BMatch: A Quality/Performance Balanced Approach for Large Scale Schema Matching*

Fabien Duchateau[1] and Zohra Bellahsene[1] and Mathieu Roche[1]

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

Abstract. Schema matching is a crucial task to gather information of the same domain. This is even more the case when dealing with data warehouses, where a large number of data sources are available and require matching and integration. However, the matching process is still largely performed manually or semi-automatically, thus discouraging the use of large scale integration systems. Indeed, these large scale scenarios require a solution which ensures both an acceptable matching quality and good performance. In this article, we present an approach to efficiently match a large number of schemas. The quality aspect is based on the combination of terminological methods and cosine measure between context vectors. The performance aspect relies on a B-tree indexing structure to reduce the search space. Finally, our approach has been implemented and experiments with real sets of schemas show that it is both scalable and provides an acceptable quality of matches as compared to results obtained by the most referenced schema matching tools.
Keywords: semantic similarity, schema matching, BMatch, B-tree index structure, node context, terminological and structural measures

1 Introduction

Data warehousing is a variation of schema integration application, in which, data sources have to be integrated into a data warehouse for decision support systems [1]. In such systems, data are extracted in advance and stored in a repository. Then user queries are addressed directly to the data warehouse system and processed without requiring access to the data sources. Data from source databases have to be transformed into warehouse format using the ETL (Extract, Transform and Loading) process [2]. Initially, data warehousing was performed with relational databases, preceded by object relational databases and semi-structured data sources. Usually, the integrated schema is built manually by the data warehouse designer and the correspondences between this schema and the data sources are discovered semi-automatically. In this paper, we focus on finding out automatically: (i) correspondences between data sources schema and (ii) integrated schema. The number of data source schemas may be very large and the size of the schema may also be large (e.g. thousands of nodes). A typical issue is how to perform and provide an integrated schema in this context, and the time performance

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

issue is crucial.

The schema matching problem consists of identifying one or more terms in a schema that match terms in a target schema. The current semi-automatic matchers [2–8] compute various similarities between elements and keep pairs 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 a dynamic environment involving large sets of schema is required in many domain areas, like B2B or company's information systems [9]. Nowadays matching tools must provide a tradeoff between quality and performance.

In this paper, we present our matching tool, i.e. BMatch. It supports both the semantic aspect by ensuring acceptable matching quality and good time performance by using an indexing structure to match a large number of schemas. Contrary to similar works, our approach is both language and domain independent. It does not use any dictionary or ontology, since these resources might not guarantee an acceptable matching quality. Thus, our approach relies on the schema structure to extract semantic.

The semantic aspect is specifically designed for schemas and involves using both terminological algorithms and structural rules. Indeed, the terminological approaches reveal elements represented by close character strings, which enables to determine semantic proximity. On the other hand, the structural rules are used to define the notion of node context. The node context includes some of its neighbours, each of which is associated with a weight representing the importance with regards to this node. Vectors composed of neighbour nodes are compared with the cosine measure to detect any similarity. Finally, the different measures are aggregated for all pairs of nodes.

Like most matchers, this semantic aspect does not provide good time performance. Indeed, comparing each node from one schema to each node from the other schemas is a time consuming process. Thus, the second aspect of our approach is aimed at improving the performance by using an indexing structure to accelerate the schema matching process. The B-tree structure was chosen to achieve this goal, as it has been designed to efficiently search and find 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 tokens indexed in the B-tree. Furthermore, we performed experiments based on large sets of schema and the results show that our approach is scalable.

Our main contributions are:

- We designed the BMatch approach to discover matches between two schemas. This method is generic: it is not language dependent and it does not rely on dictionaries or ontologies. It is also quite flexible with different parameters.
- As for the semantic aspect, we introduce the notion of context for a schema node. And a formula enables us to extract this context from the schema for a given node. Our approach is based on both terminological measures and a structural measure using this context.

– An indexing structure for matching provides good performance by clustering label tokens.
– Our approach has been implemented as a prototype.
– An experiment section allows asessment of the results provided by BMatch on both aspects: matching quality and time performance. Experiments with real-world schemas widely used in the schema matching literature enable acceptable matching quality. A large number of real XML schemas (OASIS, XCBL) show good performance and are applicable to a large scale scenario. It also enables us to fix the values of some parameters.

The rest of the paper is structured as follows: first we briefly define some general 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:

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

$(e_1, e_2) \rightarrow S_m(e_1, e_2)$ 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.

Definition 4 (Best Match selection): There could be more than one match for an element $e_1 \in E_1$ in E_2. In such situations, the match with maximum similarity value has to be selected.

