uses only terminological measures as an initial step, and then the structural aspect to refine the initial matches.

6.3. *Approaches for matching large schemas*

To the best of our knowledge, two works have dealt with large schemas. The first one (Rahm *et al.*, 2004) is based on the COMA++ tool. First, the user divides the schema into fragments and then each fragment from the source schema is mapped to target schema fragments in order to find interfragment matches. Next, these fragment matches are merged to compute the schema level matches. Thus, the tool is not able to directly process large schemas. Another issue related to this approach is the fragmentation criteria of large schemas. The second approach is Porsche (Saleem *et al.*, 2008), which presents a robust mapping method that creates a mediated schema tree from a large set of input XML schemas (converted to trees) and defines mappings from the contributing schema to the mediated schema. It combines tree mining with semantic label clustering which minimizes the target search space and improves time performance, thus making the algorithm suitable for large scale data sharing.

7. Conclusion

In this paper, we have presented our BMatch approach which deals with both the semantic aspect and performance aspect by using an indexing structure, the B-tree. Moreover, our method is generic and flexible. The semantic aspect is based on the combination of two terminological measures and a structural one, which compares the contexts of two elements using the cosine measure.

Experiments have shown that BMatch provides the best matching quality in most scenarios. The experiments have shown that the B-tree indexing structure enables to improve performance in most cases, especially when the number of information that needs to be stored becomes important. An indexing structure could be needed when the schemas are either very large or numerous. Furthermore, experiments also showed that BMatch is able to match large schemas faster than COMA++ or Similarity Flooding. Thus, our method is scalable and provide good performance while ensuring an acceptable matching quality. We are planning to seek for schemas involving more heterogeneity, thus we need to enhance BMatch by adding specific parsers for each format file. Another part of our ongoing work is to detect complex mappings and remove irrelevant ones, probably by an automatic post-match process.

8. References

Aumueller D., Do H. H., Massmann S., Rahm E., " Schema and ontology matching with COMA++", *ACM SIGMOD Conference, DEMO paper*, p. 906-908, 2005.

Avesani P., Giunchiglia F., Yatskevich M., " A Large Scale Taxonomy Mapping Evaluation", *International Semantic Web Conference*, p. 67-81, 2005.

Berlin J., Motro A., " Database Schema Matching Using Machine Learning with Feature Selection", *CAiSE*, 2002.

Cohen W., Ravikumar P., Fienberg S., " A comparison of string distance metrics for name-matching tasks", *Proceedings of the IJCAI-2003*, 2003.

Comer D., " The Ubiquitous Btree", *Computing Surveys*, 1979.

Do H. H., Melnik S., Rahm E., " Comparison of Schema Matching Evaluations", *Proceedings of the 2nd Int. Workshop on Web Databases*, 2002.

Do H. H., Rahm E., " Matching large schemas: Approaches and evaluation", *Information Systems*, vol. 32, n° 6, p. 857-885, 2007.

Doan A., Domingos P., Halevy A. Y., " Reconciling Schemas of Disparate Data Sources - A Machine Learning Approach", *IACM SIGMOD*, 2001.

Doan A., Madhavan J., Dhamankar R., Domingos P., Halevy A. Y., " Learning to match ontologies on the Semantic Web", *VLDB J.*, vol. 12, n° 4, p. 303-319, 2003.

Doan A., Madhavan J., Domingos P., Halevy A., " Ontology Matching: A Machine Learning Approach", *Handbook on Ontologies in Information Systems*, 2004.

Drumm C., Schmitt M., Do H.-H., Rahm E., " Quickmig: automatic schema matching for data migration projects", *CIKM*, ACM, p. 107-116, 2007.

Duchateau F., Bellahsene Z., Roche M., " A Context-based Measure for Discovering Approximate Semantic Matching between Schema Elements", *RCIS*, p. 9-20, 2007.

Ehrig M., Haase P., Stojanovic N., " Similarity for ontologies - a comprehensive framework", *Proc. of Practical Aspects of Knowledge Management*, 2004a.

Ehrig M., Staab S., " QOM - Quick Ontology Mapping", *ISWC*, 2004b.

