BMatch: a Semantically Context-based Tool Enhanced by an Indexing Structure to Accelerate Schema Matching[*]

Fabien Duchateau
LIRMM
Université Montpellier 2
34392 Montpellier - France
duchatea@lirmm.fr

Zohra Bellahsène
LIRMM
Université Montpellier 2
34392 Montpellier - France
bella@lirmm.fr

Mathieu Roche
LIRMM
Université Montpellier 2
34392 Montpellier - France
mroche@lirmm.fr

Abstract

Schema matching is a crucial task to gather information of the same domain. This is more true on the web, where a large number of data sources are available and require to be matched. However, the schema matching 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 an approach to match efficiently 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, BMatch, has been implemented and the experiments with real sets of schemas show that it is both scalable and provides an acceptable matching quality when compared with the results obtained by the most referenced matching tools.

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

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

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

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 **[10, 1, 12, 16, 7, 13, 19]** 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 schema. Yet in many domain areas, a dynamic environment involving large sets of schema is required. Nowadays' matching tools must combine both an acceptable quality and good 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 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. Thus, 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 the BMatch approach to discover mappings between two schemas. This method is not language-dependent. It does not rely on dictionaries or ontologies. It is also quite flexible with different parameters.

- As for the semantic aspect, we described the notion of context for a schema node. And a formula enables to extract this context from the schema for a given node. Our approach is based on both terminological measures and structural measure using this context.

- An indexing structure for matching provides good performance by clustering label tokens.
- An experiment section allows to judge on the results provided by BMatch on both aspects. Some schemas widely used in the schema matching literature enables to ensure an acceptable matching quality. And large number of real XML schemas (OASIS, XCBL) shows good performance applicable for a large scale scenario. It also enables 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, we explain the motivations that led to our work. In Section 4, an outline of our method is described; in Section 5, we present the results of our experiments; an overview of related work is given in Section 6 and in Section 7, we conclude and outline some future work.

2 Preliminaries

In this section, we define the main notions used in this paper.

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 $e_2 \in E_2$, noted as $S_m(e_1, e_2)$, is a metric value based on the likeness of their meaning/ semantic content, given as:

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

$(e_1, e_2) \rightarrow S_m(e_1, e_2)$ where a zero value means a total dissimilarity and 1 value stands for total similarity.

Definition 3 (Automatic Schema Matching): Given two schema elements sets E_1 and E_2 and a similarity measure threshold t. We define Automatic Schema Matching, between two elements e_1 and e_2, noted as match(e_1, e_2), as follows in Algorithm 1:

Algorithm 1 Automatic Schema Matching

```
Require: schema $E_1$, schema $E_2$, threshold t
Ensure: Matches automatically discovered
  for $(e_1, e_2) \in E_1 \mathrm{x} E_2$ do
    if $S_m(e_1, e_2) < t$ then
      match$(e_1, e_2)$ = false
    else
      match$(e_1, e_2)$ = true
      $d = S_m(e_1, e_2)$
    end if
  end for
```

We define d as the similarity degree (or value) with $d = S_m(e_1, e_2)$. Threshold t may be adjusted by an expert, depending upon the strategy, domain or algorithms used by the schema matching tools.

Example 2.1: If match(adresse, address) is calculated using Levenhstein distance algorithm, the value of d is 0.857 and if 3-gram algorithm is used, then the result for d is 0.333. For another example match(dept, department), Levenhstein distance value of d is 0 and 3-gram result is 0.111. The examples show that the threshold has to be adjusted by an expert depending upon the properties of strings being compared and the match algorithms being applied.

Definition 4 (Best Match selection): There can be the possibility of more than one match for an element $e_1 \in E_1$ in E_2. In such situation, the match with maximum similarity degree has to be selected. This case can be formally defined as:

Given $E_{i2} \subseteq E_2$ of size n, such that $\forall\ e_{ij}$ corresponding to element e_i, match(e_i,e_{ij}) is true; where $1 \leq j \leq n$. Best match for element e_i of E_1 noted as $match_{ib}$ is given as following:

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

Definition 5 (Schema Mapping): Given E_1 a set of elements of schema 1, E_2 a set of elements of schema 2 and I a set of mappings identifiers. We define a mapping between two elements $e_1 \in E_1$ and $e_2 \in E_2$ by the following function noted as Map:

Map: I$\text{x}E_1\text{x}E_2\text{x}Fs \rightarrow$ I$\text{x}E_1\text{x}E_2\text{x}[0,1]\text{x}K$
$(id, e_1, e_2, S_m) \rightarrow (id, e_1, e_2, d, k)$

where Fs is a set of functions performing similarity measure, d is the similarity degree returned by match(e_1, e_2) and K is the set of mapping expressions e.g. equivalence, synonym, inclusion etc., depending upon the data model being represented by schemas 1 and 2.