Given $E_{i2} \subseteq E_2$ of size n, such that $\forall\, e_{ij}$ with $1 \leq j \leq n$ and corresponding to element e_i, the best match for element e_i of E_1 denoted as $match_{ib}$ is given as follows:

$$match_{ib} = \max_{j=1}^{n} S_m(e_i, e_{ij})$$

Our approach, presented in sections 3 and 4, is based on these definitions.

3 BMatch: Semantic Aspect

In the context of a large amount of information, 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.

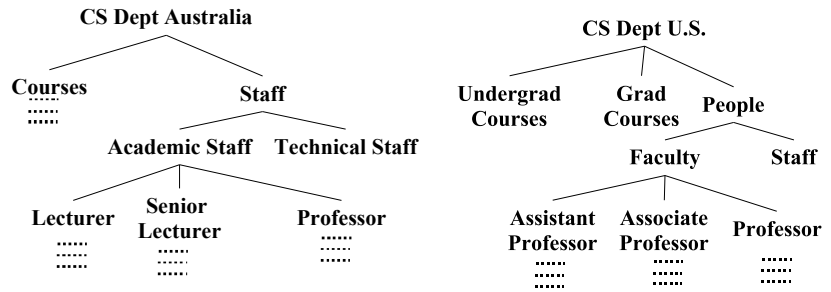
3.1 Semantic Motivations

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 [10]. They represent the organization in universities from different countries and have been widely



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

Fig. 1. The two university scenario schemas

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)*}.

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 later in section 3.2.

- 3grams(Courses, GradCourses) = 0.2
- Lev(Courses, GradCourses) = 0.42

$\Rightarrow$ **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

$\Rightarrow$ **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. Our approach is able to match a large number of schemas without the need of external resources. However, we ensure an acceptable matching quality regarding the existing matching tools (see section 5.1).

3.2 Terminological Measures

To calculate the semantic similarity between two labels, there are many measures which are often cited in the literature [7, 11, 12]. This section only presents the two terminological measures used in the BMatch semantic aspect. They are described in more detail in [13]. Both measures compute a value in [0, 1], with the 0 value denoting dissimilarity and 1 denoting total similarity.

n-grams [14]. An n-gram 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 [15, 16].

Levenshtein distance [17]. The Levenshtein distance 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:

$$\Delta d = |lev(n_c) - lev(n_i)| \tag{1}$$

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:

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

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.

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

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

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:

$$\omega(AcademicStaff, Staff) = 1 + \frac{1}{1+1+0} = 1.5 \tag{5}$$

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:

$$\omega(AcademicStaff, Courses) = 1 + \frac{1}{2 \times (2+1)} = 1.17 \tag{6}$$

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.

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.

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.

The cosine measure [18] is widely used in Information Retrieval. The cosine measure between the two context vectors, denoted **CM**, is given by the following formula:

$$CM(v_1, v_2) = \frac{v_1 \cdot v_2}{\sqrt{(v_1 \cdot v_1)(v_2 \cdot v_2)}} \tag{7}$$

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

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 ↔ 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.

In [19], the authors propose an approach to evaluate tree similarity. It is based on an approximate numerical multidimensional vector, which stores the structure and information of the trees. We do not require such methods to compute the similarity between two node contexts, which would impact the time performance. Indeed, we have demonstrated in the experiments section (see 5.1) that our approach requires a limited context (2 or 3 levels of descendents and ancestors are sufficient), implying a comparison between small subtrees. Besides, the authors show in [19] that they require around 0.1 second to determine the proximity between small trees. Thus, with schemas containing thousands of nodes, it would take several seconds to compute the proximity of the contexts for each pair.

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 Semantic 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

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

by the fact that our application is generic and works with no dictionary regardless of the domain or language.

- 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 [13], 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 [20] aim at automatically tuning matching tools.

This section describes the semantic aspect of BMatch, based on the combination of terminological and structural measures. However, this 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 already used in databases for accelerating the query response time. 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 [21], 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 $1+log_{\mathrm{M}}n$ 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 [22], 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.

	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

Table 1. Features of the different scenarios.

5.1 Matching Quality

Some metrics for evaluating quality of matches We present two metrics used to evaluate the quality of our matching tool. Both metrics are based on table 2, which classifies the relevance of evaluated matches. ROC curves are well suited to measure the quality of a ranking list [23]. Thus, ROC curves are interesting for comparing ranked lists of matching pairs obtained with different parameters. On the other hand, precision, recall and F-measure come from the information retrieval domain. These metrics are described in more detail later in this section. We use them to compare matches discovered by BMatch with results obtained by other matching tools, i.e. COMA++ and Similarity Flooding (SF).

