

**Université Montpellier II**

**Rapport pour le Stage du Master 2 Recherche D'Informatique**

---

**Rapport : Intégration sémantique données :**

**Using contextual and lexical information to map terms of schemas**

---

**Stagiaire : Tang Yi Fei**

**Encadrement : Zohra Bellahsène**

**Mathieu Roche**

## Abstract

There are many methods for estimate of similarity between elements in the schemas, but traditional similarity measures may sometimes fail to estimate terms similarity correctly.

One of the traditional methods for calculating terms similarity is string based technique. It is typically based on the following intuition: the more similar the terms, the more likely they denote the same concepts. It rely on character edit operations, such as deletions, insertions, substitutions and subsequence comparison(n-grams), The best-known string - based similarity metric is Levenshtein distance, defined as the minimum number of insertions, deletions or substitutions necessary to transform one string into another.

While string - based metrics work well for estimating distance between terms that differ due to typographical errors or abbreviations, they become computationally expensive and less accurate for larger strings. When differences between equivalent terms are expressed by very different lexical forms, string -based technique may assign high cost to non-matching strings, producing low similarity scores for equivalent strings with a few lexical differences. So we have proposed a novel context – based method to avoid this problem by viewing the context of the string as a document and disregarding the order in which the element occur in the document. Then we used the vector space method for computing the similarity of documents to calculate the similarity between context vectors.

The conventional algorithm for measuring similarity of documents can not represent term–term correlations in the vector. All co-occurrence words collected are not distinguishing whether they are syntactically or semantically related; the similar relation between the members of the vectors is ignored.

So we proposed a method we call **Algorithm Approxivect** to solve this problem, which can find the pairs of similar members in the context vectors.

The first experimental results show that our novel context – based measure with our algorithm **Approxivect** can improve the context accuracy over traditional techniques and ameliorate the results with string – based measure.

**Résumé :**

Beaucoup de méthodes de recherche de similitudes entre les éléments de schémas existent. Les mesures traditionnelles s'appuient sur la similitude entre les chaînes de caractères. De telles mesures sont fondées sur l'intuition suivante : plus les chaînes de caractères caractérisant les termes sont semblables, plus ces termes sont proches « sémantiquement ». Le calcul de ces mesures s'appuie sur des opérations de suppressions, d'insertions, de substitutions et de comparaisons de subséquences (n-grammes). Une des mesures souvent utilisée dans la littérature est la distance de Levenshtein définie comme le nombre minimum d'insertions, de suppressions et de substitutions nécessaires pour transformer une chaîne de caractères en une autre.

Par exemple, une telle mesure peut se révéler performante pour trouver la similitude avec des abréviations (comme : int/integer).

Cependant une telle technique peut assigner une distance élevée entre des termes sémantiquement proches mais ayant des chaînes de caractères très différentes (comme : car/auto).

Ainsi, nous avons proposé une nouvelle approche pour éviter un tel problème en étudiant le contexte des chaînes de caractères. Dans le cadre de la mise en correspondances d'éléments de schémas, le contexte d'un élément peut représenter les sous-arbres d'un élément (contexte descendant), les mots clés le caractérisant, etc. Dans notre approche, chaque élément sera représenté par un vecteur. Ce dernier représente le contexte (éléments des sous-arbres, etc.) de l'élément

L'algorithme **Approxivect** que nous avons proposé pour résoudre ce problème consiste à déterminer la similitude entre deux vecteurs afin de déterminer la proximité sémantique entre les éléments. Une telle approche s'inspire des techniques de recherche de similitudes entre documents. L'algorithme **Approxivect** prend également en compte la proximité des chaînes de caractères des contextes ainsi que la proximité sémantique de ces derniers établie avec des connaissances du domaine (WordNet).

Les premiers résultats expérimentaux montrent que l'algorithme **Approxivect** améliore les résultats comparativement aux approches uniquement fondées sur la recherche de similitude entre chaînes de caractères.

<b>I</b>	<b>The Schema Matching Problem</b>	7
<b>II</b>	<b>State of the Art of schema mapping</b>	
	Classification of Matching Approaches	9
2.1	Combined matcher	10
2.2	Individual matcher.	11
2.2.1	Structure-level techniques	12
2.2.2	Element-level techniques	14
2.2.2.1	Language-based techniques.	14
2.2.2.2	Semantic based - Linguistic resources.	14
2.2.2.3	Constraint-based techniques	15
2.2.2.4	String-based techniques.	15
<b>III</b>	<b>State of the art of Similarity measure for documents</b>	
3.1	Vectorial representation with frequency of the words	20
3.2	Measure cosine...	21
<b>IV</b>	<b>My algorithm: Computing the word's similarity using approximate descendents context vectors</b>	
4.1	What is contextual similar?	23
4.2	Why we use context based technique?	24
4.3	The steps for computing contextual similarity of two words in the schemas.	25
4.3.1	Step 1: Tree travel - Look up all the descendents (sub tree) to construct the context vector.	26
4.3.1.1	Introduction.	26
4.3.1.2	Travel in – width.	26
4.3.1.3	Travel in – depths.	26
4.3.2	Step 2 Algorithm Approxivect	27
4.3.2.1	Algorithm approxivect 1	28
4.3.2.2	Algorithm approxivect 2.	30
4.3.3	Step3	32
4.3.4	Step4	33
4.3.5	Step5	34
4.3.6	The whole example using algorithm Approxivect	34
4.4	WordNet-based semantic similarity measurement	42

4.4.1 WordNet. . . . .	42
4.4.2 Semantic similarity between two synsets . . . . .	42
4.4.3 The path length-based similarity measurement. . . . .	43
4.4.4 Measuring similarity. . . . .	44
<b>V Experimentation</b>	
5.1 Close lexical and close context . . . . .	46
5.2 Close lexical and distant context . . . . .	50
5.3 Distant lexical and close context . . . . .	51
5.4 Distant lexical and distant context . . . . .	54
<b>VI Conclusion and future work</b> . . . . .	55
<b>Reference</b> . . . . .	56

## REMERCIEMENTS

J'ai le plaisir de remercier tous ceux qui, par leurs encouragements, leur relecture et leur aide pratique ou simplement par leur confiance, ont contribué de façon importante à la réalisation de ce rapport de stage.

Mes remerciements vont en premier lieu à Monsieur Mathieu Roche et Madame Zohra Bellasène, qui ont accepté, sans hésitation, d'être mes directeurs pour ce stage. Bien plus que des directeurs, ils ont joué pour moi un rôle de 'mentor' dans mon stage. Ils m'ont aidé à trouver mon chemin dans le système universitaire français et ils ont toujours été prêts à me conseiller et à m'encourager. Ses aides furent précieuses à bien des égards.

Merci à tous mes collègues, Je ne peux pas tous les mentionner individuellement, mais je tiens à remercier explicitement : Merci beaucoup pour une grande partie de mes progrès linguistiques.

## I The Schema Matching Problem

**Schema (definition):** A schema consists of a set of related *elements*, such as tables, columns, classes, or XML elements or attributes.

**Matching (definition):** Manipulation on 2 input schemas producing a result mapping that identifies corresponding elements in the two schemas.

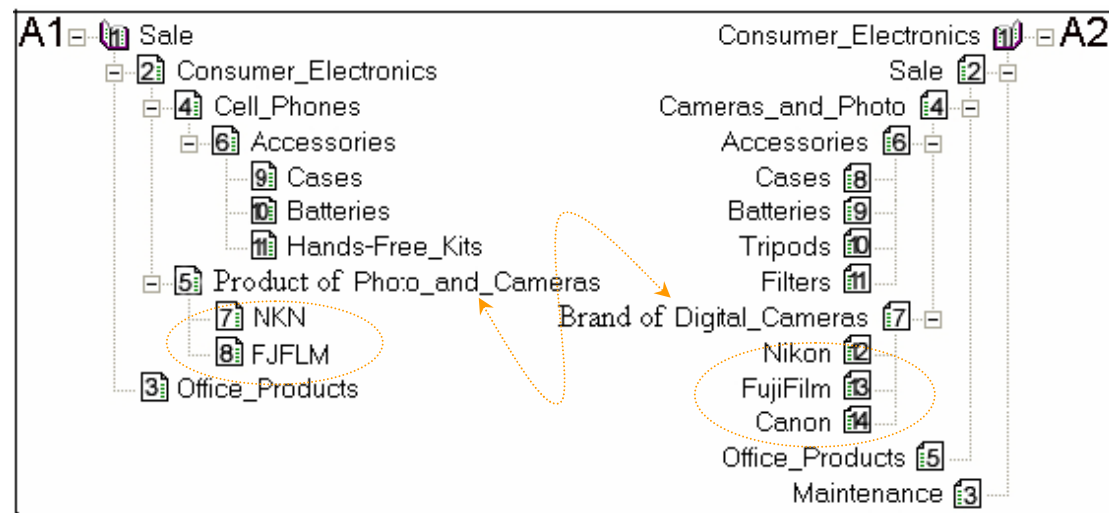


Figure 1. Two XML schemas

The result of a Match operation is a *mapping*. A mapping consists of a set of *mapping elements*, each of which indicates that certain elements of one schema are related to certain elements of other schema.

I assume that all the data and conceptual models (e.g., database schemas, taxonomies and ontologies) can be represented as graphs see for a detailed discussion [1]. Therefore, the matching problem can be decomposed in two steps:

1. Extract graphs from the data or conceptual models;
2. Match the resulting graphs.

Notice that this allows for the statement and solution of a *generic matching problem*, very much along the lines of what done in Cupid [2], and COMA [3].

Let us define the notion of matching graphs precisely. *Mapping element* is a 4-tuple  $\langle ID_{ij}, n_{1i}, n_{2j}, R \rangle$ ,  $i=1, \dots, N1$ ;  $j=1, \dots, N2$ ; where  $ID_{ij}$  is a unique identifier of the given mapping element;  $n_{1i}$  is the  $i$ -th node of the first graph,  $N1$  is the number of nodes in the first graph;  $n_{2j}$  is the  $j$ -th node of the second graph,  $N2$  is the number of nodes in the second graph; and  $R$  specifies a *similarity relation* of the given nodes. A *mapping* is a set of mapping elements. *Matching* is the process of discovering

mappings between two graphs through the application of a matching algorithm.

To motivate the matching problem, let us use two simple XML schemas that are shown in Figure 1 and exemplify one of the possible situations which arise, for example, when resolving a schema integration task.

Suppose an e-commerce company A1 needs to finalize a corporate acquisition of another company A2. To complete the acquisition we have to integrate databases of the two companies. The documents of both companies are stored according to XML schemas A1 and A2 respectively. Numbers in boxes are the unique identifiers of the nodes. A first step in integrating the schemas is to identify candidates to be merged or to have taxonomic relationships under an integrated schema. This step refers to a process of schema matching.

For example, in Figure 1, the nodes with labels *Office\_Products* in A1 and in A2 are obviously the candidates to be merged, and the node with label 'Brand of *Digital\_Cameras*' in A2 should be merged by the node with label 'Product of *Photo\_and\_Cameras*' in A1, but with the matching approach 'string based measure' they have a little difference in the word's forms that puzzled me to decide whether we should construct un mapping between them. (We will do a detailed introduction for matching approach at the next chapter.) In this paper, I will propose a method to solve this problem. According to the above example, we can first make a simple introduction for our method.

In figure 1, 'Product of *Photo\_and\_Cameras*' have two descendents: *NKN*, *FJFLM*. 'Brand of *Digital\_Cameras*' has three descendents: *Nikon*, *FujiFilm*, *canon*. After the compare of similarity measure we can find that: *NKN* and *Nikon* are synonyms, *FJFLM* and *FujiFilm* are synonyms too. So we can say that the descendents of 'Product of *Photo\_and\_Cameras*' and 'Brand of *Digital\_Cameras*' are similar. Thus we can get a result that 'Product of *Photo\_and\_Cameras*' and 'Brand of *Digital\_Cameras*' are similar because of their similar descendents. We will discuss it in detail at the succeeding chapter.

In general, a mapping element may also have an associated expression that specifies its semantics (called a *value correspondence*). For example, the expression of mapping between *NKN* and *Nikon* might be "*Sale.Consumer\_Electronics.Product of*

*photo\_and\_Cameras.NKN = Consumer\_Electronics.sale.cameras\_and\_photos.Brand of Digital\_Cameras.Nikon*” We do not treat such expressions in my paper. Rather, we only address *mapping discovery*, which returns mapping elements that identify related elements of the two schemas. Since we are not concerned with mapping expressions, we treat mappings as non-directional. Schema matching is inherently subjective. Schemas may not completely capture the semantics of the data they describe, and there may be several plausible mappings between two schemas (making the concept of a single best mapping ill-defined). This subjectivity makes it valuable to have user input to guide the match and essential to have user validation of the result. This guidance may come via an initial mapping, a dictionary or thesaurus, a library of known mappings, etc. Thus, the goal of schema matching is: *Given two input schemas in any data model and, optionally, auxiliary information and an input-mapping, compute a mapping between schema elements of the two input schemas that passes user validation.*

## II State of the Art of Schema mapping

*Match* is an important operator in many well-known application domains, such as, semantic web, schema/ontology integration, data warehouses, e-commerce, XML message mapping, catalog matching, etc. Many solutions to the matching problem include identifying terms in one information source that ”match” terms in another information source. The applications can be viewed as graph-like structures containing terms and their inter-relationships. These might be database schemas, taxonomies, or ontology, for example [4], etc. Match operator takes two graph-like structures as input and produces a mapping between the nodes of the graphs that correspond semantically to each other as output.

### Classification of Matching Approaches

At present, there exists schema matching systems, a good survey on schema matching approaches, up to 2001, is provided by Erhard Rahm and Phil Bernstein in [5] and presented in Figure 1.

The classification distinguishes between *individual* implementations of match and *combinations* of matches. Individual matchers comprise *instance-based* and *schema-based*, *element-* and *structure-level*, *linguistic-* and *constrained-based* matching techniques. Also *cardinality* and *auxiliary information* (e.g., dictionaries,

global schemas, etc.) can be taken into account. Individual matchers can be used in different ways: directly (*hybrid matchers*), or combining the results of independently executed matchers (*composite matchers*).

In my stage I focus only on **schema-based approaches** in the individual match and **simple hybrid matchers** in the combined matchers, and therefore consider only schema information, not instance data.

Let us discuss them in more detail. There are two levels of granularity while performing schema-based semantic and also syntactic matching: element-level and structure-level. Element-level matching techniques compute mapping elements by analyzing individual labels/concepts at nodes; structure-level techniques compute mapping elements by analyzing also subgraphs.