Schema mapping can be uni-directional i.e., from schema 1 toward schema 2, or bidirectional i.e., the correspondence holds in both directions e.g. if an element e_1 from schema 1 is mapped to an element e_2 of schema 2 then there exists another correspondence for element e_2 of schema 2 toward element e_1 of schema 1 [**1**].

3 Motivations

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

- Terminological measures are not sufficient, for example:
  - mouse (computer device) and mouse (animal) lead to 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 discovering of more irrelevant matches
  - not efficient with small schemas

Example of schema matching: Consider the two following schemas used in [**4**]. They represent organization in universities from different countries and have been widely used in the literature.

With those schemas, the ideal set of mappings 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,*

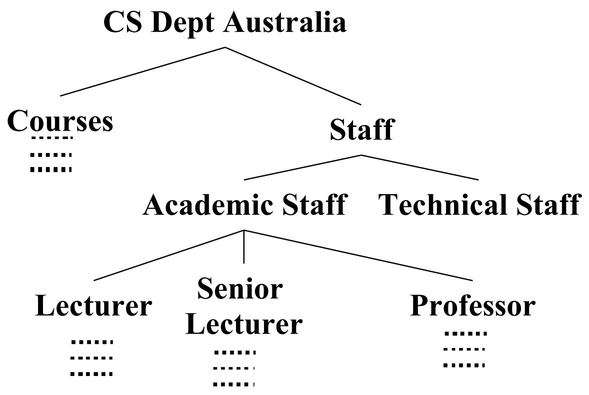

Figure 1: Schema 1: organization of an Australian university

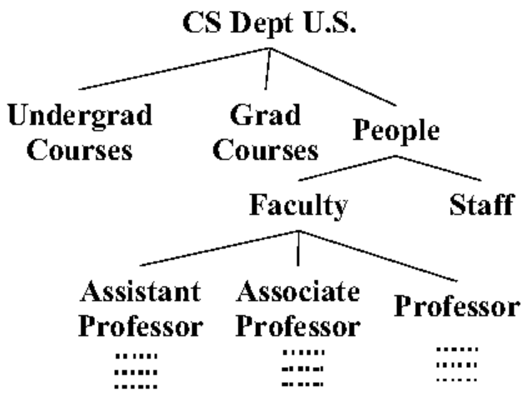

Figure 2: Schema 2: organization of a US university

associate professor), (professor, professor)}.

Let's imagine we try to find out a similarity between *Courses* and *GradCourses*. Using terminological measures, namely 3-grams and Levenhstein distance, we discover a high similarity between these labels. **StringMatching** denotes the average between 3-grams and Levenhstein similarity values and it represents the similarity obtained by terminological measures.All these measures are defined later in Section 4.1.1.

- 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 do not reveal useful to discover a match between these labels (StringMatching value of 0.002). However, the structural measures enables to match the labels with a similarity value equals to 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**. A detailed explanation of the context and the cosine measure is given in Section 5.1.

- 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**

Thus, in our approach we combine both terminological and structural measures so that we avoid the previously described problems and we ensure an acceptable matching quality.

4 Our approach: BMatch

In this section, at first, we introduce the basis of our approach: the semantic aspect which focuses on the discovery of the matches and the performance aspect, based on a B-tree indexing structure. The first aspect uses a combination of terminological and structural measures, while the second one follows the assumption that most similar labels share a common token.

4.1 Semantic Aspect

One of the contributions in our approach consists in taking into account the context of the nodes. By context of a node n, we mean the keywords, the description in natural language and the neighbouring nodes of n. As the keywords and/or description of the elements are not always available, we mainly concentrate our work on the neighbouring nodes. Indeed those lasts correspond to specific information thus such knowledge is crucial to understand the meaning of the elements. However our method works with keywords and description as well.

To compare the context from one element, we first build a vector composed of its most important neighbour elements, each of them being associated with a weight. This vector is then called *context vector*. The aim is finally to compare two context vectors of elements from different schemas in order to evaluate their semantic similarity. This similarity may be determined by using the cosine measure which enables to compare two vectors [**17**]. The cosine measure is higher (close to 1) if the terms in the two vectors tend to have a close meaning. A such measure is already used in Information Retrieval and is explained later on. In the rest of this paper, we call **CosineMeasure CM**, the cosine measure between two relative context vectors.

As explained before, two context vectors tend to be close if the terms they gather tend to be close. Yet, in the real world, those terms may be different while having character string quite close. So the idea to solve this problem is to use some terminological algorithms to replace character strings that have high lexical measures.

Here we firstly describe some important notions of our approach, the terminological measures and the context. Then BMatch's semantic aspect is explained in details. Finally, more precision is given about the parameters.