	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)

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

The ROC curves

We first present ROC curves (*Receiver Operating Characteristics*). More details are available in [24]. Initially, ROC curves were used in signal processing and they are often used in the field of medicine to evaluate the validity of diagnostic tests. ROC curves show on the X-coordinate, the rate of false positive (in our case, rate of irrelevant matches) and on the Y-coordinate the rate of true positive (rate of relevant matches). The surface under the ROC curve (*AUC - Area Under the Curve*), can be seen as the effectiveness of a measure of interest. The criterion related to the surface under the curve is equivalent to the Wilcoxon-Mann-Whitney statistical test [25]. Figure 2 depicts an example of a ROC curve. Why is this metric interesting in our context? As our tool

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

is dedicated to matching numerous and/or large schemas, resulting in a huge set of matches, we promote a good ranking of the relevant matches, i.e. they obtain the best similarity values.

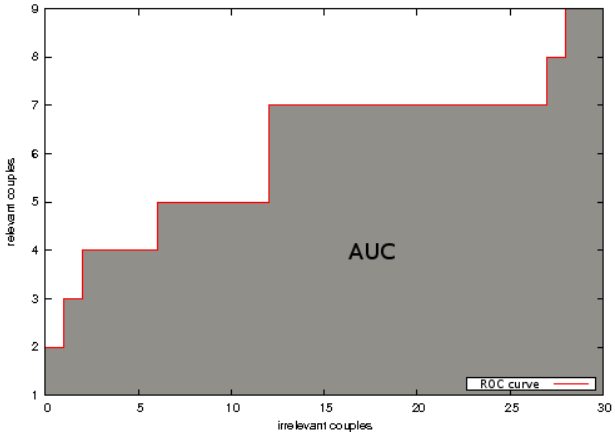


Fig. 2. An example of a ROC curve whose AUC = 0.67

In the case of pair ranking in statistical measures, a perfect ROC curve means that we obtain all relevant pairs at the beginning of the list and all irrelevant pairs at the end of the list. This situation corresponds to $AUC = 1$. The diagonal corresponds to the performance of a random system, with the progress of the rate of true positives being accompanied by an equivalent degradation of the rate of false positives. This situation corresponds to $AUC = 0.5$. If the pairs are ranked by decreasing interest (i.e. all relevant pairs are after the irrelevant ones), then $AUC = 0$. Table 3 shows the commonly accepted interpretation of AUC values. Note that an AUC value below 0.5 corresponds to a bad ranking. An effective measurement of interest to order matches consists of obtaining the highest possible AUC value. This is strictly equivalent to minimizing the sum of the rank of positive examples. The advantage of ROC curves is that they are resistant to imbalance (e.g. an imbalance in the number of positive and negative examples).

AUC value	Interpretation
0.90 -1.00	excellent
0.80 -0.90	good
0.70 -0.80	fair
0.60 -0.70	poor

Table 3. Understanding AUC values

Precision, recall and F-measure

Precision is an evaluation criterion that is very appropriate for an unsupervised approach. Precision calculates the proportion of relevant pairs extracted among extracted pairs. Using the notations of table 2, the precision is given by the formula 8.

$$Precision = \frac{TP}{TP+FP} \tag{8}$$

100% precision means that all pairs extracted by the system are relevant.

Another typical measurement of the machine learning approach is recall, which computes the proportion of relevant pairs extracted among relevant pairs. The recall is given by formula 9.

$$Recall = \frac{TP}{TP+FN} \tag{9}$$

100% recall means that all relevant pairs of elements have been found. This measurement is adapted to the supervised machine learning methods, where all positive examples (relevant pairs of elements) are known.

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 [26] :

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

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 measurements. This measure is widely used in the research field (e.g. INEX[6], TREC[7] or the evaluation of schema matching [27]).

Parameter tuning. BMatch is a flexible matching tool with parameters. Such an application needs to be tested to set some parameters or help the end-user to tune them properly. BMatch ranks all matching pairs differently according to the parameter tuning. Thus, we have used ROC curves to evaluate its matching quality on several scenarios while some parameters are tuned. The advantage of using ROC curves in this case is that the similarity threshold (i.e. the limit above which a matching is considered relevant) is not another crucial parameter to be tuned. Indeed, it is set at 0 to obtain all matching pairs ranked by similarity values. Only three parameters have been tested: the replacement threshold, the two strategies and the number of levels to select the context. The other parameters (K and *minimum weight*) have a lower impact in the matching process. The AUC is calculated for several values of the three

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

studied parameters, for each scenario (university, person, order, currency and sms). Due to space limitation, all ROC curves of these experiments are stored at this URL: *http://www.lirmm.fr/∼duchatea/projects/BMatch/appendixCourbesROC.pdf*.