Euzenat J. et al., State of the art on ontology matching, Technical Report n° KWEB/2004/D2.2.3/v1.2, Knowledge Web, 2004a.

Euzenat J., Valtchev P., " Similarity-Based Ontology Alignment in OWL-Lite", *ECAI*, p. 333-337, 2004b.

Hammer J., Stonebraker M., , Topsakal O., " Thalia: Test harness for the assessment of legacy information integration approaches", *Proceedings of ICDE*, p. 485-486, 2005.

Hernandez M. A., Miller R. J., Haas L. M., " Clio: A Semi-Automatic Tool for Schema Mapping (Software Demonstration)", *ACM SIGMOD*, 2002.

Kefi H., Ontologies et aide à l'utilisateur pour l'interrogation de sources multiples et hétérogènes, PhD thesis, Université de Paris 11, 2006.

Lee Y., Sayyadian M., Doan A., Rosenthal A., " eTuner: tuning schema matching software using synthetic scenarios", *VLDB J.*, vol. 16, n° 1, p. 97-122, 2007.

Levenshtein V., " Binary Codes Capable of Correcting Deletions, Insertions and Reversals", *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

Lin D., “ An information-theoretic definition of similarity”, *Proc. 15th International Conf. on Machine Learning*, Morgan Kaufmann, p. 296-304, 1998.

Madhavan J., Bernstein P., Rahm E., “ Generic schema matching with Cupid”, *VLDB*, 2001.

Maedche A., Staab S., “ Measuring Similarity between Ontologies”, *Proc. of EKAW*, p. 251-263, 2002.

Melnik S., Molina H. G., Rahm E., “ Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching”, *Proc. of ICDE*, 2002.

Noy N. F., “ Semantic Integration: A Survey Of Ontology-Based Approaches”, *SIGMOD Record*, vol. 33, n° 4, p. 65-70, 2004.

Rahm E., Bernstein P. A., “ A survey of approaches to automatic schema matching”, *VLDB Journal*, vol. 10, n° 4, p. 334-350, 2001.

Rahm E., Do H. H., Massmann S., “ Matching large XML schemas”, *SIGMOD Rec.*, vol. 33, n° 4, p. 26-31, 2004.

Saleem K., Bellahsene Z., Hunt E., “ PORSCHE: Performance Oriented Schema Mediation”, *to appear in Information Systems*, 2008.

Shannon C., “ A mathematical theory of communication”, *Bell System Technical Journal*, vol. 27, p. 379-423, 623-656, 1948.

Tang J., Li J., Liang B., Huang X., Li Y., Wang K., “ Using Bayesian decision for ontology mapping”, *Web Semantic*, vol. 4, n° 4, p. 243-262, 2006.

Van-Risbergen C., *Information Retrieval*, 2nd edition, London, Butterworths, 1979.

Wilkinson R., Hingston P., “ Using the cosine measure in a neural network for document retrieval”, *Proc. of ACM SIGIR Conference*, p. 202-210, 1991.

Yatskevich M., Preliminary Evaluation of Schema Matching Systems, Technical Report n° DIT-03-028, Informatica e Telecomunicazioni, University of Trento, 2003.

ANNEXE POUR LE SERVICE FABRICATION
A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :

RSTI - ISI – 13/2008. Modèles et langages pour les bases de données

2. AUTEURS :

Fabien Duchateau[*] *— Zohra Bellahsene*[*] *— Mathieu Roche*[*]

3. TITRE DE L'ARTICLE :

Improving Quality and Performance of Schema Matching in Large Scale

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :

A flexible and scalable approach

5. DATE DE CETTE VERSION :

October 29, 2008

6. COORDONNÉES DES AUTEURS :

– adresse postale :
[*] LIRMM - Université Montpellier 2
161 rue Ada 34000 Montpellier, France
prenom.nom@lirmm.fr
– téléphone : 00 00 00 00 00
– télécopie : 00 00 00 00 00
– e-mail :

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :

LaTeX, avec le fichier de style `article-hermes.cls`,
version 1.23 du 17/11/2005.

8. FORMULAIRE DE COPYRIGHT :

Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
`http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com