4.1.1 Terminological Measures

This part focuses on two terminological measures used in BMatch' semantic part.

n-grams

An n-gram is a sub-sequence of n items from a given sequence. 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, the n value vary between 1 and 5 and is often set to 3 [**9, 8**]. For example, consider the two character strings *dept* and *department*. Using tri-grams, we build the two sets $\{dep, ept\}$ and $\{dep, epa, par, art, rtm, tme, men, ent\}$.

To measure the similarity of two elements, the following formula 1 issued from [**9**] gives a value in]0,1]:

$$Tri(c1, c2) = \frac{1}{1 + |tr(c1)| + |tr(c2)| - 2 \times |tr(c1) \cap tr(c2)|} \quad (1)$$

The number of common occurrences in these sets is 1. By applying the formula 1 on those sets, we obtain a similarity between *dept* and *department*:

$$Tri(dept, department) = \frac{1}{1+2+8-(2\times 1)} = \frac{1}{9} \quad (2)$$

Levenhstein distance

The Levenhstein distance between two strings is given by the minimum number of operations needed to transform one source string into the target other, where an operation is an insertion, deletion, or substitution of a single character. The Levenhstein distance is the measure where all operation costs are set to 1. The Levenhstein similarity, noted $LevSim$, is a formula using the Levenhstein distance, noted L, and which processes a similarity measure between two strings:

$$LevSim(c1, c2) = \max\{0, \frac{\min\{|c1|, |c2|\} - L(c1, c2)}{\min\{|c1|, |c2|\}}\} \quad (3)$$

where $ch1$ and $ch2$ are two strings. The value given by the Levenhstein similarity formula is in [0,1], with the zero value denoting a dissimilarity and 1 a total similarity. Note that in the rest of the paper, we use either the term Levenhstein similarity or Levenhstein distance.

Following is a simple example for illustrating the formula 3 to obtain the Levenhstein similarity between *dept* and *department*:

$$LevSim(dept, department) = \max\{0, \frac{\min\{4, 10\} - 6}{\min\{4, 10\}}\} = 0 \quad (4)$$

Next we describe the notion of context, used by the structural part.

4.1.2 Node Context

A specific feature of our approach is to consider the neighbour nodes. We 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. From this assumption, we calculate the weight of each node n_i according to the node n_c, which evaluates how important the context node n_i is for the node n_i. First we calculate $\Delta\ d$ which represents the difference between the level of n_c and the level of n_i:

$$\Delta d = |lev(n_c) - lev(n_i)| \quad (5)$$

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

$$\omega(n_c, n_i) = \begin{cases} \omega_1(n_c, n_i), & if Anc(n_c, n_i) \text{ or } Desc(n_c, n_i) \\ \omega_2(n_c, n_i), & otherwise \end{cases} \quad (6)$$

where $Anc(n, m)$ (resp. $Desc(n, m)$) is a boolean function indicating if 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 concerned nodes. If n is an ancestor or a descendant of m, the formula 7 is applied. Else we apply formula 8. The idea behind this weight formula is based on the fact that the closer in the tree two nodes are, the most similar their meaning is.

$$\omega_1(n_c, n_i) = 1 + \frac{K}{\Delta d + |lev(n_c) - lev(n_a)| + |lev(n_i) - lev(n_a)|} \quad (7)$$

$$\omega_2(n_c, n_i) = 1 + \frac{K}{2 \times (|lev(n_c) - lev(n_a)| + |lev(n_i) - lev(n_a)|)} \quad (8)$$

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. It is described with more details in section 4.1.4. 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 consider the node *Academic Staff* from schema 1. We look for the importance of *Staff* for the node *Academic Staff*. As *Staff* is an ancestor of *Academic Staff*, we apply formula *7*. Δd, the difference between their levels in the tree hierarchy, is equal to 1. Their lowest common ancestor is *Staff*, and the difference of 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:

$$\omega(AcademicStaff, Staff) = 1 + \frac{1}{1+1+0} = 1.5 \quad (9)$$

Now we look for the weight of the node *Courses* with regards to *Academic Staff*. They have no ancestor or descendant relationship, so the formula 8 is applied. Their lowest common ancestor is the root node, namely *CS Dept Australia*. *Academic Staff* is 2 levels far from the common ancestor, and *Courses* is 1 level far from it. The weight of *Courses* for the node *Academic Staff* gives:

$$\omega(AcademicStaff, Courses) = 1 + \frac{1}{2 \times (2+1)} = 1.17 \quad (10)$$

We can then generalize to obtain the following set of couples (neighbour, associated weight) 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 in the experiments section) have influence on the context.

4.1.3 Semantic Match Algorithm

BMatch's semantic aspect is based on two steps: first we replace terms in the context vectors when they have close character strings. This step uses the Levenhstein distance and 3-grams algorithms (see Section 4.1.1). In a second time, we calculate the cosine measure between two vectors to determine if their context is close or not.