Note that in the strategy table (figure 6), the first strategy represents a combination of **CM** (context measure) and **SM** (string matching measure). The second strategy is that applied between the terminological measures (Levenshtein distance and 3-grams). And *avg* means the average while *max* stands for the maximum. Thus, $avg - max$ means that we calculated the average between **CM** and **SM**, and that we selected the maximum between Levenshtein distance and 3-grams. The default parameters are as follows: the replacement threshold is set at 0.2, the number of levels at 2 and the default strategy is $avg - max$. This means that when we vary the replacement threshold, the strategy is $avg - max$ and the number of levels is 2.

Table 4 shows the AUC obtained when varying the replacement threshold for the five scenarios. We first note that the replacement threshold should not be too high. This is due to the string matching measures used, which often return values around 0.15 for terminologically close labels. In the person scenario, the results are mostly good regardless of parameter tuning. A low replacement threshold seems best suited in this person scenario, although increasing it does not have a significant impact on the quality. On the contrary, the order and currency scenarios have normalized labels which contain many similar tokens. Thus, increasing the replacement threshold provides a better quality in this case.

Scenario	Replacement Threshold	AUC	Interpretation
University	0.1	0.80	good
	0.2	0.80	good
	0.3	0.68	poor
Person	0.1	0.88	good
	0.2	0.86	good
	0.3	0.85	good
Order	0.1	0.71	fair
	0.2	0.81	good
	0.3	0.85	good
Currency	0.1	0.87	good
	0.2	0.88	good
	0.3	0.89	good
Sms	0.1	0.73	fair
	0.2	0.74	fair
	0.3	0.74	fair

Table 4. Varying the replacement threshold

Table 5 shows the AUC obtained when varying the number of levels for all scenarios. The university and person scenarios confirm that the context should not include

too many nodes that are too far in the hierarchy. On the contrary, a small context is not sufficient to obtain the best quality. In the order and currency scenarios, the schema depth is small (3). Thus, varying the number of levels above 2 has no impact. Note that a context limited to 2 levels is a good heuristic.

Scenario	NB Levels	AUC	Interpretation
University	1	0.72	fair
	2	0.80	good
	3	0.71	fair
Person	1	0.82	good
	2	0.86	good
	3	0.82	good
Order	1	0.81	good
	2	0.81	good
	3	0.81	good
Currency	1	0.88	good
	2	0.88	good
	3	0.88	good
Sms	1	0.73	fair
	2	0.74	fair
	3	0.74	fair

Table 5. Varying the number of levels

Table 6 shows the AUC obtained when varying the strategies for the five scenarios. In the university and person scenarios, it appears that the $avg - max$ strategy is the most appropriate since it is the only one that provides a good AUC. The order scenario offers different results: the $avg - max$ strategy is not the best one, even if it provides a matching quality above 0.8. This is probably due to the *max* used to combine the string matching measures (this also applies for $max - max$ strategy whose AUC is lower too): as the labels contain many similar tokens, this *max* strategy involves more irrelevant replacements, which tends to decrease the overall matching quality. The good results obtained with the currency scenario are due to the four relevant matches (out of six) which are very similar. Regardless the used strategy, these labels are always ranked in the top-ten of the list.

Discussion on parameters All matching tools have their own parameters, and some tools like eTuner [20] help an user to automatically tune them. Otherwise, the user must manually tune the parameters on small samples of schemas. First, some of BMatch parameters depend on the domain application, and the way schemas have been designed. For instance, the replacement threshold might sometimes be tuned to a low value, so some replacements may occur. In other cases, where schema labels share many similar tokens, it needs to be set at a high value to avoid wrong replacements. Yet, those experiments enable us to scale this parameter between 0.1 to 0.3, which can be warranted by the values returned by the terminological measures. For the number of levels,

Scenario	Strategies	AUC	Interpretation
University	avg - avg	0.67	poor
	avg - max	0.80	good
	max - avg	0.67	poor
	max - max	0.71	fair
Person	avg - avg	0.81	good
	avg - max	0.86	good
	max - avg	0.75	fair
	max - max	0.75	fair
Order	avg - avg	0.86	good
	avg - max	0.81	good
	max - avg	0.85	good
	max - max	0.82	good
Currency	avg - avg	0.90	excellent
	avg - max	0.88	good
	max - avg	0.88	good
	max - max	0.89	good
Sms	avg - avg	0.74	fair
	avg - max	0.74	fair
	max - avg	0.74	fair
	max - max	0.71	fair

Table 6. Varying the strategies

a too small or too large context can decrease the quality. However, this parameter offers good results when it is set at 2. As for the strategies, the experiments clearly show that $avg - max$ is an acceptable tradeoff. This strategy provided a good AUC in each of this five scenarios. We also note that whatever the parameter tuning, our application is not able to obtain better than fair results for the sms scenario. Indeed, several relevant matches were ranked in the middle of the matches list, and neither the measures nor the tuning seem efficient to discover them with a higher similarity value.

