Measuring Similarity Between Collection of Values*

Carina F. Dorneles, Carlos A. Heuser, Andrei E. N. Lima
UFRGS, Porto Alegre, Brazil
dorneles, heuser, aenlima@inf.ufrgs.br

Altigran da Silva, Edleno de Moura
UFAM, Manaus, Brazil
alti, edleno@dcc.ufam.br

ABSTRACT

In this paper, we propose a set of similarity metrics for manipulating collections of values occuring in XML documents. Following the data model presented in TAX algebra, we treat an XML element as a labeled ordered rooted tree. Consider that XML nodes can be either *atomic*, i.e, they may contain single values such as short character strings, date, etc, or *complex*, i.e., nested structures that contain other nodes, we propose two types of similarity metrics: *MAVs*, for atomic nodes and *MCVs*, for complex nodes. In the first case, we suggest the use of several application domain dependent metrics. In the second case, we define metrics for complex values that are *structure dependent*, and can be distinctly applied for *tuples* and *collections* of values. We also present experiments showing the effectiveness of our method.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Experimentation, Measurement

Keywords

Similarity functions, Vague queries, Imprecise queries, XML

1. INTRODUCTION

An often recurrent issue in database query processing is coping with applications where users are unable to properly specify their query arguments because their understanding of the database contents is vague. As an example, consider a hospital reception database application where a clerk needs to find a patient providing just some vague facts as query arguments, for instance, the patient first name (possibly misspelled), approximate age and possible admittance date. Query processing mechanisms should be able to use these "clues" to provide users with answers that are as close as possible to their needs. Notice that, with such imprecise query arguments, several answers are likely to be considered as being plausible results, even though some of them are not anticipated by the user. This problem has gained more importance with the availability of databases on the Web, since in such cases the range of potential users is broader with widely distinct backgrounds. In this case, the problem occurs frequently in databases in which the stored instances are obtained by extraction from multiple distinct sources on the Web. In this case, a single real-world object can be stored several times with different representations. For example, in a integrated bibliography database (such as CiteSeer, http://citeseer.ist.psu.edu/), the same conference name may be stored in several distinct ways. For example, as users can possibly use any conference name variation as a query argument, the system must be able to provide answers considering approximate matchings between these arguments and the stored names.

One of the possible approaches proposed to deal with this problem is relaxing the usual requirement for the query argument and the attribute values stored in the database to be equal. Instead, some form of *similarity metric* is used for comparison. The answer to a query is a ranking of the results according to a similarity score, instead of the exact set of values equal to the query conditions.

For the case of XML databases, this issue is even more important, since in such databases the data structure is organized in multiple levels, and can involve collections of values. These features make XML database underlying details even harder to grasp for external users. On accessing such an XML database, a query processor must be able to treat multiple levels data and collection of values properly.

Several approaches can be used to evaluate a similarity score for obtaining a ranking of results. For example, in [6] the similarity is evaluated using the vector space model [2]. In that work, attribute values are assumed to be large fragments of text, rather than character strings, numbers, dates, etc (e.g. a conference title). For this last kind of attribute values, other types of textual similarity metrics such as *edit distance* (also known as *Levenstein*), or variants of it [5], have been proposed. In this context, proposals for measuring similarity between attribute values depending on the attribute domain [18, 21] has been more and more discussed. In [7] the authors experiment on several distinct similarity metrics for attribute comparison, indicating that it is an important issue to be considered. However, while some

*This work has been developed in Gerindo Project, process Nr. 55.2087/02-05 - CTInfo/CNPq

work [18] have combined metrics for flat structures, the combination of atomic similarity metrics for nested structures, particularly to collections of values, typically found in XML documents, still needs to be properly studied. In particular, it is not clear how individual similarity scores for single values occurring in the leaves of a XML tree can be combined to properly compute similarity scores for the whole tree or for a sub-tree.

In the present paper, we propose a new approach for the use of similarity metrics as a step forward when querying XML databases. Taking into account that XML elements may be *atomic*, i.e, they may contain single values as short character strings, date, numbers, etc., or *complex*, i.e., they may contain other elements as values, we have defined two types of similarity metrics: *MAV* (*metrics for atomic values*) and *MCV* (*metrics for complex values*). The *MAVs* are used with atomic XML elements, depending on the application domain of their values. For instance, for an element `name`, child of another element `person`, a MAV for *persons' name* would be applied. The *MCVs* are used with complex elements. For example, an element `conference` having `name`, `year` and `address` as children elements, is considered a complex element. Another example, is an element `publication` that has `authors` as a complex element that has several `name` elements as its children, corresponding to the name of each author in a publication. Informally, `conference` can be considered as having a structure of a *tuple*, while `authors` as having a structure of a *collection*. We believe this distinction is an important feature to be considered when comparing objects against query conditions. This way, we have defined two classes of MCVs: *tuple* and *collection*, whose metrics are dependent of the type of structure of a complex element. We call these classes of metrics *structure dependent*. Note that when we say "tuple", we refer to elements whose children elements have distinct names, as opposed to "collections" where the name of the children are the same. For tuples and collections, we define MCVs that can be *user dependent*. In this case, a collection can be treated by users as a *list* or a *set*, depending on whether their intension is indicating the order of collections members (list) or not (set). Figure 1 presents the taxonomy.