Part one: terminological measures to replace terms

The following describes in details 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 Levenhstein distance and the 3-grams. Both measures are processed and according to the adopted strategy[1], the higher 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

[1]The maximum and average strategies reveals to be a good compromise in the literature

the term with the smaller number of characters. Indeed we consider that the smaller-sized term is more general than the bigger-sized one. This assumption can be checked easily since some terms may be written singular or plural. We finally obtain after this first step the initial schemas that have possibly been modified with character string replacements.

We have also noticed the polysemia problem, where a word may have different meanings. The typical 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. However, the second part of our algorithm, by using context, enables to avoid this polysemia problem.

Part two: cosine measure applied to context vectors

In the second part of our algorithm, we traverse again the schemas - in which some string replacements may have occurred by means of step 1. 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 weight of the node. Indeed when counting the number of occurrences of a term, we multiply this number by its weight. This processing enables to calculate **CM**, the cosine measure between two context vectors, and thus the similarity between the two nodes related to these contexts too.

The cosine measure **[17]** is widely used in Information Retrieval. The cosine measure between the two context vectors, noted **CM**, is given by the following formula:

$$CM(v_1, v_2) = \frac{v_1 \cdot v_2}{\sqrt{(v_1 \cdot v_1)(v_2 \cdot v_2)}} \tag{11}$$

CM is in the interval [0,1]. A result close to 1 indicates that the vectors tend in the same direction, and a value close to 0 denotes a 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 and 2. Their respective and limited[2] context vectors, composed of couples 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.

Finally, we obtain two similarity measures, **SM** and **CM**, the first one based on terminological algorithms while the second takes into account the context. Here again, a strategy must be adopted to decide how to aggregate those similarity measures. In our approach, the maximum and the average have been chosen

[2]To clarify the example, the context has been voluntary limited thanks to the parameters

because they generally give better results in the experiments than other formulas where one of the measure is privileged. In the end of the process, BMatch considers the set of mappings all element couples whose similarity value is above a threshold given by an expert.

4.1.4 Parameters of the Semantic Aspect

Like most of the matchers, our approach include many parameters. Although this may be seen as a drawback, since a domain expert is often required to tune them, this is compensated by the fact that our application is generic and works with no dictionnary and whatever the domain or language is.

- NB_LEVELS: this parameter is used to know the number of levels, both up and down in the hierarchy, to search in 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 to have many cousin nodes - that does not have a significant importance - included in the context.
- REPLACE_THRESHOLD: this threshold is the minimum value to be reached to do 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 the weight formula 6 allows more flexibility. Indeed it represents the importance we give to the context when measuring similarities.

Given that the number of parameters is important, a such application need to be tuned correctly to give acceptable results **[15]**. In **[5]**, several experiments show the flexibility of BMatch by testing different configurations. This enabled to fix some of the parameters to default values.

This section described the semantic aspect of BMatch, based on the combination of terminological and structural measures. However, this semantic aspect suffers from the same drawback than the other matchers: low performance. This is due to the important 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 matching process by reducing the search space.

4.2 Performance Aspect

The first part of this section introduces the B-tree, an indexing structure already used in databases for accelerating query response time. Then we explain how we integrate it with the semantic part to improve the performance.

4.2.1 An Indexing Structure: the B-tree

In our approach, we use the B-tree as the main structure to locate matches and create mappings between XML tree structures. The advantage of searching for mappings using the B-tree approach is that B-tree have indexes that significantly accelerate this process. Indeed, if you consider the schemas 1 and 2, they have respectively 8 and 9 elements, 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 involving better performances.

As described in [**3**], B-trees have many features. A B-tree is composed of nodes, 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 contains a maximum of *M-1* indexes. Another feature is that the B-tree is balanced, meaning 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 access and retrieve quickly 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 databases 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.

4.2.2 Principle of the Match Algorithm

By using both the semantic aspect and the B-tree structure, the objective is to combine their main advantage: an acceptable matching quality and good performance. Contrary to most of the other matching tools, BMatch does not use a matrix to compute the similarity of each couple of elements. Instead, a B-tree, whose indexes represent tokens, is built and enriched as we parse new schemas, and the discovered mappings are also stored in this structure. The tokens reference all labels which contains it. For example, after parsing schemas 1 and 2, the **courses** token would hold three labels: **courses** from schema 1, **grad courses** and **undergrad courses** from schema 2. Note that the labels **grad courses** and **undergrad courses** are also stored respectively under the **grad** and the **undergrad** tokens.

For each input XML schema, the same algorithm is applied: the schema is parsed element by element by preorder traversal. This enables 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, in which case we try to find semantic similarities between the current label and the ones 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, 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 into the B-tree. Later on, we parse **grad courses** in schema 2. After 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 **courses** label is found) to calculate the String Measure denoted SM between them and **grad courses**. BMatch can replace one of the label by the other 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 an **undergrad** token structure to be created. The second token is already in the B-tree, and it contains the two labels previously added: **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. So the index enables 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 performs recursively the same action for the descendants nodes, thus enabling to add the children nodes to the context. Once all descendants have been processed, similarities might be discovered by comparing the label with tokens' references using the cosine and the terminological measures. A parameter can be set to extend the search to the whole B-tree if no mappings has been discovered.