This study of BMatch parameters led to the following configuration to compare our tool with other matching tools: the strategy is set at $avg - max$, the replacement threshold at 0.2 and the number of levels is 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++ [4] uses 17 similarity measures to build a matrix between every pair of elements to finally aggregate the similarity values and extract matches. Conversely, SF [5] 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.

As COMA++ and SF do not rank all matching pairs by relevance order, the ROC curves cannot be used. Thus 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 7. Our application was tuned with the following configuration: 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 8. Similarity Flooding matches are listed in table 9. 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.

Element from schema 1	Element from schema 2	Similarity value	Relevance
Professor	Professor	0.58	+
Courses	Grad Courses	0.32	+
CS Dept Australia	CS Dept U.S.	0.26	+
CS Dept Australia	People	0.25	
Courses	Undergrad Courses	0.17	+
Staff	People	0.16	+
Academic Staff	Faculty	0.15	+
Technical Staff	Staff	0.15	+

Table 7. Mappings obtained with BMatch for university scenario

Element from schema 1	Element from schema 2	Similarity value	Relevance
Professor	Professor	0.54	+
Technical Staff	Staff	0.53	+
CS Dept Australia	CS Dept U.S.	0.52	+
Courses	Grad Courses	0.50	+
Courses	Undergrad Courses	0.50	+

Table 8. Mappings with COMA++ for university scenario

Element from schema 1	Element from schema 2	Similarity value	Relevance
Professor	Professor	1.0	+
Staff	Staff	1.0	
CS Dept Australia	CS Dept U.S.	1.0	+
Courses	Grad Courses	1.0	+
Faculty	Academic Staff	1.0	+

Table 9. Mappings with SF for university scenario

In table 7, 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 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.

After this detailed example for the university scenario, we give the results in terms of precision, recall and F-measure for all scenarios. Figure 3 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.

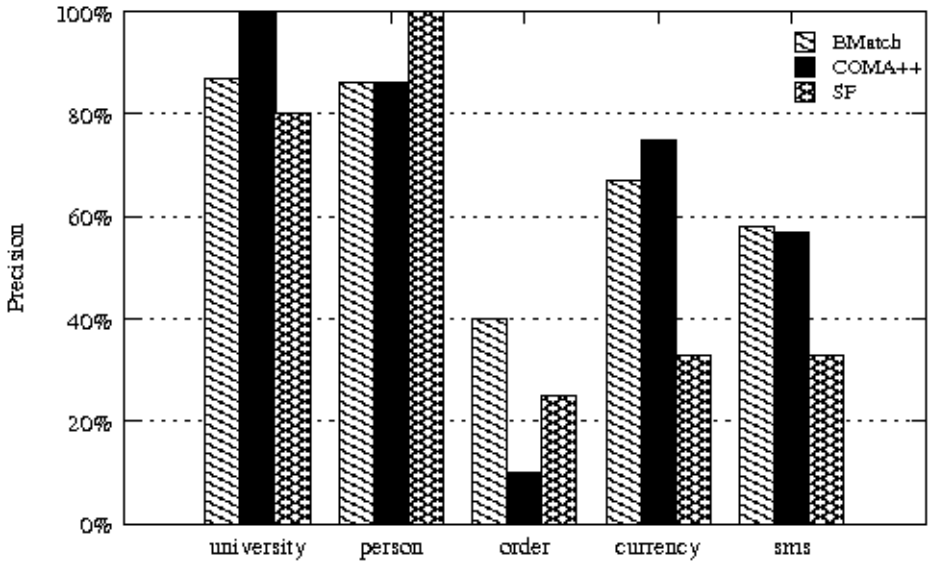


Fig. 3. Precision obtained by the matching tools in the five scenarios

Figure 4 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 seen in 5.1 that BMatch obtains better results when tuned differently. We have also demonstrated in [13] 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.