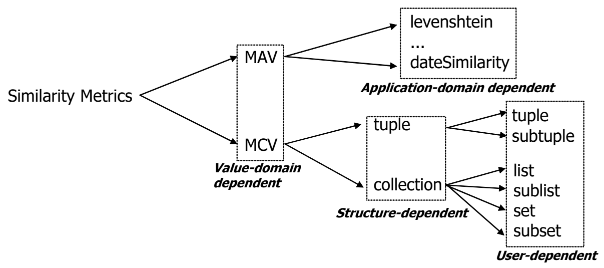


Figure 1: A taxonomy for our similarity metrics.

Throughout the paper we propose some MAV and MCVs and show how they can be implemented and incorporated into a simple query processor. This allows programmers to select proper metrics to provide users with flexible query applications allowing approximate results. Our ultimate goal is to develop a search method that supports similarity comparison between a query and the objects in an XML data source, providing a *ranking* as result. By experimentation with several types of queries and users, we demonstrate the feasibility and the effectiveness of our approach. In particular, we make contributions by proposing a method for querying XML data considering the multiple levels in the structure, take into account the distinction between tuple and collection, and the order of the children nodes of a collection. Furthermore, we proposed to query XML data (short strings) rather XML text (textual information).

Our work is quite close to the work presented in [1]. Some concepts of similarity for XML queries have been modeled and generalized in that work, where the authors define an algebra for querying XML text. The distinguishing ideas of our work are described latter in Section 2.

The paper is organized as follows. Section 2 presents the data model we rely on, which is based in [1] and the extension of some definitions adopted in our proposal. The proposed MAVs and MCVs are presented in Section 3. Section 3.4 discusses how metrics are combined to generate a final similarity score for XML documents with multiple levels. Section 4 shows experimental results achieved with our approach. Section 5 presents related work. Finally, conclusions and future work are summarized in Section 6.

2. DATA MODEL

In this section we present the data model we rely on for the development of our approach. Based on the data model used in the TIX algebra [1], we extend the notation of *scored pattern tree* and *scored data tree*.

Definition 1. *(Source and Data Tree) A* **Data Tree** $\mathcal{D}$ *is a rooted ordered tree, such that each node* ε_d *is a pair* $\varepsilon_d = (\eta, \nu)$*, where* η *is the node name and* ν *is the node value, which may assume an atomic value or another Data Tree. We define a* **Source** *as* $\mathcal{U}=\langle\mathcal{D}_1,\ ...,\ \mathcal{D}_n\rangle$ $(n > 0)$.

As stated in Definition 1, a node ε_d in $\mathcal{D}$ has a value ν and a name η. The value ν may assume an atomic content or, recursively another data tree $\mathcal{D}$. If a given node $\varepsilon_d.\nu = \mathcal{D}$, it can be considered either as *tuple* or as a *collection*.

Definition 2. *(Tuple) Let* $\varepsilon_d = \langle\eta, \nu\rangle$ *be a node and* $\nu = \mathcal{N}$*. We say that* ε_d *is a* **tuple** *iff each node* $\langle\eta_i, \nu_i\rangle$ *in the set* $\mathcal{N}$ *has a different name* η*.*

Definition 3. *(Collection) Let* $\varepsilon_d = \langle\eta, \nu\rangle$ *be a node,* $\nu = \mathcal{N}$*. We say that* ε_d *is a* **collection** *iff all node* $\langle\eta_i, \nu_i\rangle$ *in the set* $\mathcal{N}$ *has the same name* η*.*

Informally, we consider a node as belonging to a tuple if its children nodes has different names. On the other hand, if the children of a node have all the same name, we say that the node is a (homogeneous) collection of values .

Definition 4. *(Scored Pattern Tree) Let* $\mathcal{P} = (\mathcal{T}, \mathcal{F}, \mathcal{S})$ *be a triple where* $\mathcal{T}$ *is a tree, such that each node* ε_p *is a pair* $\varepsilon_p = (\eta, \nu)$*, where* η *is the node name (has a distinct integer as its label) and* ν *is the node value, which may assume an atomic value or another tree* $\mathcal{T}$*.* $\mathcal{F}$ *is a formula of boolean combination of predicates applicable to nodes and* $\mathcal{S}$ *is a set of scoring functions specifying how to calculate the score of each node in* $\mathcal{T}$*. We call* $\mathcal{P}$ *a* **Scored Pattern Tree**.

Informally, the tree $\mathcal{T}$ provides a specification of the nodes of interest, $\mathcal{F}$ presents a description about the nodes and their content, and the set $\mathcal{S}$ specify the similarity functions that must be used for resolving the vague argument problem

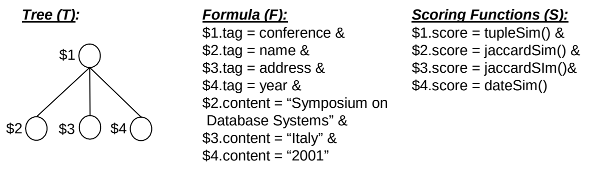


Figure 2: A scored pattern tree

and consequently generating the ranking of the results. Such a function takes two nodes and evaluates a *similarity score* between them. As we shall see latter, this metric is supposed to estimate how similar the value of a given node in a scored pattern tree is with respect to a value of a node in a data tree by implementing some *similarity metric*.