Let us carry on our example. After processing **undergrad courses**, we should go on with its children elements. As it is a leaf, we then search the B-tree again for all the tokens which compose the label **undergrad courses**. Under the **undergrad** token, we find only one label, itself, so nothing happens. Under the **courses** token, only one of the three existing labels namely **courses**, is interesting (one is itself 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 context, and the aggregation of these two measures results in the semantic measure between those labels. If this semantic measure reaches the given threshold, then a mapping may be discovered.

5 Experiments

As our matching tool focuses on both quality and performance aspects, this section is organized in two parts. The first one shows that BMatch provides an acceptable quality of matching. The second part deals with the performance, and large and numerous schemas are matched to evaluate the benefit of the B-tree. BMatch parameters have been tuned with the following configuration, which provided optimal results in [**5**]: REPLACE_THRESHOLD is equal to 0.2, MIN_WEIGHT and NB_LEVELS are respectively set to 1.5 and 2 to limit the context, K to 1 SIM_THRESHOLD equals to 0.15. This last parameter is empiric (determined by experiments) and it indicates that all the matches discovered by BMatch with a similarity below 0.15 have been discarded. For these experiments we used 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 BMatch.

5.1 Semantic Aspect

In this part, BMatch is compared on schemas **1** and **2** with another matching tool reputed to provide an acceptable matching quality: COMA++. This matcher uses 17 similarity measures to build a matrix between every couple of elements to finally aggregate the similarity values and extract mappings. It is described later in Section 6. BMatch obtained mappings are shown in tables 1 and 2. The first table is for the maximum (MAX) strategy, in which the similarity is the maximum between StringMatching and CosineMeasure. On the contrary, the second table gathers the results obtained with the average (AVG) strategy. For COMA++, all its strategies have been tried and the best obtained results are shown in the following table 3. Note that in all tables, a *+* in the relevance column indicates that the mapping is relevant.

In table 1, we notice that the second mapping between *CS Dept Australia* and *People* is irrelevant. However, the relevant mappings are also discovered with a lower similarity value: *CS Dept Australia* with *CS Dept U.S.* on line 4 and *Staff* with *People* on line 7. BMatch is currently not able to determine if one of the mappings should be removed or not. Indeed, some complex mappings can be discovered, for example *Courses* with both *Grad Courses* and *Undergrad Courses* on line 3 and 5. Applying a strategy to detect complex mappings and re-

Element from schema 1	Element from schema 2	Similarity value	Relevance
Professor	Professor	1.0	+
CS Dept Australia	People	0.46	
Courses	Grad Courses	0.41	+
CS Dept Australia	CS Dept U.S.	0.36	+
Courses	Undergrad Courses	0.28	+
Academic Staff	Faculty	0.25	+
Staff	People	0.23	+
Technical Staff	Staff	0.21	+
Senior Lecturer	Associate Professor	0.16	+

Table 1: Mappings with BMatch with MAX strategy between schemas **1** and **2** (similarity threshold set to 0.15)

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 2: Mappings with BMatch with AVG strategy between schemas **1** and **2** (similarity threshold set to 0.15)

Element from schema 1	Element from schema 2	Similarity value	Relevance
Professor	Professor	0.53545463	+
Technical Staff	Staff	0.5300107	+
CS Dept Australia	CS Dept U.S.	0.52305263	+
Courses	Grad Courses	0.5041725	+
Courses	Undergrad Courses	0.5041725	+

Table 3: Mappings with COMA++ between schemas **1** and **2**

	Precision	Recall	F-measure
COMA++	1	0.56	0.72
BMatch (MAX)	0.89	0.89	0.89
BMatch (AVG)	0.87	0.78	0.82

Table 4: Quality measures of COMA++ and BMatch

move irrelevant ones is part of ongoing research. With the average strategy shown in table 2, less relevant mappings are discovered. However, the irrelevant mapping between *CS Dept Australia* and *People* has a lower similarity, and is now ranked after the relevant mapping between *CS Dept Australia* and *CS Dept U.S.*.