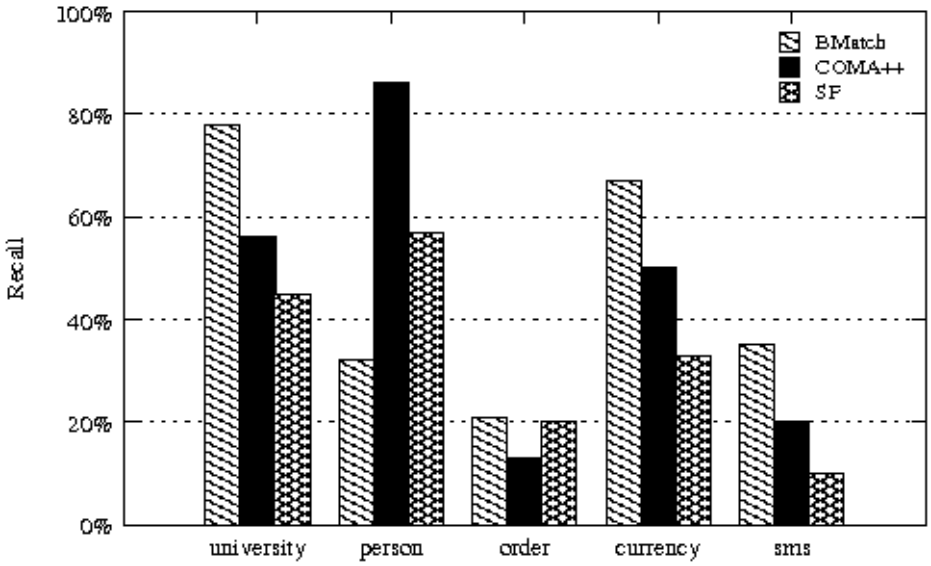


Fig. 4. Recall obtained by the matching tools in the five scenarios

F-measure, the tradeoff between precision and recall, is given in figure 5. 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++).

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. 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

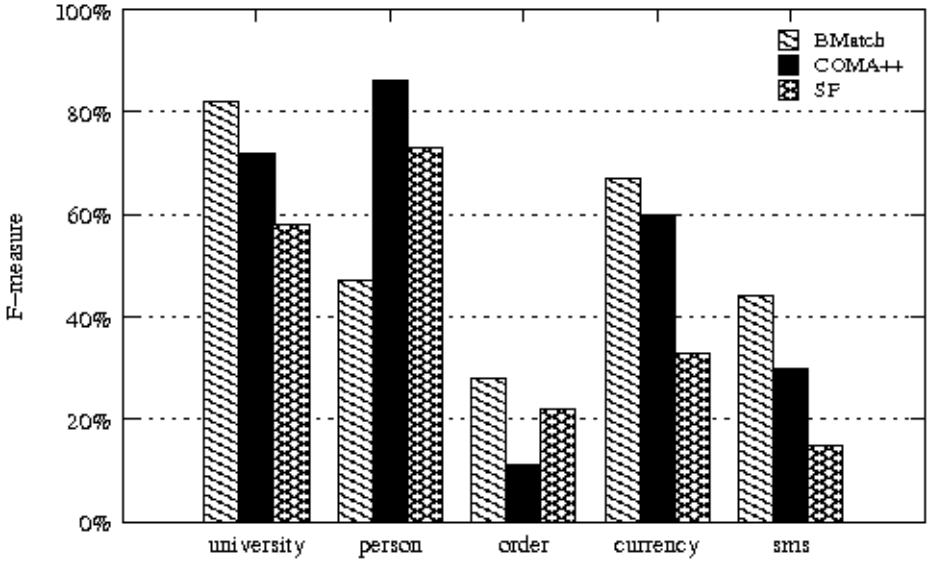


Fig. 5. F-measure obtained by the matching tools in the five scenarios

matching algorithm tries to match every pair of nodes for each schema by traversing the trees in preorder. By focusing on performance, we mainly mean the time spent to match a large number of schemas. The node context is limited to its direct parent and its children nodes. Although this constraint could be removed, it was shown in the quality experiments (see section 5.1) that the context should not include too many further nodes which could have a negative impact on the quality. BMatch parameters do not have a significant impact on the performance aspect.

Table 10 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.

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 6. 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

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

	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

Table 10. Characterization of schema sets

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.

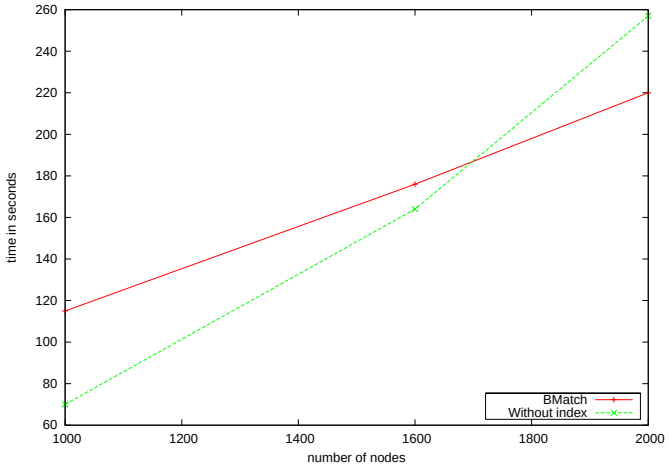


Fig. 6. Matching time with XCBL schemas depending on the number of nodes

OASIS Scenario. In this scenario, we are interested in matching 430 large schemas, with an average of 2000 nodes. The graph depicted in figure 7 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.