A scored pattern tree is an internal representation of a query, which is built in order to specify the interest nodes. In this paper we are not concerned about how the query is constructed, whether using a query language or not. The obvious candidate for expressing such queries is XPath.

Using the Definition 4, and considering the bibliografic citation domain, we can construct the scored pattern tree presented in Figure 2. Informally, the query in the example searches "*Conferences having name "Database" that have occurred in "Italy" in "2001""*. The set $\mathcal{S}$ of scoring functions states that the values of the `name` and `address` nodes are to be evaluated for a similarity using a metric called jaccardSim, and the metric yearSim measures a similarity score for the `year` node. In the case of the `conference` node, which is a composition of `name`, `address` and `year`, a metric called tupleSim is to be applied. Typically, such a metric would execute some operation using the values provided by the metrics specified for the children of the node for which they are specified. For now, just consider the tupleSim metric simply as an average of the values provided by the metrics jaccardSim and dateSim. The details on similarity metric and their evaluation will be provided latter.

In order to have a satisfactory evaluation for the node values, they must be in the same context and their children must be the same. We say two nodes are in the same context if their paths, starting from root, are equal. This way, we say a node ε_d in a data tree $\mathcal{D}$ is **well-evaluated** in relation to a node ε_p in a scored pattern tree $\mathcal{P}$ iff the nodes being evaluated are in the **same context** and have the **same children**. In this case, we have a *matching (m)* [1] of a scored pattern tree $\mathcal{P}$ to the collection $\mathcal{C}$, $m : \mathcal{P} \rightarrow \mathcal{C}$. The matching is described in Section 3.4

Although we use the TIX data model as a base, we have some distinguishing aspects in our proposal. The TIX algebra have been proposed for including both IR-style queries (i.e., keyword search) and database-style queries (i.e., boolean predicates). From a query, a *scored pattern tree* is built in order to specify the nodes of interest. The portion of the query that corresponds to the IR-style predicates is specified in IR-nodes in the *scored pattern tree*, the other nodes are concerned to the DB-style predicates. Differently of our proposal, a *scored pattern tree* is composed of a node-labeled and edge-labeled tree and the set of scoring functions are specified just for calculating the scores of the IR-nodes. The edges in the scored pattern tree may be labeled pc (for parent child relationship), ad (for ancestor descendant relationship), or ad* (for self-or-descendant relationship). An example of a scored pattern tree defined in TIX is shown in Figure 4.

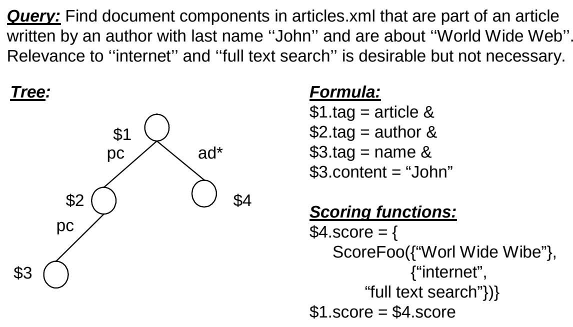

Query: Find document components in articles.xml that are part of an article written by an author with last name "John" and are about "World Wide Web". Relevance to "internet" and "full text search" is desirable but not necessary.
Tree:
$1
pc
ad*
$2
$4
pc
$3
Formula:
$1.tag = article &
$2.tag = author &
$3.tag = name &
$3.content = "John"
Scoring functions:
$4.score = {
ScoreFoo({"Worl Wide Wibe"},
{"internet",
"full text search"})}
$1.score = $4.score

Figure 4: A TIX scored pattern tree

Differently of the *scored pattern tree* defined in TIX data model, our definition of *scored pattern tree* does not have labeled edges nor IR-nodes. The former is because of the heterogeneity of the XML data. Since we are concerned about the nodes structures (tuples or collections), we do not allow relationship without involving the intervening nodes (ad and ad*), since it is needed to know what kind of structure it is specified in the query (if a collections is a set or a list, for example). So, the edges just specify a direct relationship *parent-child* between the nodes. The latter is obvious since we do not intend to work with keyword queries but with querying XML data, i.e., we are not interested in large fragments of text but in short character strings, numbers, data and so on. However, the extension of this approach seems to be straightforward since we need just to define a metric for this type of node. We have been working on extend the classical vetorial model to a similarity vetorial model in order to apply it in the nodes containing large fragments of text.

3. ESTIMATING SIMILARITY

In this section we present the metrics[1]. We suggest metrics for evaluating similarity between atomic values (*Metric for Atomic Values - MAVs*) and between complex values (*Metric for Complex Values - MCVs*). We begin by describing the metrics for atomic values and then define the metrics for complex values.

3.1 Similarity between atomic values

The *MAVs* are scoring functions applied to atomic nodes. In our approach, we propose MAVs to be specific for the domain of atomic nodes they are targeted to. For instance, there can be specific MAVs for person names, dates, prices, etc. Table 1 presents a general classification[2] of the MAVs.