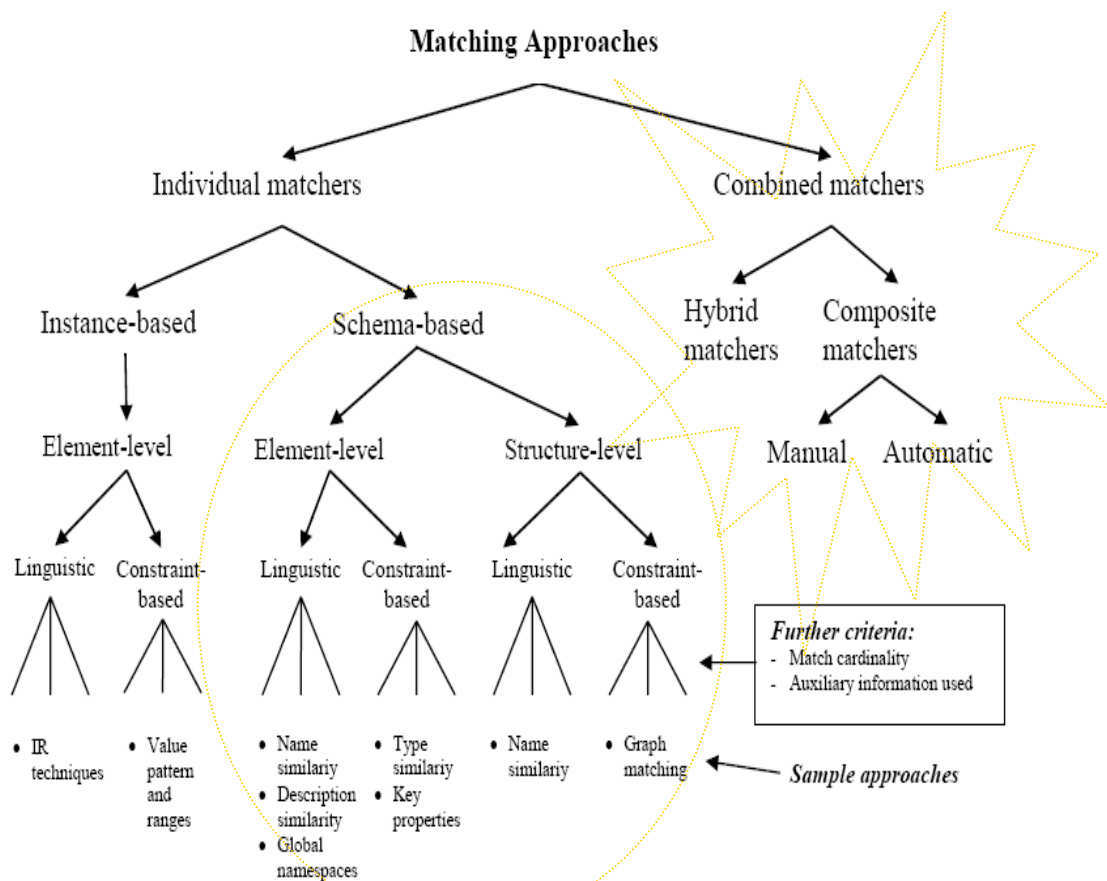


Figure 2. Matching Approaches

## 2.1 Combined matchers

We have reviewed several types of matchers and many different variations. Each utilizes different information and has thus different applicability and value for a given

match task. Therefore, a matcher that uses just one approach is unlikely to achieve as many good match candidates as one that combines several approaches.

*Hybrid matchers* directly combine several matching approaches to determine match candidates based on multiple criteria or information sources. They should provide better match candidates plus better performance than the separate execution of multiple matchers. Effectiveness may be improved because poor match candidates matching only one of several criteria can be filtered out early, and because complex matches requiring the joint consideration of multiple criteria can be solved.

*Individual vs. Combinational* – An individual matcher uses a single algorithm to perform the match. Combinational matchers can be one of two types: *Hybrid matchers* use multiple criteria to perform the matching. *Multiple matchers* run independent match algorithms on the two schemas and combine the results.

In a word, Single algorithm is not good enough. A hybrid approach is better than a single algorithm.

## 2.2 Individual matchers

*Element-level vs structure-level.* Element-level matching techniques compute mapping elements by analyzing entities in isolation, ignoring their relations with other entities. Structure-level techniques compute mapping elements by analyzing how entities appear together in a structure. This criterion is the same as first introduced in [5]. My classification for schema based individual matchers is represented as follow:

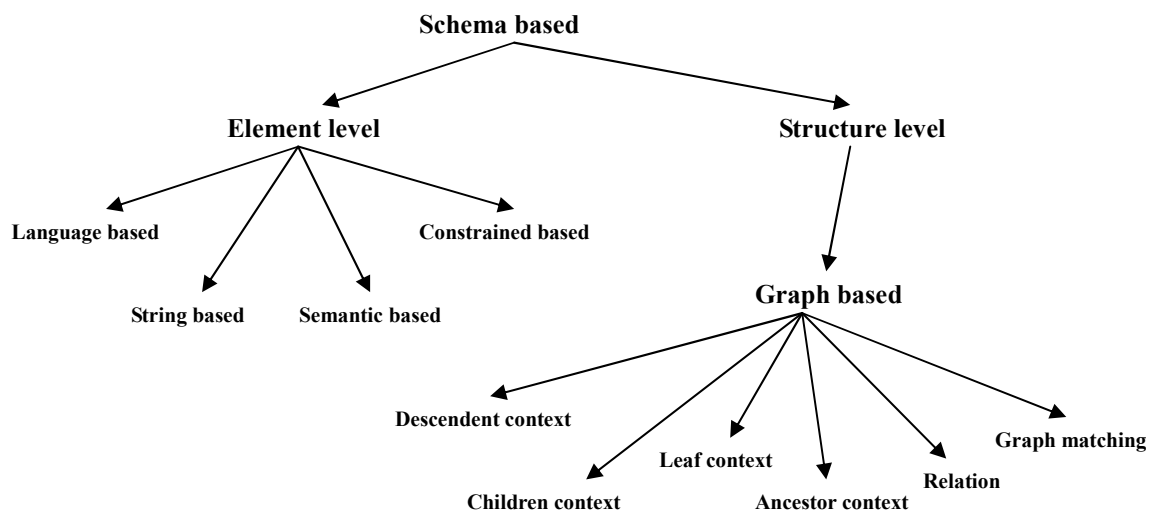


Figure 2\* my classification for individual matcher

## 2.2.1 Structure-level techniques

**Graph-based techniques** are graph algorithms which consider the input as labeled graphs. The applications (e.g., database schemas, taxonomies, or ontologies) are viewed as graph-like structures containing terms and their inter-relationships. Usually, the similarity comparison between a pair of nodes from the two schemas is based on the analysis of their positions within the graphs. The intuition behind is that, if two nodes from two schemas are similar, their neighbors might also be somehow similar. Below, we present some particular matchers representing this intuition.

*Graph matching.* There have been done a lot of work on graph (tree) matching in graph theory and also with respect to schema matching applications, see, for example, [6, 7]. Matching graphs is a combinatorial problem that can be computationally expensive. It is usually solved by approximate methods. In schema matching, the problem is encoded as an optimization problem (finding the graph matching minimizing some distance like the dissimilarity between matched objects) which is further resolved with the help of a graph matching algorithm. This optimization problem is solved through a fix-point algorithm (improving gradually an approximate solution until no improvement is made). Examples of such algorithms are [8] and [9]. Some other (particular) matchers handling tree's problems are context technique: *children context*, *leaves context*, *ancestor context* and *relations*.

*Context technique:*

- *Leaves context.* The (structural) similarity between inner nodes of the graphs is computed based on similarity of leaf nodes, that is, two non-leaf schema elements are structurally similar if their leaf sets are highly similar, even if their immediate children are not, see, for example [2, 3].
- *Children context.* The (structural) similarity between inner nodes of the graphs is computed based on similarity of their children nodes, that is, two non-leaf schema elements are structurally similar if their immediate children sets are highly similar. A more complex version of this matcher is implemented in [3].
- *Ancestor context.* The (structural) similarity between inner nodes of the graphs is computed based on similarity of ancestor nodes, that is, two schema elements are structurally similar if their ancestor nodes sets are highly similar.
- *Relations.* The similarity computation between nodes can also be based on their relations. For example, in one of the possible ontology representations of schemas of Figure 1, if class Photo and Cameras relates to class NKN by relation hasBrand in one ontology, and if class Digital Cameras relates to class Nikon by relation

has- Marque in the other ontology, then knowing that classes Photo and Cameras and Digital Cameras are similar, and also relations hasBrand and hasMarque are similar, we can infer that NKN and Nikon may be similar too, see [10].

People have distinguished three kinds of node *contexts* depending on its position in the schema graph: (1) *The ancestor-context*: of a node  $n$  is defined as the path (going through containment edges) having  $n$  as its ending node and the root of the schema graph as its starting node. The ancestor-context of the root node is empty and it is assigned a NULL value; (2) *The Children-context*: of a node  $n$  includes its immediate sub-elements (through containment edges). The children context of a node reflects its basic structure and its local composition. The children-context of an atomic node is assigned a NULL value. And (3) *The leaves context*: Leaves XML documents represent the atomic data that the document describes. The leaves-context of a node  $n$  includes the leaves of the subtrees (composed by containment relationships) rooted at  $n$  and. The leaf-context of an atomic node is assigned a NULL value.

In my paper, I will use a new context-based technique whose name is **Descendent context**. Descendent context includes all the sub elements (descendants) but not only the immediate ones like children context (people take the immediate sub elements in the children context) and leaf set like leaf context. The difference of these contexts is show in the figure 3.

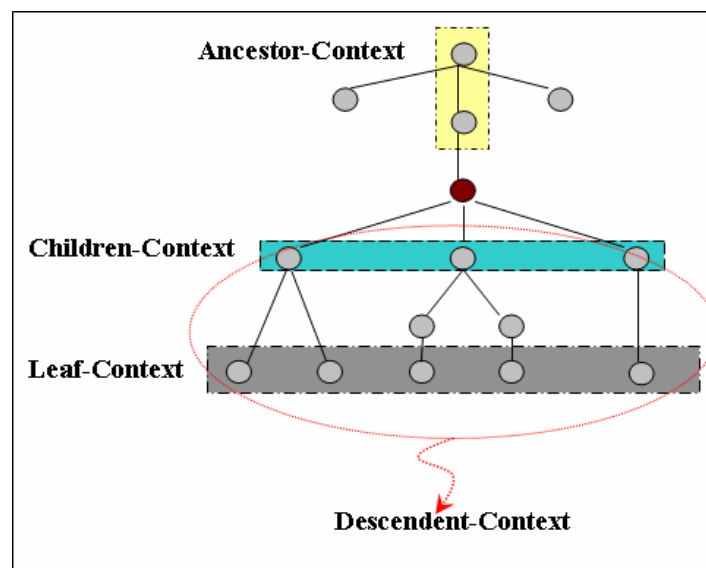


Figure 3

## 2.2.2 Element-level techniques

**2.2.2.1 Language-based techniques** consider names as words in some natural language (e.g., English). They are based on Natural Language Processing (NLP) techniques exploiting morphological properties of the input words.

– *Tokenization*. Names of entities are parsed into sequences of *tokens* by a tokenizer which recognizes punctuation, cases, blank characters, digits, etc. (e.g., Hands- Free Kits → `_hands, free, kits_`, see, for example [11]).

– *Lemmatization*. The strings underlying tokens are morphologically analyzed in order to find all their possible basic forms (e.g., Kits→ Kit), see, for example [11].

– *Elimination*. The tokens that are articles, prepositions, conjunctions, and so on, are marked (by some matching algorithms, e.g., [2]) to be discarded. Usually, the above mentioned techniques are applied to names of entities before running string-based or lexicon-based techniques in order to improve their results. However, we consider these language-based techniques as a separate class of matching techniques, since they can be naturally extended, for example, in a distance computation (by comparing the resulting strings or sets of strings).

**2.2.2.2 Semantic based - Linguistic resources** such as common knowledge or domain specific thesauri are used in order to match words (in this case names of schema/ontology entities are considered as words of a natural language) based on linguistic relations between them (e.g., synonyms, hyponyms).

– *Common knowledge thesauri*. The approach is to use common knowledge thesauri to obtain meaning of terms used in schemas/ontologies. For example, WordNet [12] is an electronic lexical database for English (and other languages), where various *senses* (possible meanings of a word or expression) of words are put together into sets of synonyms. Relations between schema/ontology entities can be computed in terms of bindings between WordNet senses, see, for instance [13, 11]. For example, in Figure 1, a sense-based matcher may learn from WordNet (with a prior morphological preprocessing of labels performed) that Camera in *A1* is a hypernym for Digital Camera in *A2*, and, therefore conclude that entity Digital Cameras in *A2* should be subsumed by the entity Photo and Cameras in *A1*. Another type of matchers exploiting thesauri is based on their structural properties, e.g., WordNet hierarchies. In particular,

hierarchy-based matchers measure the distance, for example, by counting the number of arcs traversed, between two concepts in a given hierarchy, we will give a more detail explain in the section 4.4 *WordNet-based semantic similarity measurement*. Several other distance measures for thesauri have been proposed in the literature, e.g., [14, 15].

– *Domain specific thesauri*. These thesauri usually store some specific domain knowledge, which is not available in the common knowledge thesauri, (e.g., proper names) as entries with synonym, hypernym and other relations. For example, in Figure 1, entities NKN in *A1* and Nikon in *A2* are treated by a matcher as synonyms from a domain thesaurus look up: synonym key - "NKN:Nikon = synonym", see, for instance [2].

**2.2.2.3 Constraint-based techniques** are algorithms which deal with the internal constraints being applied to the definitions of entities, such as types, cardinality of attributes, and keys. We omit here a discussion of matching keys as these techniques appear in our classification without changes with respect to the original publication [5]. However, we provide a different perspective on matching datatypes and cardinalities.

– *Datatypes comparison* involves comparing the various attributes of a class with regard to the datatypes of their value. Contrary to objects that require interpretations, the datatypes can be considered objectively and it is possible to determine how a datatype is close to another (ideally this can be based on the interpretation of datatypes as sets of values and the set-theoretic comparison of these datatypes, see [16, 17]). For instance, the datatype day can be considered closer to the datatype workingday than the datatype integer. This technique is used in [9].

– *Multiplicity comparison* attribute values can be collected by a particular construction (set, list, multiset) on which cardinality constraints are applied. Again, it is possible to compare the so constructed datatypes by comparing (i) the datatypes on which they are constructed and (ii) the cardinality that are applied to them. For instance, a set of between 2 and 3 children is closer to a set of 3 people than a set of 10-12 flowers (if children are people). This technique is used in [9].

**2.2.2.4 String-based techniques** are often used in order to match names and name descriptions of schema entities in various schema matching systems [2,3]. These techniques consider strings as sequences of letters in an alphabet. They are typically based on the following intuition: the more similar the strings, the more likely they denote the same concepts. A comparison of different string matching techniques, from *distance* like functions to *token-based distance* functions can be found in [18]. Usually, distance functions map a pair of strings to a real number, where a smaller value of the real number indicates a greater similarity between the strings. Some examples of string-based techniques which are extensively used in matching systems are *prefix*, *suffix*, *edit distance*, and *n-gram*.

- *Prefix*. This test takes as input two strings and checks whether the first string starts with the second one. *Prefix* is efficient in matching cognate strings and similar acronyms (e.g., *int* and *integer*), see, for example [2, 3, 8], but often syntactic similarity does not imply semantic relatedness. Consider examples in Table 1. The test *Prefix* produces an equivalence relation if the input labels satisfy the relation of prefix. The matcher returns equality for *hot* and *hotel* which is wrong but it recognizes the right relations in the case of the pairs *int*, *integer* and *cat*, *core*.

Input label 1	Input label 2	relation
int	integer	=
hot	hotel	=
cat	core	≠

Table 1

- *Suffix*. Suffix performs very similarly to Prefix. This test takes as input two strings and checks whether the first string ends with the second one (e.g., *phone* and *telephone*), see, for example [2,3,8]. In table 2, it correctly recognizes cognate words (*phone*, *telephone*) but makes mistakes with syntactically similar but semantically different words (*word*, *sword*).