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 11 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

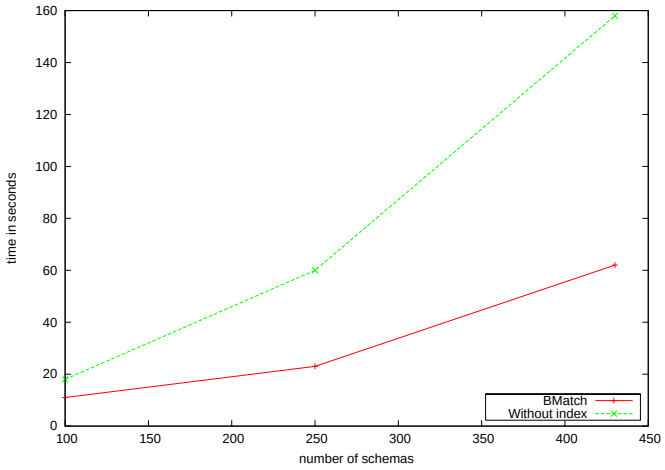


Fig. 7. Matching time with OASIS schemas depending on the number of schemas

	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

Table 11. Time performance of COMA++, Similarity Flooding and BMatch on the different scenarios

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). 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 [4]. 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.

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 [3–6, 9]. However, they mainly aim at matching a small number of schemas. SAT techniques were used in [28] while [29–32] deal with instances. Similarly, in the ontology domain, several tools [33–39] 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 [2, 7, 8, 40].

6.1 COMA++

As described in [4, 41], 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++ mainly focuses on the matching quality and is able to match only two schemas at a time.

6.2 Similarity Flooding

Similarity Flooding is a matching tool described in [5] 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 An Approach for Large Schemas Based on COMA++

To the best of our knowledge, two works have dealt with large schemas. The first one [42] is based on the COMA++ tool [4]. 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 [42] is the fragmentation criteria of large schemas. The second approach is Porsche [43], 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 the BMatch approach which was designed for improving both the quality of matches and time performance of the schema matching process. This approach is very appropriate for large scale contexts like data warehousing systems where a large number of data sources may be integrated. Our approach deals with both the semantic aspect by relying on terminological and structural measures and the performance aspect by using an indexing structure, i.e. the B-tree. Moreover, our method is generic: it is not language dependent and it does not rely on dictionaries or ontologies. It is also quite flexible with different parameters.

The results of our experiments are very enlightening. They show that our method is scalable and provides good time performance thanks to the benefit provided by the index structure. And BMatch ensures a matching quality as good as other matching tools on several real-world scenarios.

We are planning to look 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 matches and remove irrelevant ones, probably by an

automatic post-match process. Finally, we could rely on machine learning methods to discover and/or confirm matches since data are increasingly available with the schemas.

References

1. Kerkri, E.M., Quantin, C., Allaert, F., Cottin, Y., Charve, P., Jouanot, F., Ytongnon, K.: An approach for integrating heterogeneous information sources in a medical data warehouse. Journal of Medical Systems **25**(3) (2001) 167–176
2. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal **10**(4) (2001) 334–350
3. Madhavan, J., Bernstein, P., Rahm, E.: Generic schema matching with cupid. In: VLDB. (2001)
4. Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and ontology matching with coma++. In: ACM SIGMOD Conference, DEMO paper. (2005) 906–908
5. Melnik, S., Molina, H.G., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: Proc. of ICDE. (2002)
6. Tranier, J., Baraer, R., Bellahsène, Z., Teisseire, M.: Where's charlie: Family-based heuristics for peer-to-peer schema integration. In: Proc. of IDEAS. (2004) 227–235
7. Euzenat, J., et al.: State of the art on ontology matching. Technical Report KWEB/2004/D2.2.3/v1.2, Knowledge Web (2004)
8. Yatskevich, M.: Preliminary evaluation of schema matching systems. Technical Report DIT-03-028, Informatica e Telecomunicazioni, University of Trento (2003)
9. Drumm, C., Schmitt, M., Do, H.H., Rahm, E.: Quickmig: automatic schema matching for data migration projects. In: CIKM, ACM (2007) 107–116
10. Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Ontology matching: A machine learning approach. In: Handbook on Ontologies in Information Systems. (2004)
11. Cohen, W., Ravikumar, P., Fienberg, S.: A comparison of string distance metrics for name-matching tasks. In: In Proceedings of the IJCAI-2003. (2003)
12. Maedche, A., Staab, S.: Measuring similarity between ontologies. In: Proc. of EKAW. (2002) 251–263
13. Duchateau, F., Bellahsene, Z., Roche, M.: A context-based measure for discovering approximate semantic matching between schema elements. In: RCIS. (2007) 9–20
14. Shannon, C.: A mathematical theory of communication. Bell System Technical Journal **27** (1948) 379–423, 623–656
15. Lin, D.: An information-theoretic definition of similarity. In: Proc. 15th International Conf. on Machine Learning, Morgan Kaufmann (1998) 296–304
16. Kefi, H.: Ontologies et aide à l'utilisateur pour l'interrogation de sources multiples et hétérogènes. PhD thesis, Université de Paris 11 (2006)
17. Levenshtein, V.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady **10** (1966) 707
18. Wilkinson, R., Hingston, P.: Using the cosine measure in a neural network for document retrieval. In: Proc of ACM SIGIR Conference. (1991) 202–210
19. Yang, R., Kalnis, P., Tung, A.K.H.: Similarity evaluation on tree-structured data. In: SIGMOD, ACM (2005) 754–765
20. Lee, Y., Sayyadian, M., Doan, A., Rosenthal, A.: etuner: tuning schema matching software using synthetic scenarios. VLDB J. **16**(1) (2007) 97–122
21. Comer, D.: The ubiquitous btree. In: Computing Surveys. (1979)
22. Hammer, J., Stonebraker, M., , Topsakal, O.: Thalia: Test harness for the assessment of legacy information integration approaches. In: Proceedings of ICDE. (2005) 485–486