As an example, consider the Figure 2 as the scored pattern tree and the Figure 3 c) as a data tree in a source U, the MAV jaccardSim defined for the node $2 evaluates a similarity score for the node `address`. The similarity is computed using *jaccardSim(ε_p^3, ε_d^3)*. In this case, the similarity metric has as input two values of the string domain, `"Italy"` and `"Italy"`.

It is ... mentioning that we do not ... the list of possible MAVs to be closed. Indeed, deciding on the domain, more MAVs can be used. For example, the set of transformation functions presented in [21].

[1]The similarity score is in the interval [0, 1].
[2]The classification is general and it is not limited to that three ones. For lack of space we do not present a description about each one.

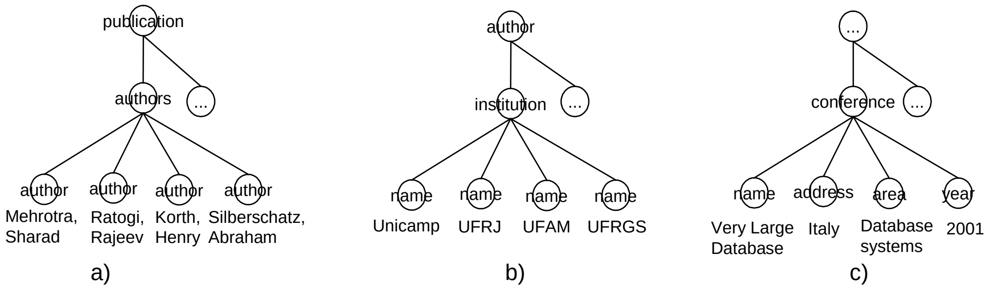


Figure 3: Examples of tuples and collection structures in XML documents

Type	Description
String	evaluates a similarity score for string values, which might be done by means *editSim*, *jaccardSim*, or *acronymSim*.
Date	calculates a similarity score for date values, which might be done by means *dateSim*, *daySim*, *monthSim* or *yearSim*.
Number	estimates a similarity score for numeric values, which might be done by means *sameNumberSim*, *diffNumberSim*.

Table 1: A general classification for MAVs

3.2 Similarity between complex values

The strategy we apply to determine the similarity between two complex nodes, consists in combining the similarity scores previously computed for the children of these nodes. These combinations are produced according to one of the MCVs we now present. If a child is atomic, a MAV is used. Otherwise, a MCV is recursively used.

We assume that complex nodes representing data items correspond to aggregations of other nodes according to one out of the possible structures mentioned early: tuple or collection. In the case of collections, the users can treat them as list or set.

For instance, in the Figure 3 the node `conference` of the data tree `c)` is considered as a tuple. A collection is illustrated in the data trees `a)` and `b)` by means of the node `authors` and `institution`, respectively. The first one might be treated as a collection of `author`. The node `institutions` of the data tree `b)` might be considered a collection of institutions. The treatment of a collection as a list or a set depends on the user intention. For instance, the data tree `b)` should represent a **list** while the data tree `a)` should represent a **set**. More specifically, the distinction between **list** and **set** depends strictly on the user query needs.

In what follows, we present each one of the metrics along with examples of their use. On defining the metrics we assume that a data tree $\mathcal{D}$ is well-evaluated in relation to a scored pattern tree $\mathcal{P}$.

3.2.1 Metrics for tuple

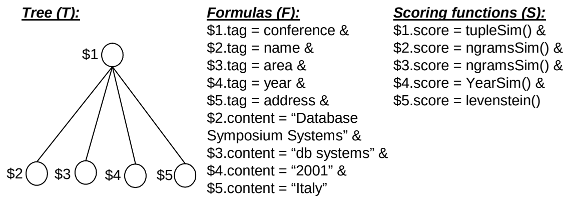
Definition 5. *Let ε_p, be a node in $\mathcal{P}$ and ε_d a node in $\mathcal{D}$, n and m the number of children of ε_p and ε_d respectively, $\varepsilon_p.score$ = "**tupleSim**". The similarity between ε_p and ε_d is defined as:*

$$tupleSim(\varepsilon_p, \varepsilon_d) = \frac{\sum_{\varepsilon_p^i.\eta=\varepsilon_d^j.\eta} (sim(\varepsilon_p^i, \varepsilon_d^j))}{max(m,n)}$$

where $(1 \leq i \leq n)$, $(1 \leq j \leq m)$.

By using the *tupleSim* metric, each child node ε_d^i of ε_d is compared with the child node ε_p^j of ε_p, with the same name $\varepsilon_p^i.\eta = \varepsilon_d^j.\eta$ and in the same context. The *max()* fuction return the greatest number of children between ε_p and ε_d.