Input label 1	Input label 2	relation
phone	Telephone	=
word	Sword	=
door	Floor	≠

Table 2

- *Edit distance (I have used this method in my algorithm)*. This distance takes as input two strings and computes the edit distance between the strings. The edit

distance is the total cost of transforming one string into another using a set of edit rules, each of which has an associated cost. It is obtained by finding the cheapest way to transform one string into another. Transformations are the one-step operations of (single-phone) insertion, deletion and substitution. In the simplest version substitutions cost about two units except when the source and target are identical, in which case the cost is zero. Insertions and deletions costs half that of substitutions. **In my algorithm, I use the Levenshtein distance for computing the value of Edit distance.**

Levenshtein distance is one kind of the Edit distance.

The process of the Algorithm Levenshtein distance is described as follows:

Step	Description
1	Set n to be the length of s. Set m to be the length of t. If n = 0, return m and exit. If m = 0, return n and exit. Construct a matrix containing 0..m rows and 0..n columns.
2	Initialize the first row to 0..n. Initialize the first column to 0..m.
3	Examine each character of s (i from 1 to n).
4	Examine each character of t (j from 1 to m).
5	If s[i] equals t[j], the cost is 0. If s[i] doesn't equal t[j], the cost is 1.
6	Set cell d[i,j] of the matrix equal to the minimum of: a. The cell immediately above plus 1: d[i-1,j] + 1. b. The cell immediately to the left plus 1: d[i,j-1] + 1. c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost.
7	After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[n,m].

Table 3

I will show how the Levenshtein distance is computed when the source string is "GUMBO" and the target string is "GAMBOL".

Let's see what happened in each step:

**Step 1 and 2**

		G	U	M	B	O
	0	1	2	3	4	5
G	1					
A	2					
M	3					
B	4					
O	5					
L	6					

**Step 3 to 6 when i =1**

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0				
A	2	1				
M	3	2				
B	4	3				
O	5	4				
L	6	5				

**Step 3 to 6 when i =2**

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1			
A	2	1	1			
M	3	2	2			
B	4	3	3			
O	5	4	4			
L	6	5	5			

**Step 3 to 6 when i =3**

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2		
A	2	1	1	2		
M	3	2	2	1		
B	4	3	3	2		
O	5	4	4	3		
L	6	5	5	4		

**Step 3 to 6 when i = 4**

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2	3	
A	2	1	1	2	3	
M	3	2	2	1	2	
B	4	3	3	2	1	
O	5	4	4	3	2	
L	6	5	5	4	3	

**Step 3 to 6 when i =5**

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2	3	4
A	2	1	1	2	3	4
M	3	2	2	1	2	3
B	4	3	3	2	1	2
O	5	4	4	3	2	1
L	6	5	5	4	3	2

**Step 7**

The distance is in the lower right hand corner of the matrix, i.e. 2. This corresponds to our intuitive realization that "GUMBO" can be transformed into "GAMBOL" by substituting "A" for "U" and adding "L" (one substitution and 1 insertion = 2 changes).

After obtain the value of edit distance , I subtract the obtained number of operations from minimum ( $length(label1), length(label2)$ ) and divider the result with minimum ( $length(label1), length(label2)$ )again. The formula is as follow:

$$SM(L_i, L_j) = \max \left( 0, \frac{\min(|L_i|, |L_j|) - |ed(L_i, L_j)|}{\min(|L_i|, |L_j|)} \right) \in [0, 1]$$

With the above formula, the value of string matching – based similarity between "GUMBO" and "GAMBOL" is  $(5-2) / 5=0,6$

I assume that all relation scores are in the [0, 1] range, which means that if the score gets a maximum value (equal to 1) then the two string are absolutely similar.

If the value exceeds a given threshold (0.6 by default) the equivalence relation is returned, otherwise,  $\neq$  (not equal) is produced.

Let’s see the matching examples in table 4:

Input label 1	Input label 2	relation
street	street1	=
proper	propel	=
owe	woe	$\neq$

**Table 4**

Edit Distance is useful with some unknowns to WordNet labels. For example , it can easily match labels *street1, street2, street3, street4* to *street* (edit distance measure is

0.83). In the case of matching *proper* with *propel* the edit distance similarity measure has 0.83 value, but equivalence is obviously the wrong output.

– ***N-gram (I have used this method in my algorithm)***. This test takes as input two strings and computes the number of common n-grams between them.

An N-gram is an N-character slice of a longer string. Although in the literature the term can include the notion of any co-occurring set of characters in a string (e.g., an N-gram made up of the first and third character of a word), in my Algorithm I use the term for contiguous slices only. Typically, one slices the string into a set of overlapping N-grams. In my algorithm, I use N-grams of several different lengths simultaneously. We also append blanks to the beginning and ending of the string in order to help with matching beginning-of-word and ending-of-word situations. (I will use the underscore character (“\_”) to represent blanks.) Thus, the word “TEXT” would be composed of the following N-grams: bi-grams: \_T, TE, EX, XT, T\_

tri-grams: \_TE, TEX, EXT, XT\_, T\_\_

quad-grams: \_TEX, TEXT, EXT\_, XT\_\_ , T\_\_\_

In general, a string of length k, padded with blanks, will have k+1 bi-grams, k+1 tri-grams, k+1 quad-grams, and so on. N-gram-based matching has had some success in dealing with noisy input in other problem domains, such as in interpreting postal addresses, in text retrieval, and in a wide variety of other natural language processing applications. The key benefit that N-gram-based matching provides derives from its very nature: since every string is decomposed into small parts, any errors that are present tend to affect only a limited number of those parts, leaving the remainder intact. If we count N-grams that are common to two strings, we get a measure of their similarity that is resistant to a wide variety of textual errors.

Input label 1	Input label 2	relation
address	address1	=
contract	contact	=
door	floor	≠

**Table 5**

In the past, Ngram and Edit Distance have been applied to matching database and XML schemas [3] [2]. Both matchers are effective in matching acronyms and words with minor syntactic differences like (address, address1).

**Our intuition (confirmed by the preliminary testing results) is that *string based***

matchers are better suited for XML/relational schema matching. They work better with acronyms and abbreviations, which are quite common in this domain.

Although string based matchers are useful for some unknown to Wordnet, we can not trust it completely, because it sometimes gives a false comparison result. (such as : ‘hot’ and ‘hotel’ in table 1 with the method prefix, ‘word’ and ‘sword’ in table 2 with the method suffix, ‘proper’ and ‘propel’ in table 4 with the method edit distance, ‘contract’ and ‘contact’ in the table 5 with the method n-grams. These pairs of words are all mistakenly considered as similar with the string based matcher ) We will discuss how we could improve the measure result of string – based matcher with our algorithm in detail on the succeeding chapter.

### III State of the art of Similarity measure for documents

#### 3.1 Vectorial representation with frequency of the words

To classify or compare documents, one can use a vectorial representation of documents, then measurements of distances and similarity. A document is a succession of words (or labels, or of letters or...). A vector is a succession of numbers. There are two types of vectorial representation: complete, who takes into account the frequency of the words, and the other which takes into accounts the presence or the absence of the words. Here we adopt the first one which considers the frequency.

Let us see an example:

$$D1 = \{ I, I, will \}$$

$$D2 = \{ I, I, I, I, will, will \}$$

$$D3 = \{ I, will, will \}$$

$$D4 = \{ will, will \}$$

The vocabulary of these documents consists of two words: ‘I’, ‘will’.

In the first D1 document, the word ‘I’ appears 2 times, the word ‘will’ 1 time. The V1 vector will comprise this information.

In the second document, the word ‘I’ appears 4 times, the word ‘will’ 2 times....

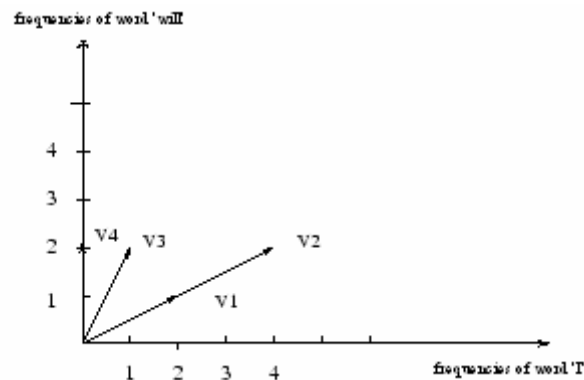
$$V_1 = \{2, 1\}$$

$$V_2 = \{4, 2\}$$

$$V_3 = \{1, 2\}$$

$$V_4 = \{0, 2\}$$

Figure 3 illustrates the vectorial representation with the frequencies of words of the documents D1 to D4. The axis horizontal is associated the word “I” and the vertical axis with the word “will”. The space of representation has two dimensions<sup>1</sup>.



**Figure 3:** vectorial representation of documents D1, D2, D3 et D4.

Texts which resemble each other contain the same words or the words which appear in similar contexts (cf. the assumption of Harris: the words which have identical contexts are similar, Harris, Z. (1968). *Mathematical Structures off Language*, Wiley, New York.).

If one places oneself in the vector space, two similar documents correspond to close vectors. To gather close vectors is to find the vectors which have similar directions or to which the ends are close (NB: the word order does not have importance in the vectorial representation).

### 3.2 Measure cosine

Cosine similarity is a common vector based similarity measure. Whereby the input string is transformed into vector space so that the cosine rule can be used to determine similarity.

For this measurement, the complete vectorial representation is used, i.e. with the frequency of the words. Their vectors of two documents form an angle whose cosine is worth:

$$\cos \alpha = \cos(\mathbf{V}_1, \mathbf{V}_3) = \frac{\mathbf{V}_1 \cdot \mathbf{V}_3}{\|\mathbf{V}_1\| \|\mathbf{V}_3\|}$$

Scalar product  $\mathbf{V}_1 \cdot \mathbf{V}_3$  divided by the product of the norm of  $\mathbf{V}_1$  multiplied by the norm of  $\mathbf{V}_3$ ; the norm of a Vector = the length of a Vector.

Two documents are similar if the value of angular cosine between their vectors is less than a threshold, otherwise they are not similar.

:Figure 4 illustrates the vectorial representation with the frequencies of words of the documents D1 to D4 and the angle which they form.

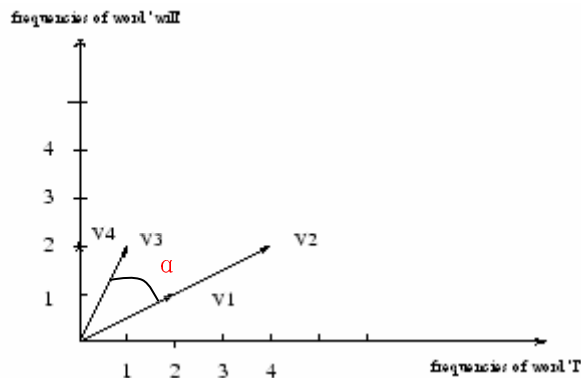


Figure 4

**Rewrites itself in a space with two dimensions:**

$$V_1 = \{a_1, a_2\} = \{2, 1\}$$

$$V_3 = \{c_1, c_2\} = \{1, 2\}$$

$$\cos \alpha = \cos(V_1, V_3) = \frac{a_1 c_1 + a_2 c_2}{\sqrt{a_1^2 + a_2^2} \sqrt{c_1^2 + c_2^2}}$$

**Spreads in a space of dimension 2 to L:**

$$V = \{a_1, \dots, a_j, \dots, a_L\}$$

$$V' = \{b_1, \dots, b_j, \dots, b_L\}$$

$$\cos \alpha = \cos(V, V') = \frac{\sum_{j=1}^L a_j b_j}{\sqrt{\sum_{j=1}^L a_j^2} \sqrt{\sum_{j=1}^L b_j^2}}$$

Exemple :

$$D2 = \{I, I, I, I, will, will\}$$

$$D3 = \{I, will, will\}$$

$$V_2 = \{b_1, b_2\} = \{4, 2\}$$

$$V_3 = \{c_1, c_2\} = \{1, 2\}$$

$$\cos(V_2, V_3) = \frac{b_1 c_1 + b_2 c_2}{\sqrt{b_1^2 + b_2^2} \sqrt{c_1^2 + c_2^2}} = \frac{(4 * 1) + (2 * 2)}{\sqrt{16^2 + 2^2} \sqrt{1^2 + 2^2}} = \frac{8}{\sqrt{20} \sqrt{5}} = \frac{8}{2(\sqrt{5} \sqrt{5})} = \frac{8}{10} = \frac{4}{5}$$

This value of cosine corresponds to an angle of 37 degrees.

The probably most often used similarity measure compares two documents by using a weighted histogram of the words they contain [23, 24]. Specifically, the “term

frequency” weighing scheme works as follows: it counts the frequency of occurrence of a term in a document in relation to the word’s occurrence frequency in a whole corpus of documents. The resulting word counts are then used to compose a weighted term vector describing the document. The similarity between the two documents is then computed as the cosine between their respective weighted term vectors.

Vectorial representation of documents is very common in information retrieval [23, 24] or machine learning [25]. They represent each object as a vector of features in a k-dimensional space and compute the similarity by measures such as cosine. We can adapt the technique of Vectorial representation of documents in the domain of schema mapping. We look at the aggregation of all the descendents of a node in a schema as the object setting by representing it as a k-dimensional vector. Here k is the number of unique words and the length of the k-th component of the vector is associated with the object part frequency in the objects. The similarity between two objects’ vectors is now simply defined as their inner product.

#### IV My algorithm: **Computing the word’s similarity using approximate descendents context vectors**

Many tasks require determining whether two knowledge representations encode the same knowledge. Solving this matching problem is hard because representations may encode the same content but differ substantially in form. Previous approaches to this problem have used syntactic measures which we called string – based method, such as edit distance, n-grams .We have showed that this is not enough because mismatches between representations go largely unaddressed in the section **2.2.2.4 String-based techniques**. We know that single algorithm is not good enough and a hybrid approach is better than a separate execution of single algorithm. So, in this section, we will show how our context matching can augment existing string based approaches to further improve matching.

If the value of context based similarity measure is greater than a threshold, we call it contextual similar.

##### 4.1 What is contextual similar?

**The Contextual similar in my paper (definition):** The Strong or Weak Contextual

Hypothesis of [20] says that the similarity of the contextual representation of two words determines (fully or partially) the semantic similarity of those words. That is, more similar words are, more similar context they have.

Two schema elements are similar if the vectors composed by their all descendents are similar. This is because the descendents represent the contextual environment that the schema ultimately describes. The rationale is that descendents with highly similar ancestors occur in similar contexts. So the presence of such similar contexts should reinforce their similarity of ancestors. We can say that the similar context could increase the similarity of two schema elements. For same reasons, distant context could decrease the similarity of two schema elements. It is a little like the structural children similar which we have presented in the section classification of matching approach. The difference is that our contextual similar consider all descendents but not only the children.

**Context similarity** is the similarity between two nodes in the schemas based on the Vector Space Model [22]. The approach is also used in [21]. A word is defined by its contexts. Similar individual contexts belong to the same word sense. His rationale is derived by parsing the contextual environment of each word. That is, two nodes in the schemas with the similar context tend to have the same meaning and be the mapping ones. We get the term frequency of each term in the context (sub tree) vector of each string. The degree of similarity between a pair of words is then computed by some similarity or distance measure that is applied to the corresponding pairs of vectors. Our weight of this metric is the Cosine similarity between the two vectors, which is a value between 0 and 1 since no negative counts are allowed.