Finally, table 4 shows the matching quality of both matchers thanks to commonly used measures in the literature, namely precision, recall and f-measure. Precision calculates the proportion of relevant mappings extracted among extracted mappings. Another typical measure is recall which computes the proportion of relevant mappings extracted among relevant mappings. F-score is a tradeoff between precision and recall. COMA++ found 5 mappings on the 9 relevant similarities, implying that 4 mappings are never discovered. The recall is 0.56, the precision is obviously 1 since the extracted list gives only the relevant similarities. We obtain a F-measure equal to 0.72. As it discovers most of the relevant mappings, BMatch obtains a higher F-score than COMA++, whatever strategy is adopted. However, it seems that the maximum strategy is able to discover more mappings. Other experiments have shown that in most configurations[3], BMatch's F-measure is equal or above 0.73. These configurations enable to discover between 7 and 9 relevant similarities, compared to the 5 given by COMA++. Besides, BMatch is able to have a recall equal to 1 in some configurations, which means that all relevant mappings are discovered. This is not the case with COMA++ which has forgotten almost half of the mappings.

[3]Configuration means the tuning of the parameters

5.2 Performance Aspect

A matching tool that ensures good performance could be used in large scale scenario, or on the Internet where numerous quantities of data sources are available. Since there is no matching tool available which is dedicated to large scale scenarii, we compare our BMatch application with a BMatch version without any indexing structure. In this case, the matching algorithm tries to match every couples 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 context of a node is limited to its direct parent and its children nodes. Although this constraint could be removed, it has been shown in the quality experiments (see Section 5.1) that the context should not include too many further nodes which could have a bad impact on the quality.

The table 5 shows the different features of the sets of schemas we used in our experiments. Two large scale scenarii are presented: the first one involves a thousand of average-sized schemas about business-to-business e-commerce, taken from the XCBL[4] standards. In the second case, we deal with OASIS[5] schemas which are also business domain related. We use only sev-

[4]www.xcbl.org
[5]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 5: Characterization of the schema sets

eral hundreds of those schemas because they are quite large, with an average of 2000 nodes per schema.

5.2.1 XCBL Scenario

Here we compare the performances 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 3. We can see that the version without indexing structure is efficient when the number of schemas is not very large (less than 1600). BMatch method provides good performance with a larger number of schemas, since two thousand schemas are matched in 200 seconds.

5.2.2 OASIS Scenario

In this scenario, we are interested by matching large schemas, with an average of 2000 nodes. The graph depicted in Figure 4 shows that the version without indexing structure is not suited for large schemas. On the contrary, BMatch is able to match an important number of large schemas in less than one minute. The graph also shows that BMatch is quite linear. Indeed, it has been tested for 900 schemas, and BMatch needs around 130 seconds to perform the matching.

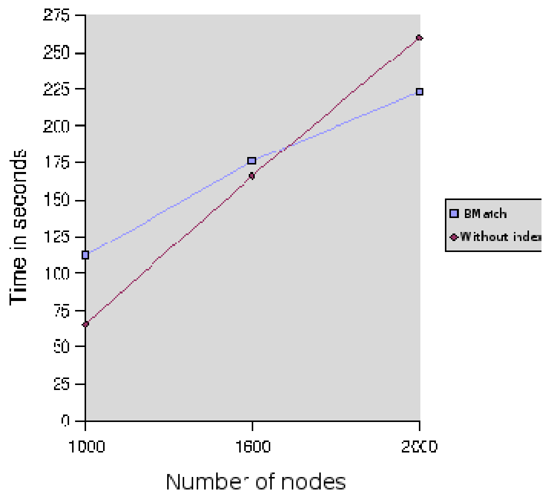

Figure 3: Matching time with XCBL schemas depending on the number of nodes.

5.3 Comparison With other Matchers

Now we compare the performance of three matching tools: BMatch, COMA++ and Similarity Flooding.

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

Table 6: Features of the different scenarii.

Four real-world scenarii composed of two schemas each are used: the first one describes a person, the second is related to university courses, the third one on a business order and the last one comes from the biology domain. Their main features are given by table 6.

Table 7 depicts the matching performance of each matching tool for each scenario. All match-

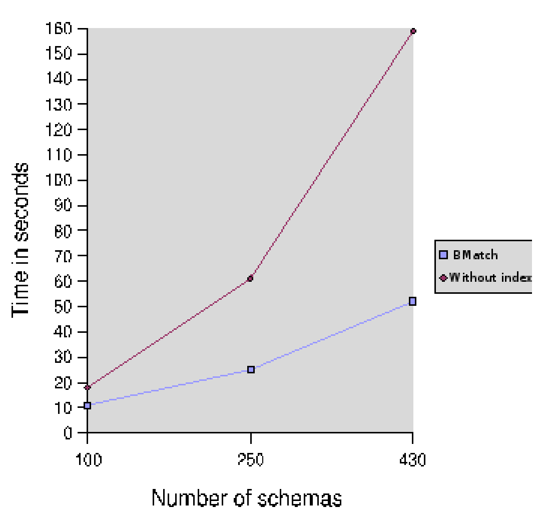

Figure 4: Matching time with OASIS schemas depending on the number of schemas.