Example 1. *Considering the data tree* `c)` *in Figure 3 suppose a query: "Retrieve the conferences having name similar to 'database system symposium', area similar to 'db systems' and that occurred in 'Italy' on '2001'".*



Formulas (F):
$1.tag = conference &
$2.tag = name &
$3.tag = area &
$4.tag = year &
$5.tag = address &
$2.content = "Database Symposium Systems" &
$3.content = "db systems" &
$4.content = "2001" &
$5.content = "Italy"

Scoring functions (S):
$1.score = tupleSim() &
$2.score = ngramsSim() &
$3.score = ngramsSim() &
$4.score = YearSim() &
$5.score = levenstein()

Figure 5: Scored pattern tree - Example 1

A scored pattern tree is presented in Figure 5. In this example, the node `conference` is evaluated for a similarity score using the *tupleSim* function that uses the values of similarity scores provided by the MAV specified for each child element, `name`, `address`, `area` and `year`. Considering $\mathcal{P}$ in Figure 5 and the data tree `c)` in Figure 3, we have: *tupleSim(ε_p, ε_d)= (jaccardSim(ε_p^1, ε_d^1)+ levenshtein(ε_p^2, ε_d^2)+ jaccardSim(ε_p^3, ε_d^3)+ yearSim(ε_p^4, ε_d^4))/max(m,n)*. So, having $max(m, n) = 4$ the final similarity score is $tupleSim = ((0,3333) + (1) + (0.55) + (1))/4 = 0,720825$ (Real values obtained by running the MAVs).

The user can provide the system with just some complex nodes children, meaning that there is no interest in the other nodes values, just in those provided in the scored pattern tree. In this case, the user must inform the system that the query node is just a *sub-tuple* of the data tree node, matching as follows.

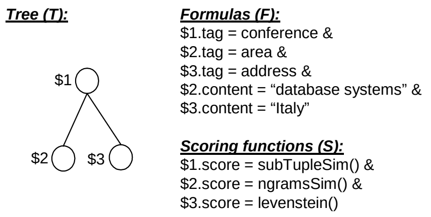
Definition 6. *Let ε_p be a node in $\mathcal{P}$ and ε_d a node in $\mathcal{D}$, n and m the number of children of ε_p and ε_d respectively, $\varepsilon_p.score$ = "**subTupleSim**". The similarity between ε_p and ε_d is defined as:*

$$subTupleSim(\varepsilon_p, \varepsilon_d) = \frac{\sum_{\varepsilon_p^i.\eta=\varepsilon_d^j.\eta} (sim(\varepsilon_p^i, \varepsilon_d^j))}{n}$$

where $(1 \leq i \leq n)$.

Informally, having $\varepsilon_p.score$ = "*subTupleSim*" the similarity score do not decrease by having $m \neq n$, because we are looking for a minimum amount of children nodes, those defined in ε_p.

Example 2. *Consider the query: "Retrieve conferences in 'database system' area, occurred in 'Italy'".*



Formulas (F):
$1.tag = conference &
$2.tag = area &
$3.tag = address &
$2.content = "database systems" &
$3.content = "Italy"

Scoring functions (S):
$1.score = subTupleSim() &
$2.score = ngramsSim() &
$3.score = levenstein()

Figure 6: scored pattern tree - Example 2

Distinctly from Example 1, the node `conference` is evaluated for a similarity score by using the *subtuple* MCV, which consideres only the values of similarity scores of the children nodes which are in the query, `area` and `address`.

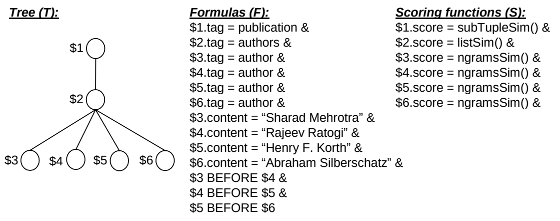
3.2.2 Metrics for collections

Definition 7. *Let ε_p be a node in $\mathcal{P}$ and ε_d a node in $\mathcal{D}$, n and m the number of children of ε_p and ε_d respectively, $\varepsilon_p.score$ = "**listSim**". The similarity between ε_p and ε_d is defined as:*

$$listSim(\varepsilon_p, \varepsilon_d) = \frac{\sum_{\varepsilon_p^i.\eta=\varepsilon_d^j.\eta \ and \ i=j} (sim(\varepsilon_p^i, \varepsilon_d^j))}{max(m,n)}$$
where $(1 \le i \le n)$, $(1 \le j \le m)$.

The *listSim* asserts that ε_d^j in $\mathcal{D}$ is compared with ε_p^i in $\mathcal{P}$ *if and only if* $i = j$, i.e. just if they are in the same order and having the same name $\varepsilon_p.\eta = \varepsilon_d.\eta$.

Example 3. *Suppose the following query, having as Data Tree that one shown in Figure 3* a)*: "Retrieve publications having the list of the following authors 'Sharad Mehrotra', 'Rajeev Ratogi', 'Henry F. Korth' and 'Abraham Silberschatz' in this order".*



Formulas (F):
$1.tag = publication &
$2.tag = authors &
$3.tag = author &
$4.tag = author &
$5.tag = author &
$6.tag = author &
$3.content = "Sharad Mehrotra" &
$4.content = "Rajeev Ratogi" &
$5.content = "Henry F. Korth" &
$6.content = "Abraham Silberschatz" &
$3 BEFORE $4 &
$4 BEFORE $5 &
$5 BEFORE $6

Scoring functions (S):
$1.score = subTupleSim() &
$2.score = listSim() &
$3.score = ngramsSim() &
$4.score = ngramsSim() &
$5.score = ngramsSim() &
$6.score = ngramsSim() &