## 4.2 Why we use context based technique?

Because the string based techniques are not enough in the cases as follow:

- Two words are close at the aspect of lexical but distant at the semantic level.  
(Typical example in this case is polysemous: Single word can express different senses: bank  $\equiv$  financial institute  $\neq$  bank  $\equiv$  border of a river.)
  
- Two words are distant at the aspect of lexical but close at the semantic level.  
(typical example in this case is synonyms: A word does not directly express a concept, and several words may express the same concept.)

Using our context based technique can resolve the above problems: we can use context based technique to disambiguate the words by verifying whether the context of a word is highly consistent with the context of the other word.

### 4.3 The steps for computing contextual similarity of

#### two words in the schemas with our algorithm:

- **Step 1:** Find all the descendents of each word (context of sub tree ) we are preparing compare, put the descendents in the tables that are called context vectors and be stored as arrays
- **Step 2:** Using our algorithm Approxivect to changer the context vectors
- **Step 3 - 5** Computing the similarity between the contexts we have created in the step 2. The similarity of the incoming context vectors can be computed by the means of cosine angles between the two vectors to determine if two contexts are similar or not. Suppose that the context vectors are  $C_i$  and  $C_j$ , we can use the formula as follows:

$$\text{Similarity } (C_i, C_j) = \frac{V_i * V_j}{\|V_i\| \|V_j\|}$$

Where  $V_i$  is the vector representation of  $C_i$ ; \* represents the inner product between two vectors;  $\|V_i\|$  represents the norm of a vector  $V_i$ . Based on the formula, two context vectors with the same words set will have the highest value of similarity between them because the inner product of the two context vectors will be one while two contexts without any word in common will have the lowest similarity: zero.

**Step 3:** A vocabulary is built from all the words in the context vectors changed

**Step 4:** Each context of sub tree is represented as a vector based against the vocabulary and broken down into a word frequency table

**Step 5:** Computing the similarity between the context vectors using measure Cosine

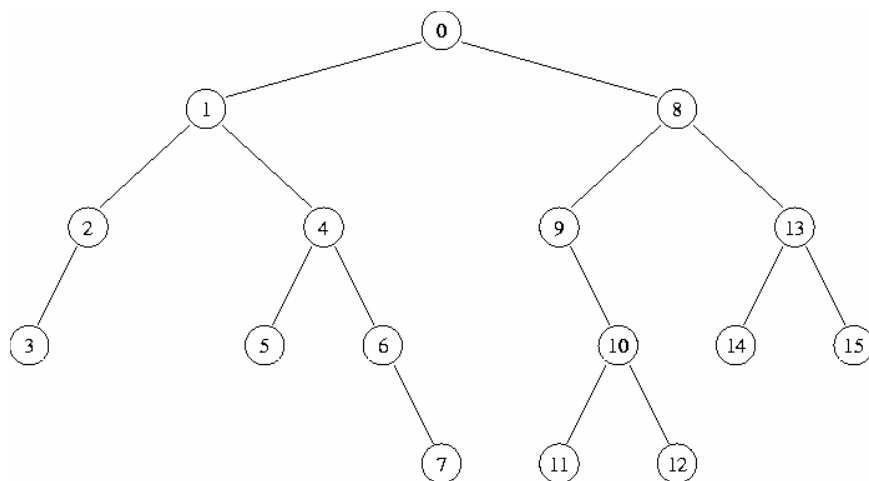
I do not want to say more than is needed for step 3, step 4, and step 5 because they are the same with the techniques for measure of documents.

### 4.3.1 Step1: Look up all the descendents (sub tree) to construct the context vector using tree travel technique

#### 4.3.1.1 Introduction

If we want to find the sub tree of a node in a schema tree, we should first find the node in the schema tree. We can use Tree Travel technique to do that. A tree travel is a way of ordering the nodes of a tree in order to traverse them. One can see it as a function which with a tree associates a list of its nodes even if the list is not often explicitly built by the travel.

One distinguishes primarily two types of travel: the travel in-width and the travel in-depth. Among in-depth travel, one distinguishes the travel again: the prefix travel, the infix travel and the suffix travel.



**Figure 5**

#### 4.3.1.2 Travel in - width

##### Definition and example

The travel in width consists in traversing the tree level by level. The nodes of level 0 are initially traversed then the nodes of level 1 and so on. In each level, the nodes are traversed left towards the line. The travel in width of the above tree traverse nodes in the order [0,1,8,2,4,9,13,3,5,6,10,14,15,7,11,12].

#### 4.3.1.3 Travel in - depths

In-depth travels are defined in a recursive way on the trees. The travel of a tree consists in treating the root of the tree and recursively traversing the sub-trees left and right of the root. The travel prefix, infix and suffix are characterized by the order in

which these treatments are made.

### Definitions and examples

In the travel prefixes, the root is treated before the recursive calls on the sub-trees left and right. The travel prefixes above tree traverses the nodes in the order [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15].

In the travel infix, the treatment of the root is made between the calls on the sub-trees left and right. The infix travel of the above tree traverses the nodes in the order [3,2,1,5,4,6,7,0,9,11,10,12,8,14,13,15].

In the travel suffix, the root is treated after the recursive calls on the sub-trees left and right. The travel suffix of the above tree traverses the nodes in the order [3,2,5,7,6,4,1,11,12,10,9,14,15,13,8,0]. The process is represented graphically in the figure 6.

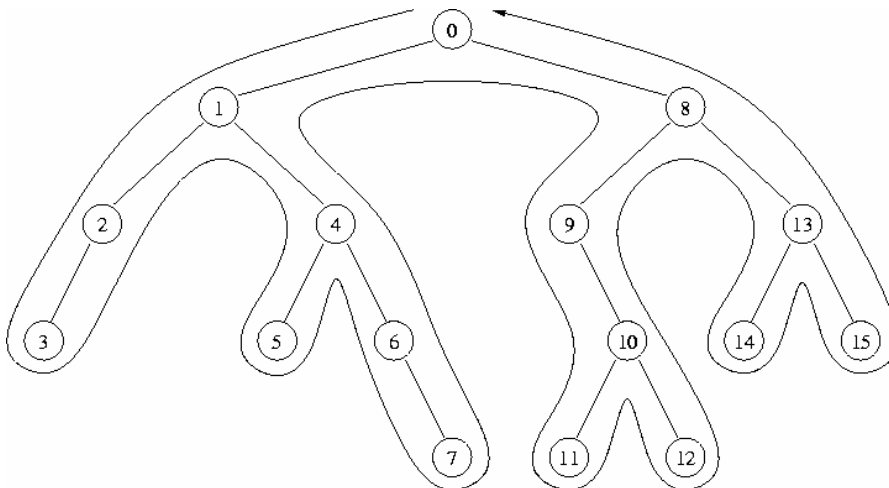


Figure 6

After find the node we want, we can use the class defined in a program language to find all his descendents. In a language like C#, it is often useful to have a class allowing an in-depth travel of a tree. The travel class defines an object which traverses a generic schema tree. From the travel class we can easily know how much and what are a node's descendents.

#### 4.3.2 Step 2: Using our algorithm Approxivect to changer the context vectors

Using the conventional measurement for computing the similarity of documents to measurer the context vectors suffers from the following drawback: The information in

the context vectors is vague. It cannot represent term–term correlations. All co-occurrence words collected are not distinguishing whether they are syntactically or semantically related. The result of similarity measure between the context vectors is often weak. For instance, vector 1 contain two member [enseignant, téléphone] and vector 2 have two member [enseignante, téléphonie ], obviously we can see that these two vectors are very similar because their members ‘enseignant’ and ‘enseignante’, ‘téléphone’ and ‘téléphonie’ are very close in the lexical form in which case they often have the same sense. But with the conventional algorithm for measuring similarity of documents I have explained **in the section III State of the art of similarity measure for documents**, the similar relation between the members of the vectors is ignored and the value of similarity measure between two above vectors is 0, so these two vectors are not considered as similar ones.

In this paper, I proposed a method I call **Algorithm Approxivect** to solve this problem, which can find the pairs of similar members in the context vectors and practice replacement between them. The algorithm compares two entities in the vectors to find out whether the edit-distance (string matching) or tri-grams between the two entities is within the allowed threshold. If the edit-distance is within the allowed threshold  $R$ , a replacement between two entities happens. We can adjust the thresholds  $R$  to obtain the best replacements. The advantage of our algorithm is that it can obtain a better value of similarity measure between the context vectors than the conventional algorithm if they have the similar word’s members. Our algorithm Approxivect has two parts: Approxivect 1 and Approxivect 2. Let us see how it works.

---

#### 4.3.2.1 Algorithm approxivect 1

*Input : context vector  $V_1 = [A_i, I \in (0, N)$ ,  $A_i$  is the element of vector  $V_1$ , where  $N$  is the number of the elements in the vector  $V_1$ ]*

*context vector  $V_2 = [B_o, O \in (0, M)$ ,  $B_o$  is the element of vector  $V_2$ , where  $M$  is the number of the elements in the vector  $V_2$ ]*

*Output : approximate vector  $V_1$ ’ similar to vector  $V_1$ .*

*approximate vector  $V_2$ ’ similar to vector  $V_2$ .*

*MeasureSimilarity.String-Based =  $1/2(\text{Measure.StringMatching} + \text{Measure.Tri})$*

*MeasureSimilarity.Linguistic-based = MeasureSimilarity.Wordnet*

*R is the value of threshold which is initialized by the users.*

*For each element  $A_i$  in vector  $V_1$*

*For each element  $A_j$  in vector  $V_1$  ( $J > I$ )*

*If the valor of similarity measure of string matching between  $A_i$  and  $A_j$  is bigger than a threshold  $R$ , moreover  $A_i$  and  $A_j$  they are not the same ones.*

*//MeasureSimilarity.String-Based ( $A_i, A_j$ ) >  $R$  &&  $A_i \neq A_j$*

*Then*

*If the length of  $A_i$  is smaller than the one of  $A_j$   $\| |A_i| < |A_j|$*

*Then*

*Replace  $A_j$  with  $A_i$   $\| A_j = A_i$*

*Else*

*Replace  $A_i$  with  $A_j$   $\| A_i = A_j$*

*End if*

*End if*

*End for*

*End for*

*For each element  $B_o$  in vector  $V_2$*

*For each element  $B_j$  in vector  $V_2$  ( $J > O$ )*

*If the valor of similarity measure of string matching between  $B_j$  and  $B_o$  is bigger than a threshold  $R$ ,  $B_j$  and  $B_o$  they are not the same ones.*

*//MeasureSimilarity.String-Based ( $B_j, B_o$ ) >  $R$  &&  $B_j \neq B_o$*

*Then*

*If the length of  $B_j$  is smaller than the one of  $B_o$   $\| |B_j| < |B_o|$*

*Then*

*Replace  $B_o$  with  $B_j$   $\| B_o = B_j$*

*Else*

*Replace  $B_j$  with  $B_o$   $\| B_j = B_o$*

*End if*

*End if*

*End for*

*End for*

---

There are not only the members existing respectively in two different vectors which can be similar, but the members in one same vector which may be also similar. Thus, before we check up if there are the similar members existing respectively in two

vectors, we should first look for the similar words in every individual vector.

The output of algorithm **approxivect 1** is the approximate vector of each input vector which is generated using the string – based similarity. For each member in a vector, if the member with higher weighted similarity to another member in the same vector is acceptable ( the string – based similarity exceed the threshold  $R$  ), then the replacement between two members is returned.(we use the shorter member to replace the other one ,because ordinarily the more short a word is , the more possible it represent general form.)

We see an example:

Input vector  $V_1$  [ enseignante , école , enseignant ]

vector  $V_2$  [enseignants,université, nom ,enseignement]

Output vector  $V_1'$  [ enseignant , école , enseignant ]

vector  $V_2'$  [enseignants, université ,nom, enseignants]

In this small example, threshold  $R$  which we establish is equal to 0.6. We compare the members two by two in vector  $V_1$ : the value of similarity measure between ‘enseignante’ and ‘école’ is 0.1, being smaller than  $R$ . the value of similarity measure between ‘école’ and ‘enseignant’ is 0.1, still smaller than  $R$ . Only the value of similarity measure between ‘enseignante’and ‘enseignant’ is 0,7 and greater than  $R$ . According to our algorithm **Approxivect 1**, only when the value of similarity measure is greater than the threshold  $R$ , we can carry on the replacement, so the replacement occur between ‘enseignante’ and ‘enseignant’, and that the word ‘enseignante’ is longer than the word ‘enseignant’, therefore we choose to replace ‘enseignante’ with ‘enseignant’. With the same principle, we replaces ‘enseignement’ with ‘enseignants’ in vector  $V_2$ , because only the value of similarity measure between ‘enseignement’ with ‘enseignants’ is 0,65 which is greater than the threshold  $R$ .

#### 4.3.2.2 Algorithm approxivects 2

*Input: two approximate context vectors ( the output vectors at the end of step 1)*

*$V_1$  [  $A_i, I \in ( 0 , N )$  ,  $A_i$  is the element of vector  $V_1$ ,where  $N$  is the number of*

*the elements in the vector  $V_1$ ]*

*$V_2'$  [  $Bo, O \in (0, M)$  ,  $Bo$  is the element of vector  $V_2$ , where  $M$  is the number of the elements in the vector  $V_2$ ]*

*Output : approximate vector  $V_1'$  similar to vector  $V_1$*

*approximate vector  $V_2'$  similar to vector  $V_2$ .*

*MeasureSimilarity.String-Based =  $1/2(\text{Measure.StringMatching} + \text{Measure.Tri})$*

*MeasureSimilarity.Linguistic-based = MeasureSimilarity.Wordnet*

*$R$  is the value of threshold which is initialized by the users.*

*For each element  $A_i$  in vector  $V_1'$*

*For each element  $Bo$  in vector  $V_2'$*

*If the value of similarity measure of string matching between  $A_i$  and  $Bo$  is bigger than a threshold  $R$  , moreover  $A_i$  and  $Bo$  they are not the same ones.*

*//MeasureSimilarity.String-Based ( $A_i, Bo$ ) >  $R$  &&  $A_i \neq Bo$*

*Then*

*If the length of  $A_i$  is smaller than the one of  $Bo$  //  $|A_i| < |Bo|$*

*Then*

*Replace  $Bo$  with  $A_i$  //  $A_j = A_i$*

*Else*

*Replace  $A_i$  with  $Bo$  //  $A_i = Bo$*

*End if*

*End if*

*End for*

*End for*

---

Like algorithm **Approxivect 1**, the output of algorithm **Approxivect 2** is also the approximate vector which is generated using the string – based similarity measure, the difference is that the replacement of elements this time don't happen in the individual vector but between two vectors. The replacement of element in a vector in the algorithm **Approxivect 1** is based on the vector itself, but the replacement of element in a vector in the algorithm **Approxivect 2** is based on not only the vector itself but also the other vector. The difference between the algorithm **Approxivect 1** and the algorithm **Approxivect 2** is represented graphically in the figure 7.