23. Roche, M., Kodratoff, Y.: Pruning Terminology Extracted from a Specialized Corpus for CV Ontology Acquisition. In: Proc. of onToContent'06 workshop - OTM'06. (2006) 1107–1116
24. Ferri, C., Flach, P., Hernandez-Orallo, J.: Learning decision trees using the area under the ROC curve. In: Proceedings of ICML'02. (2002) 139–146
25. Yan, L., Dodier, R., Mozer, M., Wolniewicz, R.: Optimizing classifier performance via an approximation to the Wilcoxon-Mann-Whitney statistic. In: Proceedings of ICML'03. (2003) 848–855
26. Van-Risbergen, C.: Information Retrieval. 2nd edition, London, Butterworths (1979)
27. Do, H.H., Melnik, S., Rahm, E.: Comparison of schema matching evaluations. In: Proceedings of the 2nd Int. Workshop on Web Databases (German Informatics Society. (2002)
28. Avesani, P., Giunchiglia, F., Yatskevich, M.: A large scale taxonomy mapping evaluation. In: International Semantic Web Conference. (2005) 67–81
29. Hernandez, M.A., Miller, R.J., Haas, L.M.: Clio: A semi-automatic tool for schema mapping (software demonstration). In: ACM SIGMOD. (2002)
30. Berlin, J., Motro, A.: Database schema matching using machine learning with feature selection. In: CAiSE. (2002)
31. Doan, A., Domingos, P., Halevy, A.Y.: Reconciling schemas of disparate data sources - a machine learning approach. In: IACM SIGMOD. (2001)
32. Bilke, A., Naumann, F.: Schema matching using duplicates. icde **0** (2005) 69–80
33. Ehrig, M., Haase, P., Stojanovic, N.: Similarity for ontologies - a comprehensive framework. In: Proc. of Practical Aspects of Knowledge Management. (2004)
34. Euzenat, J., Valtchev, P.: Similarity-based ontology alignment in owl-lite. In: ECAI. (2004) 333–337
35. Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., Halevy, A.Y.: Learning to match ontologies on the semantic web. VLDB J. **12**(4) (2003) 303–319
36. Ehrig, M., Staab, S.: Qom - quick ontology mapping. In: ISWC. (2004)
37. Tang, J., Li, J., Liang, B., Huang, X., Li, Y., Wang, K.: Using bayesian decision for ontology mapping. Web Semant. **4**(4) (2006) 243–262
38. Cudre-Mauroux, P., Agarwal, S., Aberer, K.: Gridvine: An infrastructure for peer information management. IEEE Internet Computing **11**(5) (2007) 36–44
39. Li, Y., Bandar, Z.A., McLean, D.: An approach for measuring semantic similarity between words using multiple information sources. IEEE Transactions on Knowledge and Data Engineering **15**(4) (2003) 871–882
40. Noy, N.F.: Semantic integration: A survey of ontology-based approaches. SIGMOD Record **33**(4) (2004) 65–70
41. Do, H.H., Rahm, E.: Matching large schemas: Approaches and evaluation. Information Systems **32**(6) (2007) 857–885
42. Rahm, E., Do, H.H., Massmann, S.: Matching large xml schemas. SIGMOD Rec. **33**(4) (2004) 26–31
43. Saleem, K., Bellahsene, Z., Hunt, E.: Porsche: Performance oriented schema mediation. to appear in Information Systems (2008)