Figure 7: scored pattern tree - Example 3

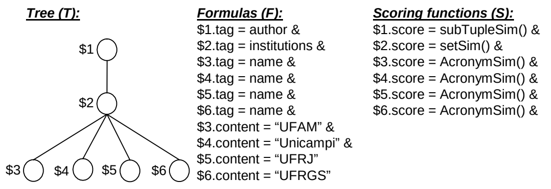
The corresponding scored pattern tree is presented in Figure 7. The node `authors` is evaluated for a similarity score by using the metric *listSim* which uses the values of each `author` element. For example, on matching the scored pattern tree and the data tree a) we have *jaccardSim*(ε_p^1,ε_d^1)+ *jaccardSim*(ε_p^2, ε_d^2)+ *jaccardSim*(ε_p^3, ε_d^3)+*jaccardSim*(ε_p^4, ε_d^4). Having each child node evaluated and $max(m,n) = 4$, the node `authors` can be evaluated by means of $listSim = ((1) + (1) + (1) + (1))/4 = 1$. For the `publication` node the *subTupleSim* is applied using the score similarity of its child node `authors`.

Definition 8. *Let ε_p be a node in $\mathcal{P}$ and ε_d a node in $\mathcal{D}$, n and m the number of children of ε_p and ε_d respectively, $\varepsilon_p.score$ = "**setSim**". The similarity between ε_p and ε_d is defined as:*

$$setSim(\varepsilon_p, \varepsilon_d) = \frac{\sum_{\varepsilon_p^i.\eta=\varepsilon_d^j.\eta} (max(sim(\varepsilon_p^i, [\varepsilon_d^1, \ldots, \varepsilon_d^m])))}{max(m,n)}$$
where $(1 \le i \le n)$, $(1 \le j \le m)$.

When $\varepsilon_p.score$ = "*setSim*", each element ε_p^i in the $\mathcal{P}$ is compared with all children nodes $[\varepsilon_d^1, \ldots, \varepsilon_d^m]$ in ε_d, and a function *max()* returns the maximum similarity score.

Example 4. *Consider the following query given the data tree* b) *in Figure 3: "Retrieve authors who work in the following institutions 'UFAM', 'Unicamp', 'UFRJ' and 'UFRGS'"*



Formulas (F):
$1.tag = author &
$2.tag = institutions &
$3.tag = name &
$4.tag = name &
$5.tag = name &
$6.tag = name &
$3.content = "UFAM" &
$4.content = "Unicampi" &
$5.content = "UFRJ"
$6.content = "UFRGS"

Scoring functions (S):
$1.score = subTupleSim() &
$2.score = setSim() &
$3.score = AcronymSim() &
$4.score = AcronymSim() &
$5.score = AcronymSim() &
$6.score = AcronymSim() &

Figure 8: scored pattern tree - Example 4

A scored pattern tree is presented in Figure 8. In this example, the node `institution` is evaluated for a similarity score using the *setSim* metric. This metric evaluate for each child node in ε_p a similarity score with all child nodes in ε_d and returns the greatest one, which is used as value for calculate the simlarity score between ε_p and ε_d. Considering $\mathcal{P}$ in Figure 8 and the data tree b) in Figure 3, we have:
setSim($\varepsilon_p, \varepsilon_d$)=(
max(AcronymSim(ε_p^1, ε_d^1), *AcronymSim*(ε_p^1, ε_d^2),
AcronymSim(ε_p^1, ε_d^3), *AcronymSim*(ε_p^1, ε_d^4)) +
max(AcronymSim(ε_p^2, ε_d^1), *AcronymSim*(ε_p^2, ε_d^2),
AcronymSim(ε_p^2, ε_d^3), *AcronymSim*(ε_p^2, ε_d^4)) +
max(AcronymSim(ε_p^3, ε_d^1), *AcronymSim*(ε_p^3, ε_d^2),
AcronymSim(ε_p^3, ε_d^3), *AcronymSim*(ε_p^3, ε_d^4)) +
max(AcronymSim(ε_p^4, ε_d^1), *AcronymSim*(ε_p^4, ε_d^2),
AcronymSim(ε_p^4, ε_d^3), *AcronymSim*(ε_p^4, ε_d^4))
)/max(m,n).

So, having $max(m,n) = 4$ the final similarity score is $tupleSim = ((1) + (1) + (1) + (1))/4 = 1$.

As in the tuple case, it is possible the user informs the system that just a *sublist* or *subset* of nodes are being provided. In this case, the similarity metric differs from the list, and set, in the sense of the divisor is the number of children nodes informed in the scored pattern tree. For space restrictions they are not defined here.

In general, the similarity score between complex nodes is given by means of the similarity score of their children nodes divided by the greatest number of children nodes between the two nodes being evaluated. This divisor is used because if the number of children in $\mathcal{P}$ and in $\mathcal{D}$ is the same, the similarity is given by the average of their children similarity score. On the other hand, if there are more children in one of the nodes being compared, the similarity have to decrease since the children amount is different. The distinction among the MCVs is in the process of how to combine the similarity score of the nodes children (if a child is atomic, a MAV is used, otherwise, a MCV is recursively used). This combination depends on the structure type of a complex node as well as the user needs expressed in the query. As this combination, sometimes, depends on the query needs, we have to allow a relaxing of the metric in the case the user provides just part of the complex node structure. For instance, it is possible for the user to specify a query providing just the first author of a citation that has three authors. In this case, the divisor is the number of children in a scored pattern tree.