We see an example:

Input two vectors :  $V_1'$  [ enseignant , école , enseignant ]

$V_2'$  [ enseignants, université, nom,enseignants]

Output two vectors:  $V_1''$ [ enseignant , école , enseignant ]

$V_2''$ [enseignant,université,nom,enseignant]

For this example, threshold R is still 0.6. We seriatim compare each member in vector  $V_1'$  with the elements in the vector  $V_2'$ . For instance, the word 'enseignant' in the vector  $V_1'$ , it has a similarity measure 0,8 with the word 'enseignants' in the vector  $V_2'$ , 0,2 with the word 'université' in the vector  $V_2'$ , 0 with the word 'nom' in the vector  $V_2'$ , 0,8 with the other 'enseignants' in the vector  $V_2'$ , so we replace two 'enseignants' with 'enseignant' because of their high similarity measure bigger than the threshold R.

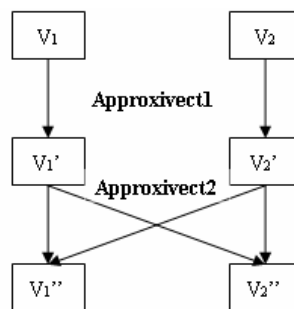


Figure 7

**4.3.3 / step 3/:** A vocabulary is built from all the words in the context vectors changed

Input two vectors:  $V_1''$ [  $A_i, I \in ( 0 , N )$  ,  $A_i$  is the element of vector  $V_1$ ,where  $N$  is the number of the elements in the vector  $V_1$ ]

$V_2''$ [  $B_o, J \in ( 0 , M )$  ,  $B_o$  is the element of vector  $V_2$ ,where  $M$  is the number of the elements in the vector  $V_2$ ]

Output one vocabulary  $V$  [  $C_y, Y \in ( 0 , P )$  ,  $C_y$  is the element of vector  $V$  ,where  $P$  is the number of the elements in the vector  $V$ ],vector  $V$  contain all the elements used in the vector  $V_1$  and vector  $V_2$ .

Set up : vector  $V = \{\emptyset\}$

For each element  $A_i$  in vector  $V_1$

If vocabulary  $V$  does not contain  $A_i$  //  $A_i \notin V$

Then

Add  $A_i$  to  $V$

End if

---

```

End for
For each element Bo in vector V2
  If vocabulary V does not contain Bo // Bo ∉ V
    Then
      Add Bo to V
    End if
  End for

```

---

Let's see an example:

Input two vectors:  $V_1$  [enseignant , école , enseignant ]

$V_2$  [enseignant , école , nom]

Output Vocabulary :  $V$  [enseignant, école , nom]

---

**4.3.4 / step 4 /:** Each context of sub tree is represented as a vector based against the vocabulary and broken down into a word frequency table

*Input :  $V_1$  [  $A_i, I \in (0, N)$  ,  $A_i$  is the element of vector  $V_1$ , where  $N$  is the number of the elements in the vector  $V_1$ ]*

*$V_2$  [  $B_o, J \in (0, M)$  ,  $B_o$  is the element of vector  $V_2$ , where  $M$  is the number of the elements in the vector  $V_2$ ]*

**Vocabulary  $V$**  [  $C_y, Y \in (0, P)$  , where  $P$  is the number of the elements in the vector  $V$ ]

vocabulary  $V$  contain all the elements used in the vector  $V_1$  and vector  $V_2$ .

*Output: the vector  $V_1$  contain the word frequency of each element of vector  $V$  in  $V_1$   
the vector  $V_2$  contain the word frequency of each element of vector  $V$  in  $V_2$*

*Set up :*

*vector  $V_1$  [  $D_y = 0, Y \in (0, P)$  where  $P$  is the number of the elements in the vector  $V$ ]*

*vector  $V_2$  [  $E_y = 0, Y \in (0, P)$  where  $P$  is the number of the elements in the vector  $V$ ]*

*For each element  $C_y$  of  $V$*

*For each element  $A_i$  of  $V_1$*

*If  $A_i = C_y$*

*Then*

*$D_y = D_y + 1$*

*End if*

*End for*

*End for*

For each element  $Cy$  of  $V$   
 For each element  $Bo$  of  $V_2$   
 If  $Bo = Cy$   
 Then  
 $Ey = Ey + 1$   
 End if  
 End for  
 End for

---

Let's see an example:

Input vectors :  $V_1$ "[enseignant , école , enseignant ]  
 $V_2$ "[enseignant , école , nom]  
 $V$  [enseignant, école , nom]  
 Output vectors :  $V_1$ "[ 2 , 1 , 0 ]  
 $V_2$ "[ 1 , 1 , 1 ]

---

#### 4.3.5 /Step 5/ : Computing the similarity between the context vectors using measure Cosine

Input: *the vector  $V_1$ " contain the word frequency of each element of vector  $V$  in  $V_1$   
 the vector  $V_2$ " contain the word frequency of each element of vector  $V$  in  $V_2$*

Output: *the value of  $\cos \alpha$  that represent the similarity between the two context vectors*

Use the following formula that we have presented in section 3.2 to calculate the value of  $\cos \alpha$

$V = \{a_1, \dots, a_j, \dots, a_L\}$   
 $V' = \{b_1, \dots, b_j, \dots, b_L\}$

$$\cos \alpha = \cos(V, V') = \frac{\sum_{j=1}^L a_j b_j}{\sqrt{\sum_{j=1}^L a_j^2} \sqrt{\sum_{j=1}^L b_j^2}}$$


---

#### 4.3.6 The whole example using algorithm Approxivect

Let's see how our algorithm **Approxivect** works with a real example of xml document:

```

<?xml version="1.0"?>
<université>
<nom></nom>
<téléphonie></téléphonie>
<enseignant>
<nom></nom>
<prénom></prénom>
</enseignant>
<enseignante>
<nom></nom>
<prénom></prénom>
</enseignante>
<formation>
<spécialité></spécialité>
</formation>
<étudiants></étudiants>
</université>

<?xml version="1.0"?>
<école>
<nom></nom>
<téléphone></téléphone>
<enseignants>
<noms></noms>
<prénoms></prénoms>
</enseignants>
<formations>
<spécialités></spécialités>
</formations>
<étudiant></étudiant>
<enseignement>
<discipline></discipline>
<heures></heures>
</enseignement>
</école>
    
```

XML document example 1

XML document example 2

We can view these Xml documents with the form of *schema* tree:

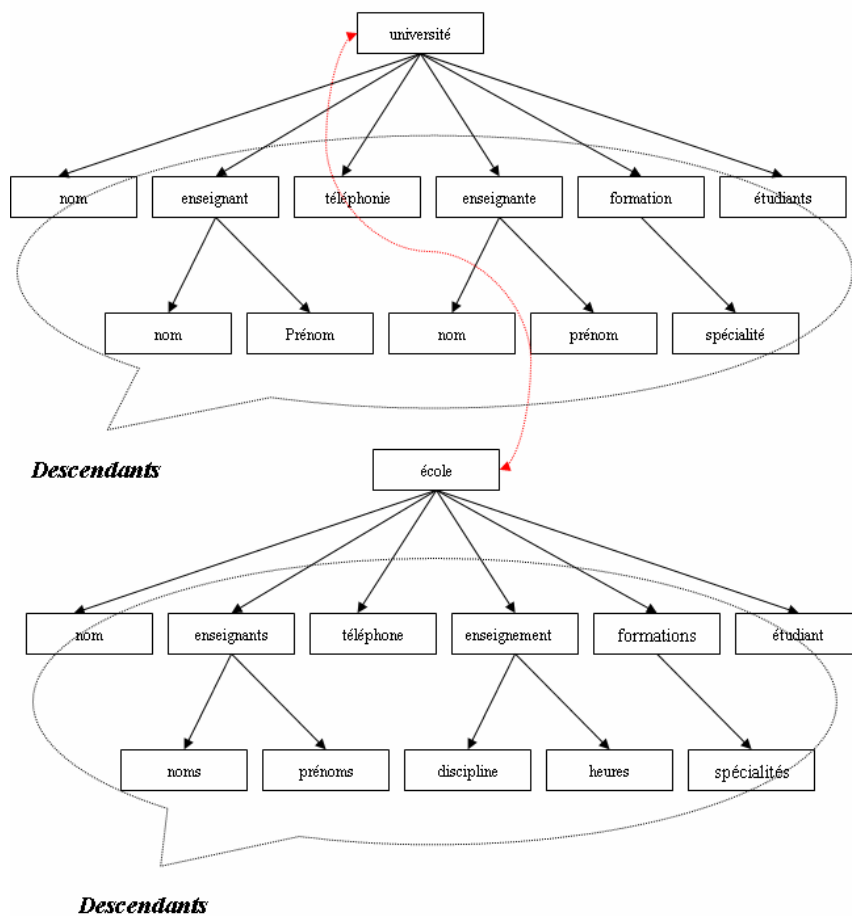


Figure 8

Let us discuss the context matching approach by analyzing how it works on the two

XML schemas of Figure 8. The approach is schema-based, and, as such, it does not exploit the information encoded in data instances.

The key idea of the context matching approach is to calculate context relations by mapping the vector constituted by the sub tree which is the descendents of element we are preparing to compare in the given schemas.

In the two schemas of XML files above, we will calculate the measure of similarity between the root node 'université' and the root node 'école'. Via the tree travel suffix, we can acquire the vectors contextual who has all the descendents of the nodes which we intend to compare.

V<sub>1</sub> [nom, enseignant, nom, prénom, téléphonie, enseignante, nom, prénom, formation, spécialité, étudiants]

V<sub>2</sub> [nom, enseignants, noms, prénoms, téléphone, enseignement, discipline, heures, formations, spécialités, étudiant ]

We suppose that the threshold R in the Algorithm **Approxivect 1** and **Approxivect 2** are both 0,6 for this example.

With **Algorithm Approxivect 1**, after the compares of similarity measure who is string - based between all the possible pairs of elements in vector V<sub>1</sub>, we can find a pair of similar elements with a similarity measure greater than the threshold which have been set to 0.6 that we have decided in advance , they are 'enseignant' and 'enseignante'. 'enseignant' has 10 letters and 'enseignante' has 11 letters, the length of 'enseignant' is shorter than the one of 'enseignante' , so 'enseignant' is more general than 'enseignante', and then we replace 'enseignante' with 'enseignant'.

In the same way, we can find the pairs of similar elements in vector V<sub>2</sub> and replace 'enseignement' with 'enseignant', replace 'noms' with 'nom' .

The two vectors are changed into:

( Words changed are represented in bold-face and italic type )

V<sub>1</sub>'[ nom, enseignant ,nom ,prénom ,téléphonie, ***enseignant***, nom , prénom ,formation , spécialité ,étudiants]

V2'[ nom, enseignant, nom ,prénoms , téléphone , **enseignant**, discipline ,heures , formations ,spécialités, étudiant ]

With **Algorithm approxivect 2** , the compares of similarity measure who is string - based don't happens between those pairs of elements who belongs to the same vector, but between the elements which comes from the different vectors.

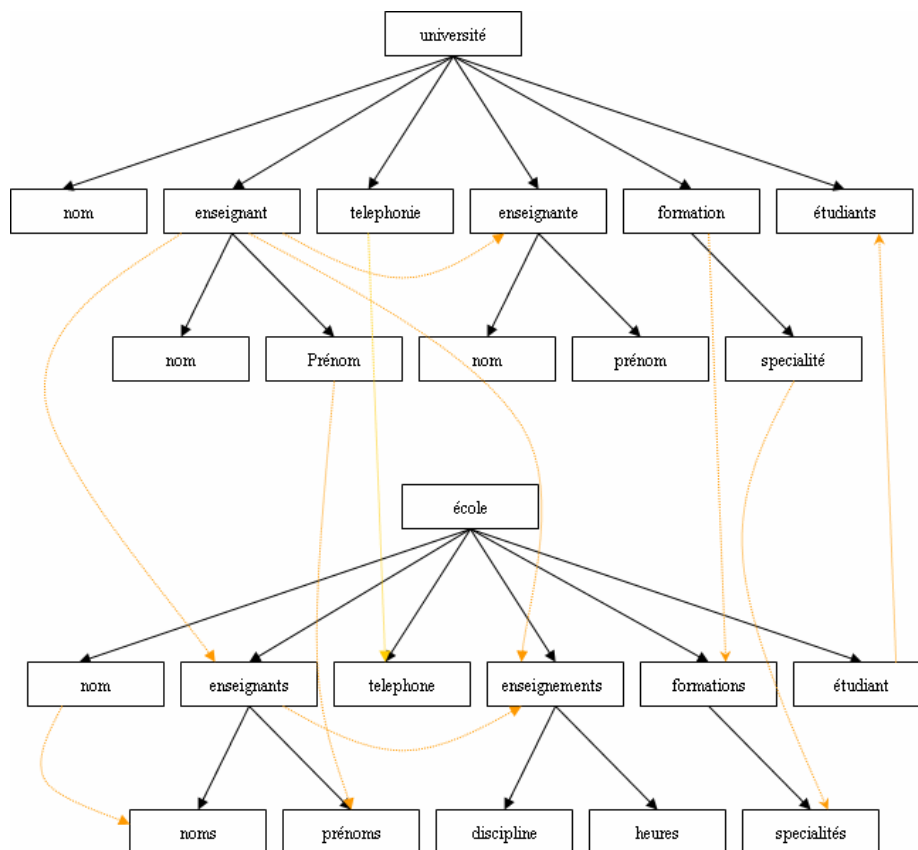
The two vectors are changed into:

(Words changed are represented in bold-face and italic type)

V1''[ nom, enseignant ,nom ,prénom , **téléphone**, enseignant, nom , prénom ,formation , spécialité ,**étudiant**]

V2''[ nom, enseignant, nom , **prénom** , téléphone , enseignant, discipline ,heures , **formation** ,**spécialité**, étudiant ]

Our replacing the nodes with the high similarity is shown graphically in the figure 5.



**Figure 9** The yellow dashed represent the replacement between the elements in the schemas when the method of comparison is based on string and the threshold R equal to 0.5 or 0.6.

If we change the value of threshold R in the **Algorithm Approxivect**, the replacement of elements in the vector will change too. The blow table 6 has recorded the

replacement happened when the threshold R is respectively 0,5 ; 0,6 ; 0,7; 0,8; 0,9.  
 We can find that the best result of replacement happened with the threshold R 0,5 or 0,6.

Vector(Algo) Method based	Vector 1 (Approxivect 1)	Vector 2 (Approxivect 1)	Vector 1↔Vector 2 (Approxivect 2)
Strings based replacement ( R = 0.9 )			enseignante→enseignants
Strings based replacement ( R = 0.7 or 0,8 )	enseignante→enseignant		téléphonie→téléphone;enseignants→enseignant formations→formation ;étudiants→étudiant spécialités→spécialité ;prénoms→prénom
Strings based replacement ( R = 0.5 or 0,6)	enseignante→enseignant	enseignements→enseignants noms→nom	téléphonie→téléphone; enseignants→enseignant; enseignants→enseignant;enseignements→enseignant formations→formation ;étudiants→étudiant spécialités→spécialité ;prénoms→prénom

**Table 6**

With **Step 3**, we can obtain a new vector which contains all words used in the vectors V1'' and V2''.

The vocabulary needs to be sorted from V1''

- nom, enseignant, prénom, téléphone, formation, spécialité, étudiant

The vocabulary needs to be sorted from V2''

- discipline, heures

So the new vector is : V [nom, enseignant, prénom, téléphone, formation, spécialité, étudiant, discipline, heures]

With **Step 4**, we can obtain vector V1''' which contain the word frequency of each element of vector V occurring in V1'' and vector V2''' which contain the word frequency of each element of vector V occurring in V2''.