	Person		University		Order		Biology	
NB	S1	S2	S1	S2	S1	S2	S1	S2
Nodes	11	10	18	18	20	844	719	80
COMA++	≤ 1 s		≤ 1 s		3 s		4 s	
SF	≤ 1 s		≤ 1 s		2 s		4 s	
BMatch	≤ 1 s		≤ 1 s		≤ 1 s		2 s	

Table 7: Matching performance of COMA++, Similarity Flooding and BMatch on the different scenarii.

ers are able to match the small schemas in less than one second. However, with larger schemas, COMA++ and Similarity Flooding are less efficient. On the other hand, BMatch still ensures good performance.

5.4 Discussion

In this section, we have conducted some experiments to demonstrate both the quality and the performance of BMatch. Our matching tool has been compared with COMA++, and we have shown that BMatch provides an acceptable matching quality : both matching tools obtain a F-score slightly above 0.7. However, COMA++ discovers only half of the relevant mappings (thus involving a 0.56 recall) while BMatch has a 0.89 recall. BMatch also performed well in the performance aspect: the B-tree indexing structure enables to match 430 schemas in 50 seconds, contrary to the BMatch version without indexing structure which needs 160 seconds. Thus BMatch seems to be suitable for a large scale scenario.

6 Related Work

This section covers related work in both the schema matching and the similarity measures domains.

6.1 Schema matching tools

In the literature, many schema matching approaches [**10, 1, 12, 16, 7, 13, 19**] have been studied at length. Most of them have been designed to demonstrate their benefit in different scenarii. However, the currently matching tools employ techniques for mapping two schemas with human intervention.

6.1.1 COMA++

As described in [**1**], COMA++ is a hybrid matching tool that can incorporate many independent matching algorithms. Different strategies, for example the reuse-oriented matching or the 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 from the *matcher library*, the required match algorithms. 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 mapping possibilities and the user checks and validates their accuracy.

The advantage of COMA++ is the good matching quality and the ability to re-use mappings, while supporting many formats and ontologies. During the match process or at the end of the process, the user has the final decision to choose the appropriate mappings since COMA++ has done most of the work in selecting the potential matches. New matching algorithms can be added and the list of synonyms can be completed, thus offering advantages for specific field areas. It is also a good platform to evaluate and compare new matching algorithms.

However the weak point of COMA++ is the time required, both for adding the files into the repository and to match 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 the matching of many schemas directly.

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++ is able to match only two schemas at a time.

6.1.2 Similarity Flooding

Similarity Flooding is an algorithm described in **[12]** and is based on structural approaches. Input schemas are converted into directed labeled graphs and the aim is to find relationships between those graphs. The structural rule used is the following: 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 the adjacent nodes until there is no changes anymore. As in most of matchers, Similarity Flooding generates mappings for the nodes having a similarity value above a certain threshold.

This algorithm mainly exploits the labels with some semantic-based algorithms, like String Matching, to determine the nodes to which it should propagate. Similarity Flooding has been implemented through the Rondo matching tool. Finally, it supports different formats like XML Schema and relational database schemas.

Similarity Flooding does not give good results when labels are often identical, especially for polysemic terms. Thus involving wrong mappings to be discovered by propagation.

BMatch uses the same structural rule stating that two nodes from different schemas are similar if most of their neighbour 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 mappings.

6.1.3 An Approach for Large Schemas Based on COMA++

To the best of our knowledge, **[14]** is the only one previous work dealing with large schemas, using COMA++ tool [**1**]. In this work, first, the user divides the schema into fragments and then each fragment from source schema is mapped to target schema fragments, to find inter-fragment matching. Next, these fragment mappings are merged to compute the schema level mappings. Thus, the tool is not able to process directly large schemas. Another issue of this approach [**14**] is which criteria are the best for fragmenting the large schemas.

To conclude, the existing tools are semi-automatic and are designed for small schemas. Moreover, they did not focus on the performance aspect, while our method has the following properties:

- semantic aspect ensuring an acceptable matching quality
- designed for processing large schemas
- scalable

6.2 Similarity Measures

To calculate the semantic similarity between two labels, there exists many measures which are often cited in the literature [**7, 2, 11**].

6.2.1 Jaro Winkler distance

The Jaro-Winkler distance [**18**] is a measure of similarity between two strings. It is a variant of the Jaro distance metric.

As Jaro-Winkler, with its character comparison and its transpositions, is quite close to **n-grams** and **levenhstein distance**, thus we use **n-grams** and **levenhstein distance** in our approach.

6.2.2 Maedche et al.

In [**11**], an approach to measure the similarity between two (parts of) ontologies is proposed. These measures are both lexical and conceptual and compute how one ontology covers the other. The lexical part is ensured by the String Matching, which is mainly based on Levenhstein distance. The conceptual part refers to the upwards cotopy, which gathers the super- and sub-concepts of a given concept while taking into account the position in the hierarchy of this given concept. An experiment section enables to study the efficiency of the proposed measures on real-world scenario.