3.3 Similarity Properties

A important feature of similarity function is the compliance with similarity properties. So, most work in the literature discuss similarity (or a *(di)similarity*) as a distance in a metric space [17, 8]. Such a funtion must be defined obeying some requirements. Let $\tau : \mathcal{S}\times\mathcal{S}\rightarrow$[0,1] be a (di)similarity function and $\mathcal{S}$ the data source, $\forall\ \epsilon_d^1, \epsilon_d^2, \epsilon_d^3 \in \mathcal{S}$,:

1. self-similarity: $disim(\epsilon_d^1, \epsilon_d^1) = disim(\epsilon_d^2, \epsilon_d^2)$
2. minimality: $disim(\epsilon_d^1, \epsilon_d^2) \geq disim(\epsilon_d^1, \epsilon_d^1)$
3. symmetry: $disim(\epsilon_d^1, \epsilon_d^2) = disim(\epsilon_d^2, \epsilon_d^1)$
4. triangular inequality: $disim(\epsilon_d^1, \epsilon_d^3) + disim(\epsilon_d^3, \epsilon_d^2) \geq disim(\epsilon_d^1, \epsilon_d^2)$

Any function satisfying 1, 2 and 4 is a metric function, any application satisfying 1, 2 and 3 is a (di)similarity function [17]. It is important to observe that the similarity properties described above only ensure a consistent definition of a (di)similarity function, and cannot be used to save comparisons in a proximity query. This consistence is important on defining indexing methods. The metrics we have defined in this paper satisfy properties 1, 2 and 4 easily. However in order to satisfy the symmetry property the metric *setSim* must have as first parameter the tree (a data tree or a scored pattern tree) that has the minor children nodes.

3.4 Matching and Ranking

The matching of a scored pattern tree $\mathcal{P}$ to a data tree $\mathcal{D}$ is given when there exists a matching of the scored pattern tree and the data tree. The mapping does not need to be total, since if the formula $\mathcal{F}$ is not completely satisfied, the scoring functions $\mathcal{S}$ defined evaluate a low score similarity between the two trees. From the mapping of a scored pattern tree $\mathcal{P}$ to a data tree $\mathcal{D}$ a *scored data tree* is generated.

Definition 9. *A* **Scored Data Tree** *is a rooted ordered tree, such that each node ε_d is a triple $\varepsilon_d = (\eta, \nu, \mu)$, where η is the node name, ν is the node value, which may assume an atomic value or another scored data tree and μ is the similarity score.*

A TIX scored data tree is a rooted ordered tree, such that each node carries data in the form of a set of attribute-value pairs. The score of a node is obtained by means of a scored pattern tree. A score is calculated through a scoring functions just for nodes where an IR-style predicate is imposed. The score of the tree is the score of the root node, which is reached just by means of the IR-nodes. In the case of the scored data tree defined in our approach, all nodes have a similarity score.

Definition 10. *(Ranking) Let $\mathcal{U}$ =\{ $\mathcal{D}_1$, ..., $\mathcal{D}_n$ \}* ($n > 0$) *be the XML source, $\mathcal{P} = (\mathcal{T}, \mathcal{F}, \mathcal{S})$ be a scored pattern tree and τ a similarity function defined $\mathcal{S}$ for the node* \$1 *(the root node). We call $\mathcal{R}(\mathcal{P})$=$\langle$ $\mathcal{D}_1$, $\mathcal{D}_2$, ..., $\mathcal{D}_k$ $\rangle$ a* **k-ranking** *for $\mathcal{U}$ given $\mathcal{P}$, where $\mathcal{D}_i \in \mathcal{U}$, $\tau(\mathcal{D}_i, \mathcal{P}) > 0$* ($i = 1, \ldots, k$) *and $\tau(\mathcal{D}_1, \mathcal{P}) \geq \tau(\mathcal{D}_2, \mathcal{P}) \geq \cdots \geq \tau(\mathcal{D}_k, \mathcal{P})$.*

By definition, the ranked query results are the *top-k* data trees ε_d closer to scored pattern tree ε_p.

As in TIX, the matching of a full pattern tree into a collection is the first step to generated the score for the ranking. So, after the matching process, the result is used for various operators in TIX, which also manipulate the calculated score.

3.4.1 Weighted nodes

A relevant issue to be considered is the importance of each node in a data tree, given by a weight. A very common solution used as weighting method is the application of TF/IDF method [5], or a variation [20]. However, considering we have different representations for the same information in the XML database, for instance, a conference name, *"VLDB"* and *"Very Large Database"*, it is not trivial to define an importance for each term in database (in the example, both terms *"VLDB"* and *"Very Large Database"* would be considered as the same term). So, we suggest a weighting method that corresponds to an intuitive notions of semantic weighting. Specifically, it should depend only on the semantic relation between the concepts involved, and not on their syntactic content as considered in TF/IDF.