Vector V	Nom	Enseignant	Prénom	Téléphone	Formation	Specialite	Etudiant	Discipline	heures
Vector V1'	3	2	2	1	1	1	1	0	0
Vector V2'	2	2	1	1	1	1	1	1	1

**Table 7**

Vector V1'''[ 3, 2, 2, 1, 1, 1, 1, 0, 0]

Vector V2'''[ 2, 2, 1, 1, 1, 1, 1, 1, 1]

There are many different ways to measure how similar two vectors are; the cosine measure is a very common similarity measure.

For two vectors  $V_1'''$  and  $V_2'''$ , the cosine similarity between  $V_1'''$  and  $V_2'''$  is given by:

$$V_1''' = \{ a_1, \dots, a_j, \dots, a_L \}$$

$$V_2''' = \{ b_1, \dots, b_j, \dots, b_L \}$$

$$\cos \alpha = \cos(V_1''', V_2''') = \frac{\sum_{j=1}^L a_j b_j}{\sqrt{\sum_{j=1}^L a_j^2} \sqrt{\sum_{j=1}^L b_j^2}}$$

The value of context similarity with our algorithm **Approxivect** is :

$$\begin{aligned} \cos \alpha &= \cos(V_1''', V_2''') = (3*2+2*2+2*1+1*1+1*1+1*1+1*1+0*1+0*1) \\ &\quad /(\sqrt{(3^2+2^2+2^2+1^2+1^2+1^2+1^2+0^2+0^2)}\sqrt{(2^2+2^2+1^2+1^2+1^2+1^2+1^2+1^2)}) \\ &= 0.90 \end{aligned}$$

Let's see the measure of similarity between two vectors without adopting the technique of similar vectors (**Algorithm Approxivect**), We directly carry out the third step. The input vectors for step 3 are:

$V_1$  [nom, enseignant, nom, prénom, téléphonie, enseignante, nom, prénom, formation, spécialité, étudiants]

$V_2$  [nom, enseignants, noms, prénoms, téléphone, enseignement, discipline, heures, formations, spécialités, étudiant ]

The vocabulary needs to be sorted from  $V_1$

- nom, enseignant, prénom, téléphonie, enseignante, formation, spécialité, étudiants

The vocabulary needs to be sorted from  $V_2$

- enseignants, noms, prénoms, téléphone, enseignement, discipline, heures, formations, spécialités, étudiant

We construct a new vector  $V$  which contains all words used in the vectors  $V_1$  and  $V_2$

Vector  $V$  [nom, enseignant, prénom, téléphonie, enseignante, formation, spécialité, étudiants, enseignants, téléphone, enseignement, discipline, heures, formations, spécialités, étudiant]

We have finished the work in the step 3, let us see what we can obtain in the step 4.

The vector containing the word frequency:

$$V_1' [3,1,2,1,1,1,1,1,0,0,0,0,0,0,0]$$

$$V_2' [1,0,0,0,0,0,0,0,1,1,1,1,1,1,1]$$

The value of context similarity without our algorithm is :

$$\begin{aligned} \text{Cos } \alpha &= \text{Cos } (V_1', V_2') \\ &= (3*1+1*0+2*0+1*0+1*0+1*0+1*0+0*1+0*1+0*1+0*1+0*1+0*1) \\ &\quad /(\sqrt{(3^2+1^2+2^2+1^2+1^2+1^2+1^2+1^2+0^2+0^2+0^2+0^2+0^2+0^2+0^2)}\sqrt{(1^2+0^2+0^2+0^2+0^2+0^2+0^2+0^2+1^2+1^2+1^2+1^2+1^2+1^2)}) \\ &=0.21 \end{aligned}$$

The measure of similarity **without our algorithm Approxivect** between two vectors is 0.21 which is very disappointing.

By contraries, with the similar vectors of our algorithm, the measure of similarity is 0.90. The reason of using our algorithm **Approxivect** is that it can get a more accurate value than the conventional ones.

The result de measure of similarity between the nodes ‘université’ and ‘école’ is shown in the table below (Table 8).

Measure Algorithm	String-based measure: <small>1/2(N-Grams(3))+1/2( String Matching)</small>		Context-based measure	Combined Measure			
	N-Grams(3) ①	String Matching ②	Measure Cosine ③	<small>1/2 String-based +1/2 context-based</small>	<small>1/3 String-based +2/3 context-based</small>	<small>2/3 String-based +1/3 context-based</small>	Maximal Value in ①,②,③
Without our algorithm	0.111	0	0.21	0.133	0.158	0.107	0.21
<b>With our algorithm and Replacement is string based</b>							
Approxivect <small>(St.R=0,5 or 0,6)</small>	0.111	0	0.902	0.479	0.506	0.338	0.902
approxivect <small>(St.R=0,7 or 0,8)</small>	0.111	0	0.724	0.39	0.447	0.279	0.724
approxivect <small>(St.R=0,9)</small>	0.111	0	0.277	0.166	0.372	0.129	0.277

Table 8 (St.R = R is for string based method)

When the measure context based exceed a threshold K, we take the maximal measure among measure N-Grams, measure string matching and measure context based as the final combined measure. (in general, the context based measure is the best one when they have the similar descendent context) Otherwise, we take one of isobarycentre combined measures. We use this

**principle to acquire the final combined measure in all the experimentations.**

```

If measure context based > K
(the value of K is temporarily equal to 0.2. I will make more experiments with the real data to determine the best
threshold K.)
then
The final combined measure=Max ( measure N-Grams, measure string matching, measure context based)
Else
The final combined measure=1/3 string based measure + 2/3 context based measure

```

From this example, we may discover very clearly the difference between the result with our algorithm and the result without using our algorithm: Our algorithm can enhance very greatly the accuracy of Context-based measure. And according to the analytical data above, we can get this conclusion:

Our algorithm is effective to improve the result with method string – based of computing the similarity between the words which have the approximate senses but distant lexical form, like ‘université’ and ‘école’.

In the above example, the approach of capturing similarity between words for the replacement between the context vectors in our algorithm **Approxivect** was concerned with the syntactic similarity of two strings. But it is not enough and sometimes it can not give us a perfect replacement’s results. For example there are two vectors  $V_1$  [ bike , car ] ,  $V_2$  [ bicycle , auto ], from the string – based similarity measure nothing could happen between the elements of two vectors because their element are not similar at all by the lexical form. So we would mistakenly think that  $V_1$  and  $V_2$  are not similar. In fact, the element ‘bike’ of vector  $V_1$  and the element ‘bicycle’ of vector  $V_2$  are the synonyms, the element ‘car’ of vector  $V_1$  and the element ‘auto’ of vector  $V_2$  have the same senses too. We should do replacement between the elements of two vectors and the vector  $V_1$  and  $V_2$  should be considered as similar. Now we are back to discuss another approach that is more concerned with the meaning of words.

**\*\*PS:** the reason that I make a presentation for wordnet based semantic similarity measurement is not for comparison between our algorithm **Approxivect** and semantic based measure (*in fact we make comparison between our algorithm **Approxivect** and string based measure*), but for using the wordnet based measure to improve our algorithm **approxivect**.

## **4.4 WordNet-based semantic similarity measurement**

### **4.4.1 WordNet**

Wordnet is a lexical database which is available online and provides a large repository of English lexical items. There is a multilingual WordNet for European languages which are structured in the same way as the English language WordNet.

WordNet was designed to establish the connections between four types of Parts of Speech (POS) - noun, verb, adjective, and adverb. The smallest unit in a WordNet is synset, which represents a specific meaning of a word. It includes the word, its explanation, and its synonyms. The specific meaning of one word under one type of POS is called a sense. Each sense of a word is in a different synset. Synsets are equivalent to senses = structures containing sets of terms with synonymous meanings. Each synset has a gloss that defines the concept it represents. For example, the words night, nighttime and dark constitute a single synset that has the following gloss: the time after sunset and before sunrise while it is dark outside. Synsets are connected to one another through the explicit semantic relations. Some of these relations (hypernym, hyponym for nouns and hypernym and troponym for verbs) constitute is-a-kind-of (holonymy) and is-a-part-of (meronymy for nouns) hierarchies.

For example, tree is a kind of plant, tree is a hyponym of plant and plant is a hypernym of tree. Analogously, trunk is a part of a tree and we have that trunk as a meronym of tree and tree is a holonym of trunk. For one word and one type of POS, if there is more than one sense, WordNet organizes them in the order of the most frequently used to the least frequently used (Semcor).

### **4.4.2 Semantic similarity between two synsets**

We capture semantic similarity between two word senses based on the path length similarity. In WordNet, each part of speech words (nouns/verbs...) are organized into taxonomies where each node is a set of synonyms (synset) represented in one sense. If a word has more than one sense, it will appear in multiple synsets at various locations in the taxonomy. WordNet defines relations between synsets and relations between word senses. A relation between synsets is a semantic relation, and a relation between word senses is a lexical relation. The difference is that lexical relations are relations

between members of two different synsets, but semantic relations are relations between two whole synsets. For instance:

- Semantic relations are hypernym, hyponym, holonym , etc.
- Lexical relations are antonym relation and the derived form relation.

Using the example, the antonym of the tenth sense of the noun light (light: 10) in WordNet is the first sense of the noun dark (dark: 1). The synset to which it belongs is {light: 10, lighting: 1}. Clearly it makes sense that light: 10 is an antonym of dark: 1, but lighting: 1 is not an antonym of dark: 1; therefore the antonym relation needs to be a lexical relation, not a semantic relation. Semantic similarity is a special case of semantic relatedness where we only consider the IS-A relationship.

#### 4.4.2.1 The path length-based similarity measurement

To measure the semantic similarity between two synsets we use hyponym/hypernym (or is-a relations). Due to the limitation of is-a hierarchies, we only work with "noun-noun", and "verb-verb" parts of speech.

A simple way to measure the semantic similarity between two synsets is to treat taxonomy as an undirected graph and measure the distance between them in WordNet. Said P. Resnik : "The shorter the path from one node to another, the more similar they are". Note that the path length is measured in nodes/verteces rather than in links/edges. The length of the path between two members of the same synsets is 1 (synonym relations). This figure shows an example of the hyponym taxonomy in WordNet used for path length similarity measurement:

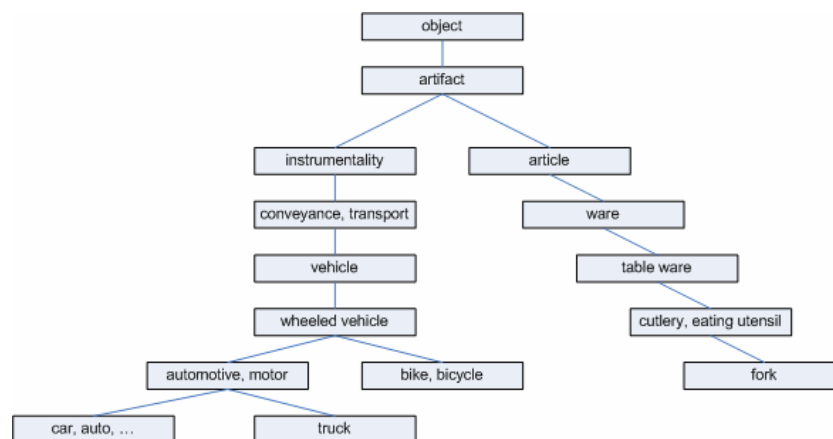


Figure 10

In the above figure, we observe that the length between car and auto is 1, car and truck is 3, car and bicycle is 4, car and fork is 12.

A shared parent of two synsets is known as a sub-sumer. The least common sub-sumer (LCS) of two synsets is the sumer that does not have any children that are also the sub-sumer of two synsets. In other words, the LCS of two synsets is the most specific sub-sumer of the two synsets. Back to the above example, the LCS of {car, auto..} and {truck..} is {automotive, motor vehicle}, since the {automotive, motor vehicle} is more specific than the common sub-sumer {wheeled vehicle}.

#### 4.4.2.2 Measuring similarity

There are many proposals for measuring semantic similarity between two synsets. In my work, we experimented with one simple measurement: for example there are two synset  $s$  and  $t$ , the value of similarity between them is :

$$\text{Similarity}(s, t) = 1/\text{distance}(s, t).$$

where distance is the path length from  $s$  to  $t$  using node counting.

The path length gives us a simple way to compute the relatedness distance between two word senses. There are some issues that need to be addressed:

- It is possible for two synsets from the same part of speech to have no common sub-sumer. Since we did not join all the different top nodes of each part of speech taxonomy, a path cannot always be found between the two synsets. But if a unique root node is being used, then a path will always exist between any two noun/verb synsets.
- Note that multiple inheritance is allowed in WordNet; some synsets belong to more than one taxonomy. So if there is more than one path between two synsets, the shortest such path is selected.
- Lemmatization: when looking up a word in WordNet, the word is first lemmatized.(I have presented in the section **Language-based techniques**) Therefore, the distance between "book" and "books" is 0 since they are identical.
- This measurement only compares the word senses which have the same part of speech (POS). This means that we do not compare a noun and a verb because they are located in different taxonomies. We just consider the words that are

nouns, verbs, or adjectives respectively. With the omission of the POS tagger, we will use Jeff Martin's Lexicon class. When considering a word, we first check if it is a noun and if so we will treat it as a noun and its verb or adjective will be disregarded. If it is not a noun, we will check if it is a verb...

- Compound nouns like "travel agent" and "teacher lecturer" will be treated as two single words via the tokenization which I have presented in section **Language-based techniques**. And then I used method **Average**:

$$\frac{2 \times \text{Match}(X, Y)}{|X| + |Y|}$$

for computing the final similarity measure where match(X, Y) are the matching word tokens between X and Y. This similarity is computed by dividing the sum of similarity values of all match candidates of both compound words X and Y by the total number of set tokens. An important point is that it is based on each of the individual similarity values, so that the overall similarity always reflects the influence of them.

Now, we will put the wordnet - based similarity measure on our algorithm approxivect.

For each element  $A_i$  in vector  $V1$

For each element  $A_j$  in vector  $V1$  ( $J > I$ )

If the value of measure similarity of string matching between  $A_i$  and  $A_j$  is bigger than a threshold  $R$ ,  $A_i$  and  $A_j$  they are not the same ones.

//MeasureSimilarity.String-Based ( $A_i, A_j$ ) >  $R$  &&  $A_i \neq A_j$

Then

If the length of  $A_i$  is smaller than the one of  $A_j$  //  $|A_i| < |A_j|$

Then

Replace  $A_j$  with  $A_i$  //  $A_j = A_i$

Else

Replace  $A_i$  with  $A_j$  //  $A_i = A_j$

End if

End if

End for

End for

**Change to:**

If the value of **similarity measure of wordnet** between  $A_i$  and  $A_j$  is bigger than a threshold  $R$ , moreover  $A_i$  and  $A_j$  they are not the same ones.

//MeasureSimilarity.wordnet ( $A_i, A_j$ ) >  $R$  &&  $A_i \neq A_j$

We have changed the condition of replacement for our algorithm **Approxivect** from string – based measure to semantic based measure. And we make the rest parts of our algorithm changeless. So now we have two methods to see whether there will be a replacement in our algorithm Approxivect or not.