The lexical layer only relies on Levenhstein distance, therefore can be quite deceptive in some cases. For example, two labels, like *tower* and *power*, have totally different meanings but they will be considered as very similar. The cotopy measure could be compared to our context notion, however the relationships between elements of an ontology are predefined. On the contrary, relationships between elements of schema are not as explicit, thus our context needs to compute some weights to measure the importance of neighbour elements. Besides, limiting the cotopy to the super- and sub-concepts is restrictive.

6.2.3 Ehrig et al

The authors present in [**6**] a framework to measure similarities between ontologies. It is based on three layers: the data layer, which aims at discovering similarities between instances. The edit distance is the measure used by this layer. This also enables to detect duplicates values that could be mismatched.

The second layer is the ontology layer in which semantic relationship are considered. Several heuristics can determine the similarity of two entities by comparing their hierarchies, the specific relationships like restrictions, etc. Some similarity measures from the data layer can also be useful here, for example to provide initial similarities. Finally, the context layer relies on the following assumption: *similar entities are used in similar contexts.* This context gathers the user's resources, i.e the stored information and the executed queries. An amalgamation function is in charge of computing the similarity results of the three layers, and some experiments with Bibster application are described.

Even if this approach and BMatch uses an amalgamation function to combine several similarity values, both methods differ from the fact that in the schema matching domain, the data and an ontology are rarely available for different reasons: difficulties to build an ontology, privacy or lack of data... Thus, using these resources are often optional while trying to match schemas. Concerning the context, Ehrig et al gather within this notion all information around the user (queries, stored information) while BMatch consider the context as the most important[6] neighbour nodes of a given node in a schema. Both approaches are definitely different and it seems difficult to compare the results of their experiments.

7 Concluding Remarks

In this paper, we have presented our BMatch approach to improve the time elapsed on schema matching. Our approach deals with both the semantic aspect by relying on terminological and structural measures and performance aspect by using an indexing structure, the B-tree. Moreover, our method is not language-dependent and it does not rely on dictionaries or ontologies. It is also quite flexible with different parameters.

For the semantic aspect, the terminological approaches enable to discover elements represented by close character strings. On the other hand, the structural measures compare the contexts of two elements using the cosine measure. Experiments have shown that BMatch and COMA++ matching quality are similar, with a F-score above 0.7.

To evaluate the benefit provided by the index structure, we made some comparisons between BMatch alone and BMatch without any indexing structure. 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. Note that the B-tree can directly store the mappings into memory. Furthermore, experiments also showed that BMatch is able to match large schemas faster than COMA++ or Similarity Flooding.

The results of our experiments are very interesting and showing that 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.

[6]Importance with regards to semantics

References

[1] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *ACM SIGMOD Conference, DEMO paper*, pages 906–908, 2005.

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

[3] D. Comer. The ubiquitous btree. In *Computing Surveys*, 1979.

[4] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Ontology matching: A machine learning approach. In *Handbook on Ontologies, International Handbooks on Information Systems*, 2004.

[5] F. Duchateau, Z. Bellahsene, and M. Roche. A context-based measure for discovering approximate semantic matching between schema elements. In *RCIS*, pages 9–20, 2007.

[6] M. Ehrig, P. Haase, and N. Stojanovic. Similarity for ontologies - a comprehensive framework. In *Proc. of Practical Aspects of Knowledge Management*, 2004.

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

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

[9] D. Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann, San Francisco, CA, 1998.

[10] J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB01*, 2001.

[11] A. Maedche and S. Staab. Measuring similarity between ontologies. In *Proc. of the European Conference on Knowledge Acquisition and Management - EKAW*, pages 251–263, 2002.

[12] S. Melnik, H. G. Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. of the International Conference on Data Engineering (ICDE'02)*, 2002.

[13] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.

[14] E. Rahm, H. Do, D. Aumueller, and S. Massmann. Matching large xml schemas. In *ACM SIGMOD Record 33(4):26-31*, 2004.

[15] M. Sayyadian et al. Tuning schema matching software using synthetic scenarios. In *Proceedings of the 31th VLDB Conference*, 2005.

[16] J. Tranier, R. Barar, Z. Bellahsène, and M. Teisseire. Where's charlie: Family-based heuristics for peer-to-peer schema integration. In *Proc of IDEAS*, pages 227–235, 2004.

[17] R. Wilkinson and P. Hingston. Using the cosine measure in a neural network for document retrieval. In *Proc of ACM SIGIR Conference*, pages 202–210, 1991.

[18] W. Winkler. The state of record linkage and current research problems. In *Statistics of Income Division, Internal Revenue Service Publication R99/04*, 1999.

[19] M. Yatskevich. Preliminary evaluation of schema matching systems. Technical Report DIT-03-028, Informatica e Telecomunicazioni, University of Trento, 2003.