## V Experimentation

The matching techniques - string based described in sections 2.2.2.4 may provide

incorrect match candidates. Our context based matching is used to correct such match candidates based on their descendents context and thus derive correct direct and complex matches. We will analyse 4 kinds of cases with our Algorithm Approxivect: they are in turn ‘close lexical and close context’, ‘close lexical and distant context’, ‘distant lexical and close context’, ‘distant lexical and distant context’.

When we use string based matching at the element level and consider the element in isolation, ignoring its context, we can say that there are two results: the close lexical (they have a high similarity with the measure of string based) and the distant lexical (they have a low similarity with the measure of string based).

Given two nodes in two schemas, string based measure gives a score depending on their lexical difference. Even if two nodes are very lexically close, they are not always the mapping elements. Let us first see an example of close lexical and close context.

## 5.1 Close lexical and close context

```
<?xml version="1.0"?>
<laboratory_LIRM>
<University></University>
<address1>
<Street></Street>
<zip_code></zip_code>
</address1>
<formation>
<speciality></speciality>
</formation>
<instructor>
<Name></Name>
<First_Name></First_Name>
</instructor>
<lab></lab>
</laboratory_LIRMM>
```

Xml document – LIRMM.xml

```
<?xml version="1.0"?>
<laboratory_CRI>
<College></College>
<address>
<Avenue></Avenue>
<postcode></postcode>
</address>
<formation>
<speciality></speciality>
</formation>
<teacher_lecturer>
<name></name>
<first_name></first_name>
</teacher_lecturer>
<laboratory></laboratory>
</laboratory_CRI>
```

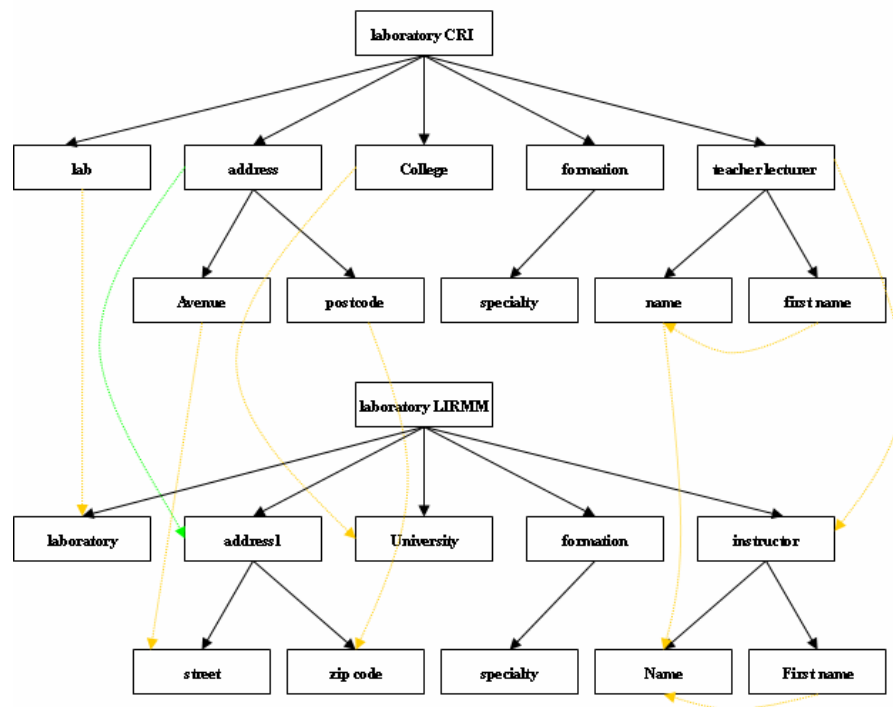
Xml document – CRI.xml

The method string – based can not always give a satisfying replacement in our algorithm Approxivect, because sometimes the words have very different lexical forms but very close senses and the method string – based dose not work in that case. For instance, in this example, ‘street’ and ‘avenue’, ‘teacher lecturer’ and ‘instructor’, ‘lab’ and ‘laboratory’, each pair of words are very close from the semantic senses but different from the lexical forms, with the string – based measure they are not all similar. So we have used the measure wordnet that is semantic based to ameliorate the result of replacement.

Since the semantic based measure is better than the string based one, why we do not

only use the measure semantic based for the replacement in our algorithm? It is because that **string based matchers are useful for some unknown to Wordnet**. That is, it can find some mapping that the semantic based measure can not find. For instance, still in this example, ‘address’ and ‘address1’ are very close in their lexical form, but with the wordnet’s measure the value of similarity is 0. That is not the result we wanted. So in my algorithm I have used both the string – based technique and semantic- based technique for the replacement between the elements of context vectors. But for this moment, I do not mix two techniques together for their facile comparison of results.

The replacing of the element with the high similarity in the schemas is shown graphically in the figure 5.



**Figure 11**, The tree of schema (The yellow dashed represent the replacement between the elements in the schemas when the method of comparison is based on semantic and the threshold R equal to 0,5. the green dashed represent the replacement that string based method can find , but semantic based method cannot find it)

Table 9 showed the string based replacement and table10 showed the semantic based replacement. Form the tables we can see that the most replacements found by the string based technique is 4 (the threshold set to 0,5 for the replacements), and the most replacement found by the semantic based technique is 8 (the threshold set to 0.5 for the replacements). To this example, the thresholds for the replacement of string based technique and semantic based technique are both 0.5.

Method based \ Vector(Algo)	Vector 1 (Approxivect 1)	Vector 2 (Approxivect 1)	Vector 1 ↔ Vector 2 (Approxivect 2)
<b>Replacement is Strings based</b>			
<i>Strings based replacement</i> ( R = 0.9 )	No replacement happened.		
<i>Strings based replacement</i> ( R = 0.8 )			address1 → address First Name → first name
<i>Strings based replacement</i> ( R = 0.6 or 0.7 )			address1 → address; Name → name First Name → first name
<i>Strings based replacement</i> ( R = 0.5 )			address → address1; postcode → zip code Name → name; First Name → first name

**Table9.** The Replacement is *Strings* based

Method based \ Vector(Algo)	Vector 1 (Approxivect 1)	Vector 2 (Approxivect 1)	Vector 1 ↔ Vector 2 (Approxivect 2)
<b>Replacement is Semantic based</b>			
<i>Semantic based replacement</i> ( R = 0.5 )	first Name → name	First Name → Name	Street → Avenue; laboratory → lab postcode → zip code; teacher lecturer → instructor Name → name ; Name → name
<i>Semantic based replacement</i> ( R = 1 )	first name → name	First Name → Name	Name → name Name → name; laboratory → lab

**Table 10.** Replacement is Semantic based

In this example, the context based measure have exceeded the threshold 0.2, so we use the maximal measure in measure N-Grams, measure string matching and measure context based as the final combined measure.

From the results of string based replacements in the table 11 and semantic based replacements in the table 12, we can prove again that our algorithm **Approxivect** can consumedly increase the accuracy of context-based measure: without our algorithm the context based similarity measure is 0.2. In the case of using our algorithm the maximal value of context based similarity measure is 0,6 for the replacement being string based, is 0,833 for the replacement being semantic based.

Measure \ Algorithm	String-based measure: <i>1/2(N-Grams(3))+1/2( String Matching)</i>		Context-based Measure	Combined Measure			Maximal Value in ①,②,③
	N-Grams(3) ①	String Matching ②	Measure Cosine ③	<i>1/2 String-based</i> <i>+1/2 context-based</i>	<i>1/3 String-based</i> <i>+2/3 context-based</i>	<i>2/3 String-based</i> <i>+1/3 context-based</i>	
Without our algorithm	0,647	0,714	0,2	0,44	0,36	0,521	0,714
	0,68						
<b>With our algorithm and Replacement is semantic based</b>							
approxivect (Se.R=0,5)	0,647	0,714	0,833	0,757	0,783	0,732	0,833
	0,68						
approxivect (Se.R=1)	0,647	0,714	0,583	0,631	0,615	0,648	0,714
	0,68						

**Table 12** (Se.R = R is the threshold for semantic based method)

Measure Algorithm	String-based measure: <i>1/2(N-Grams(3))+1/2( String Matching)</i>		Context-based Measure	Combined Measure			
	N-Grams(3) ①	String Matching ②	Measure Cosine ③	<i>1/2 String-based +1/2 context-based</i>	<i>1/3 String-based +2/3 context-based</i>	<i>2/3 String-based +1/3 context-based</i>	Maximal Value in ①,②,③
Without our algorithm	0,647	0,714	0,2	0,44	0,36	0,521	0,714
With our algorithm and Replacement is <b>string</b> based							
Approxivect (St.R=0,5)	0,647	0,714	0,6	0,64	0,627	0,654	0,714
	0,68						
approxivect (St.R=0,6 or 0,7)	0,647	0,714	0,5	0,59	0,56	0,621	0,714
	0,68						
approxivect (St.R=0,8)	0,647	0,714	0,4	0,54	0,494	0,587	0,714
	0,68						
approxivect (St.R=0,9)	0,647	0,714	0,2	0,44	0,36	0,521	0,714
	0,68						

Table11 (St.R = R is the threshold for string based method)

in this example, string based techniques are executed first. Being based on the labels of the nodes, they provide a relevant similarity measure. Two nodes ‘laboratory\_LIRMM’ and ‘laboratory\_CRI’ we are comparing in the schemas are very close from their lexical forms, so they have a high similarity score with the string based measure. Although they are very similar with the string based similarity measure, we are not sure whether they denote the same concept. On the other hand, they have also the close context with our algorithm **approxivect**. The similar context can increase their possibility of denoting the same concept. The context based technique can remove the worry that the string based technique leave us.

The context based measure have exceeded the threshold 0.2, so we take the maximal measure among the N-Grams measure, string matching measure, and context based measure as the final combined measure. We can say that two nodes in the schemas with close lexical and close context are mapping ones.

### 5.2 Close lexical and distant context

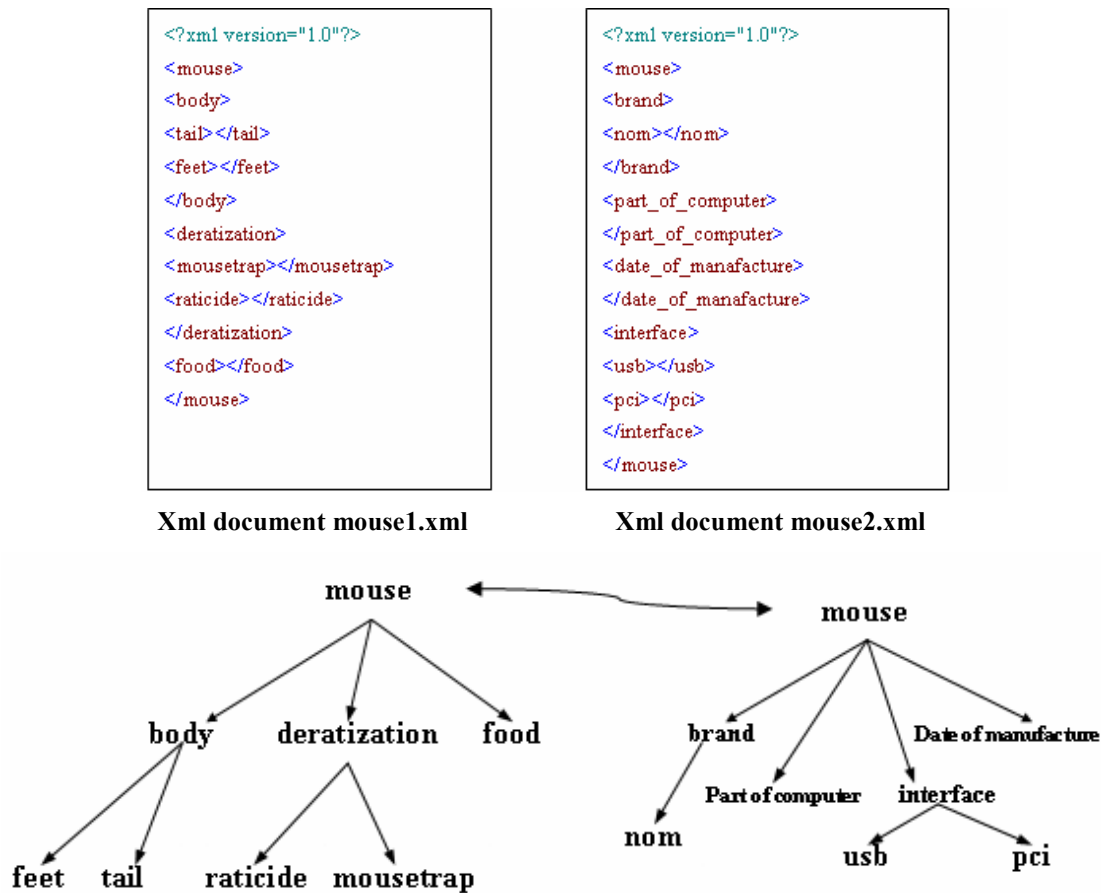


Figure 12

	Vector(Algo)	Vector 1 (Approxivect 1)	Vector 2 (Approxivect 1)	Vector 1↔Vector 2 (Approxivect 2)
Method based				
With our algorithm and Replacement is string based or semantic based				
No replacement happened.				
<i>Strings based replacement And linguistic based replacement (R = 0,5 Or 0,6 or 0,7 or 0,8 or 0.9)</i>				

Table 13

Measure	String-based measure: <small>1/2(N-Grams(3))+1/2( String Matching)</small>		Context-based measure	Combined Measure			Maximal Value in ①,②,③
	N-Grams(3) ①	String Matching ②		Measure Cosine ③	<small>1/2 String-based +1/2 context-based</small>	<small>1/3 String-based +2/3 context-based</small>	
Without our algorithm	1	1	0	0,5	0,333	0,667	1
With our algorithm and Replacement is string based or semantic based							
Approxivect Algorithm	1	1	0	0,5	0,333	0,667	1

Table 14

In this example, with string base techniques, the equivalence relationship is first discovered. They mapped two nodes (‘mouse’ and ‘mouse’) we are comparing with a

similarity measure corresponding to the strongest similarity that is '1'. But they have a very different context. With the context based technique, their similarity measure is the weakest one that is '0'. Because the context based measure do not attain the threshold K (for example, K=0.2), we take the combined measure of (1/3 string based +2/3 context based) as the final value of measure similarity. With this final measure of 0.333, we can say that the two nodes are not the mapping ones.

Two words have very high string based similarity measure, which does not mean that they must be the mapping ones. We should see whether they are context similar or not.

The most typical example in this case is the polysemous words. A polysemous word can have many sens. People frequently employ the same form to convey more than one meanings in the speech. (e.g. the word "line" in "a line of code" or "a line in the bank") and *homonyms* (e.g. "bat" the mammal or "bat" the athletic implement). The ambiguity of polysemous words can be resolved by our context based measure. The same word can be used by different groups to convey different meanings, or even within one group to convey different meanings in different situations, in whichever case they must have the different context. We see the nodes we comparing in our example: 'mouse', it represents the animal in the first schema but a part of computer in the second schema and their descendant context is absolutely dissimilar.

### 5.3 Distant lexical and close context

```
<?xml version="1.0"?>
<Arts>
<Literature>
<Chat-and-formum></Chat-and-formum>
</Literature>
<Music>
<History>
<Baroque></Baroque>
</History>
</Music>
<Art-history>
<Organizations></Organizations>
</Art-history>
<Visual-arts>
<Photography></Photography>
<Galleries>
<North-Amenca>
<United-States>
<Arizona></Arizona>
</United-States>
</North-Amenca>
</Galleries>
</Visual-arts>
</Arts>
```

Xml document google.xml

```
<?xml version="1.0"?>
<Arts_Humanities>
<Photography></Photography>
<Humanities>
<Chat-and-Forum></Chat-and-Forum>
</Humanities>
<Design-art>
<Architecture>
<History>
<Baroque></Baroque>
</History>
</Architecture>
</Design-art>
<Art-history>
<Organizations></Organizations>
</Art-history>
<Visual-Arts></Visual-Arts>
</Arts_Humanities>
```

Xml document yahoo.xml

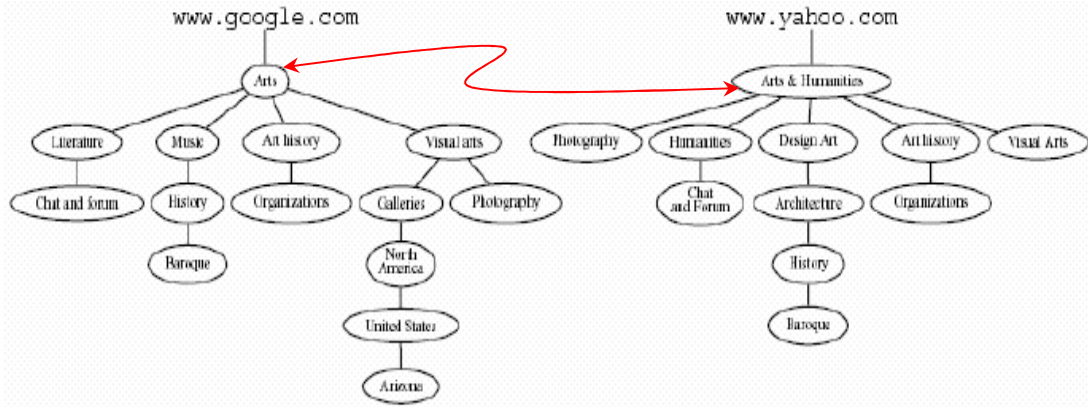


Figure 13

We can redo all steps in the first example and obtain the result de measure of similarity between the nodes ‘arts’ and ‘arts humanities’.

The operations of replacement of nodes:

Vector(Algo) Method based	Vector 1 (Approxivect 1)	Vector 2 (Approxivect 1)	Vector 1↔Vector 2 (Approxivect 2)
<b>Replacement is Strings based</b>			
Strings based replacement ( R = 0.9 )			Visual arts → Visual Arts
Strings based replacement ( R = 0.6 or 0.7 or 0.8 )			Chat and forum → Chat and Forum Visual arts → Visual Arts
Strings based replacement ( R = 0.5 )	Art history → History	Art history → History	Chat and forum → Chat and Forum Visual arts → Visual Arts

Table 15

Vector(Algo) Method based	Vector 1 (Approxivect 1)	Vector 2 (Approxivect 1)	Vector 1↔Vector 2 (Approxivect 2)
<b>Replacement is Semantic based</b>			
Semantic based replacement ( R = 0.5 )	Art history → History	Humanities → History Art history → History	Chat and forum → Chat and Forum Visual arts → Visual Arts
Semantic based replacement ( R = 1 )			Visual arts → Visual Arts

Table 16

The result de measure of similarity between the nodes ‘arts’ and ‘arts humanities’ is shown in the table 17 and table 18.

The two nodes have similar descendent context, so their measure context based have exceeded threshold 0.2. And we use the maximal measure among the measure N-Grams, the measure string matching and the measure context based as the final combined measure.

Measure Algorithm	String-based measure: $\frac{1}{2}(N\text{-Grams}(3))+\frac{1}{2}(\text{String Matching})$		Context-based Measure	Combined Measure			
	N-Grams(3) ①	String Matching ②	Measure Cosine ③	$\frac{1}{2}$ String-based $-\frac{1}{2}$ context-based	$\frac{1}{3}$ String-based $-\frac{2}{3}$ context-based	$\frac{2}{3}$ String-based $+\frac{1}{3}$ context-based	Maximal Value in ①,②,③
Without our algorithm	0.46	0	0.43	0.33	0.36	0.30	0.46
	0.23						
<b>With our algorithm and Replacement is string based</b>							
approxivect ( <i>St.R=0,5</i> )	0.46	0	0.67	0.45	0.52	0.38	0.67
	0.23						
approxivect ( <i>St.R=0,6</i> )	0.46	0	0.61	0.42	0.48	0.36	0.61
	0.23						
approxivect ( <i>St.R=0,7</i> )	0.46	0	0.61	0.42	0.48	0.36	0.61
	0.23						
approxivect ( <i>St.R=0,8</i> )	0.46	0	0.61	0.42	0.48	0.36	0.61
	0.23						
approxivect ( <i>St.R=0,9</i> )	0.46	0	0.53	0.38	0.43	0.33	0.53
	0.23						

Table 17

Measure Algorithm	String-based measure: $\frac{1}{2}(N\text{-Grams}(3))+\frac{1}{2}(\text{String Matching})$		Context-based Measure	Combined Measure			
	N-Grams(3) ①	String Matching ②	Measure Cosine ③	$\frac{1}{2}$ String-based $-\frac{1}{2}$ context-based	$\frac{1}{3}$ String-based $-\frac{2}{3}$ context-based	$\frac{2}{3}$ String-based $+\frac{1}{3}$ context-based	Maximal Value in ①,②,③
Without our algorithm	0.46	0	0.43	0.33	0.36	0.30	0.46
	0.23						
<b>With our algorithm and Replacement is semantic based</b>							
approxivect ( <i>Se.R=0,5</i> )	0.46	0	0.71	0.47	0.55	0.39	0.71
	0.23						
approxivect ( <i>Se.R=1</i> )	0.46	0	0.53	0.38	0.43	0.33	0.53
	0.23						

Table 18

In the last example, we have said that two words have very high string based similarity measure, which does not mean that they must be the mapping ones. Conversely, two words with low string based similarity may be the mapping ones. We should see whether they are context similar or not. The most typical example for this is synonyms.

What are synonyms: Different forms are often used to convey the same meaning (*synonyms*, e.g. “cheap” and “inexpensive”). And the same meaning can be expressed by different words in the dialects of different cultural or professional groups [19].

We have resolved the problem of ambiguity caused by polysymous in the last example. (the same word can be used by different groups to convey different meanings, or even within one group to convey different meanings in different situations.) Let us see

whether our algorithm is effective in the case of synonyms.

For example, people sometimes say “making a house” to mean ‘building a house’. The word ‘making’ and ‘building’ have the same meaning for representing the meaning of construction. We can say that they are synonyms in this case. They are very different from their lexical form but similar from their context. Context based similarity measure are useful for comparing the schemas in such cases. ‘Build’ may be hard to relate to ‘make’ in a general context, but it should be related in the context of residential construction like in our example. In such a situation, the user could obtain a better result that ‘making’ and ‘building’ are added to the mapping candidate. We can make use of this heuristics to analyzer the synonyms or the words that have the similar senses.

Let us see our example: ‘arts’ and ‘arts humanities’ are different from lexical forms but they have the similar context. As we have explained, we could construct a mapping between them.

### 5.4 Distant lexical and distant context

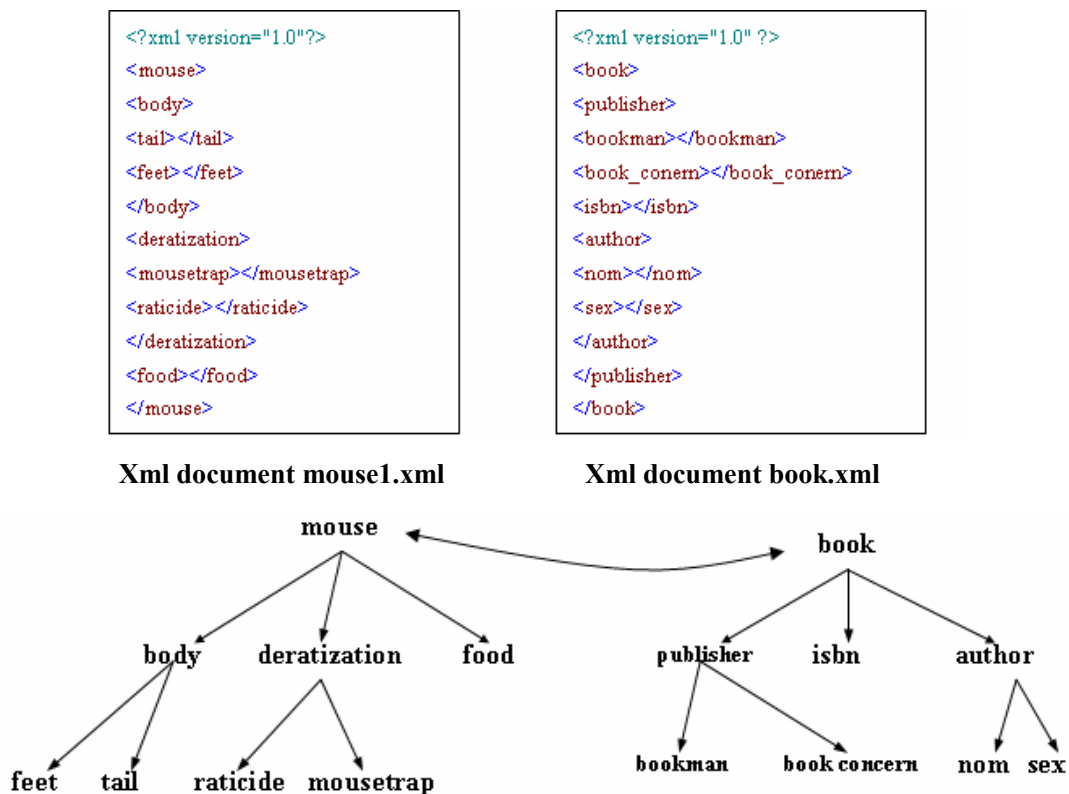


figure 14

Vector(Algo)	Vector 1 (approxivect1)	Vector 2 (approxivect 1)	Vector 1↔Vector 2 (approxivect 2)
Method based	No replacement happened.		
<i>Strings based replacement And linguistic based replacement (R =0,5 Or 0,6 or 0,7 or 0,8or 0.9 )</i>			

Table 19

Measure	String-based measure: <i>1/2(N-Grams(3))+1/2( String Matching)</i>		Context-based measure	Combined Measure			
	N-Grams(3) ①	String Matching ②	Measure Cosine ③	<i>1/2 String-based +1/2 context-based</i>	<i>1/3 String-based +2/3 context-based</i>	<i>2/3 String-based +1/3 context-based</i>	Maximal Value in ①,②,③
Without our algorithm	0	0	0	0	0	0	0
With our algorithm and Replacement is <b>string</b> based or <b>semantic</b> based							
Approxivect Algorithm	0	0	0	0	0	0	0

Table 20

I have nothing to say in this example; it is very easy to understand. If two words are not similar from both the lexical form and context, we think that they are not relational ones.

## VI Conclusion and future work

Other techniques are needed to provide additional mappings when string comparison is not sufficient. In this paper, we focus on the descendent context based technique. This technique leads to identify the context as correspondences of the mapping between two elements in the schemas, assuming that if the string based mapping is wrong, we can use the context based technique to establish the right mapping. It is a guide for the user who does not have the confidences for the results the string based similarity when deciding the mapping.

In the experimental results, we find that the precision results of context based with our algorithm Approxivect are surprisingly satisfactory for ameliorating the results with string – based measure. The results show that the nodes with similar context (for us the context is descendents.) often have similar meaning.

In the future, I will make experiments with the real data at a large scale and try to use the context based measure with our algorithm Approxivect to improve the performance of the other method of measure, such as semantic based measure.

**Reference:**

1. Giunchiglia F., Shvaiko P.: Semantic matching. *The Knowledge Engineering Review Journal*, 18(3):265-280, 2004. Short versions: Proceedings of *Ontologies and Distributed Systems* workshop at *IJCAI* and *Semantic Integration* workshop at *ISWC*, 2003.
2. Madhavan J., Bernstein P., Rahm E.: Generic schema matching with Cupid. Proceedings of *VLDB*, (2001) 49-58.
3. Do H.H., Rahm E.: COMA – A System for Flexible Combination of Schema Matching Approaches. Proceedings of *VLDB*, (2002) 610-621.
4. F. Giunchiglia and I. Zaihrayeu. Making peer databases interact - a vision for an architecture supporting data coordination. In *Proceedings of international workshop on Cooperative Information Agents*, pages 18–35, 2002. ]
5. Rahm E., Bernstein P.: A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4): 334-350, 2001.
6. D. Shasha, J. T. L. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 39–52, 2002.
7. K. Zhang and D. Shasha. Approximate tree pattern matching. In A. Apostolico and Z. Galil, editors, *Pattern matching in strings, trees, and arrays*, pages 341–371. Oxford University, 1997.
8. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 117–128, 2002.
9. J. Euzenat and P. Valtchev. Similarity-based ontology alignment in OWL-lite. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 333–337, 2004.
10. Maedche and S. Staab. Measuring similarity between ontologies. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 251–263, 2002.
11. F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of the European Semantic Web Symposium (ESWS)*, pages 61–75, 2004.
12. G. Miller. WordNet: A lexical database for English. *Communications of the ACM*, (38(11)):39–41, 1995.
13. P. Bouquet, L. Serafini, and S. Zanobini. Semantic coordination: A new approach

- and an application. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 130–145, 2003.
14. R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man and Cybernetics*, (19(1)):17–30, 1989.
  15. P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 448–453, 1995.
  16. P. Valtchev. *Construction automatique de taxonomies pour l'aide à la repr'ésentation de connaissances par objets*. Thèse d'informatique, Université Grenoble 1, 1999.
  17. P. Valtchev and J. Euzenat. Dissimilarity measure for collections of objects and values. *Lecture Notes in Computer Science*, 1280:259–272, 1997.
  18. W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. In *Proceedings of the workshop on Data Cleaning and Object Consolidation at the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2003.
  19. B.Katzenberg and P. Piela. Work Language Analysis and the Naming Problem. *Communications of the ACM*, 36(4):86–95, June 1993.
  20. Miller, George A. and Walter G. Charles. 1991. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28.
  21. W. Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. ACM SIGMOD Conference, Seattle, WA, 1998.
  22. W. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures & Algorithms*. Prentice Hall, Englewood Cliffs, N.J. 1992.
  23. Baeza-Yates, R.; Ribeiro-Neto, B.: *Modern Information Retrieval*. ACM Press; Addison-Wesley: New York; Harlow, England; Reading, Mass., 1999.
  24. Salton, G.; McGill, M. J.: *Introduction to modern information retrieval*. McGraw-Hill: New York, 1983.
  25. Mitchell, T. M.: *Machine Learning*. McGraw-Hill: New York, 1997.
  26. Hassen Kefi, Ontologies et aide à l'utilisateur pour l'interrogation de sources multiples et hétérogènes, *Thèse de Doctorat, Université Paris-Sud, France*, 2006.
  27. the assumption of Harris: the words which have identical contexts are similar, Harris, Z. (1968). *Mathematical Structures off Language*, Wiley, New York..