Our proposal can be easily extended for defining a weight for the nodes. For example, considering the example c) in Figure 3, the conference name can have more importance than year. So, let $\mathcal{N}$ a set of nodes $[\varepsilon_1, ..., \varepsilon_n]$, we assign weights respectively to each nodes, such that $w_1 + ... + w_n = 1$. During the matching process, the weight value is used to evaluate the similarity score for the respective node, the rest of the query processing remain unchanged.

3.4.2 Indexing the database

Obviously, a sequencial search on the XML database is an inefficient task to be executed when matching the full scored pattern tree into a source of data trees. For this reason we have developed two methods for indexing the XML database, which support vague query arguments. The first is a simple hash table, whose indexes are the several representations of each term in the set of data trees. Another proposal a modified M-Tree is proposed, applying a twisting function over the metric space to generate a new space, the twisted space[3]. Due to lack of space we do not describe the methods in this paper.

4. EXPERIMENTS

In this section we describe some experiments that show the effectiveness of our method. To evaluate our similarity metrics we have used recall/precision curve [2], which is the standard for evaluating information retrieval methods. We have three main goals in our experiments:

- to demonstrate the effectiveness of handling data collections;
- to show the importance of stating the specific data collection type (list or set) rather than using just a single generic collection (i.e., to consider nodes order);
- to corroborate previous related work on the use of application domain dependent MAVs [20].

[3]A paper has been submitted[19] proposing the modified M-Tree.

4.1 Data Sources

We have used two real data sources domains for our experiments: i) citations from BibTEX files and; ii) movie data from different video rental companies in the internet. The citations data are originated from *The Collection of Computer Science Bibliographies* (http://liinwww.ira.uka.de/ bibliography/Database/index.html) and from members of *UFRGS Database Group* (http://metropole.inf.ufrgs.br/ DBGroup/) totalizing 16.519 citation entries. In the first source, the files used were the following: ACM/SIGMOD, Information Systems, Bibliography on database systems, Bibliography on Semistructured Data, VLDB journal, VLDB Conference and ACM Transactions on Information Systems. As the sources are originally in BibTEX format, we have converted them into XML using bib2XML (http://www.cs.duke.edu/ ~sprenkle/bibtex2html/). The XML files can be found at http://metropole.inf.ufrgs.br/~dorneles/xml_bibsources. The video rental data sources are originated from multiples data sources, such as Blockbuster US, Blockbuster UK and Blockbuster CA. The data were manually extracted, the label used in the HTML pages were transformed into XML tags.

4.2 Methodology

For the experiments, we have specified 28 queries for each domain (56 on the total). The final ranking was obtained in four distinct ways, each one exploring different similarity metrics (examples of queries can be found in [9]). The variations were constructed as follows.

- *Tp-Apd (Tuple and Application-domain dependent).* In this experiments we have handled collections and tuples with a general similarity function. So, we have used *tupleSim* as the scoring function for complex values, and application-domain dependent for the atomic values. Notice that in this case, we could consider that we have simulated the VAGUE system approach [18] (described briefly in Section 5), i.e. using domain-specific similarity metric for atomic nodes but without considering collections;
- *Std-Apd (Structure dependent and Application dependent).* In this case, we have consider tuples and collections separately, but the we have ignored order in the collections, consider however just *tupleSim* and *setSim* for complex values and application-domain dependent metrics for atomic nodes. Notice that in this case we have made no distinction between sets and list, i.e., we do not rely on user-specified ordering constraint.
- *Usd-Gnl (User dependent and General MAV).* The queries in this tests were constructed using tuple, set or list in complex element, and a general string metric for atomic nodes. The general string metric used was the Levenshtein edit distance;
- *Usd-Apd (User dependent and Application-domain dependent)*: as in *Usd-Gnl*, the queries were constructed using tuple, set or list, depending on case, and application-domain dependent metrics for atomic element.

The queries in *Usd-Apd* and *Usd-Gnl* are user dependent. In this case, we have requested a set of users to specify the *MCVs*. In both cases, the users access data through a form-like interface having just to fill in fields for specifying a query. In this strategy, the users received two instructions for posing a query: i) they were not using a keyword search system, but a system that supports vague query arguments, and ii) they had to indicate whether they are concerned with the order of the attributes or not.

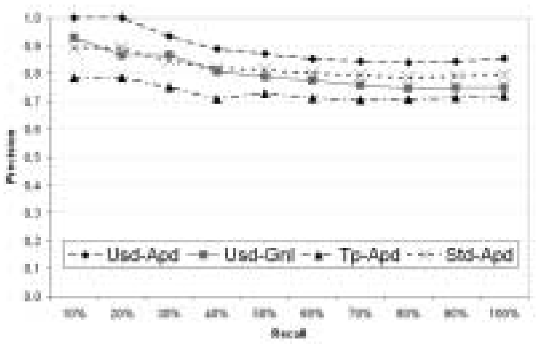


Figure 9: Recall/precision curve

4.3 Results

The goals stated above were reached by comparing each set of queries described above. The comparisons and the results are the following.

1. *Tp-Apd X Std-Apd.* In order to demonstrate the effectiveness of correctly handling collections, we have evaluated queries in which collections are not taken in account (*Tp-Apd*) against queries considering collections *Std-Apd.* Figure 9 illustrates a recall/precision curve for this evaluation. Looking at the results we can see that the distinction between a tuple and a collection is an important feature to be considered when comparing XML data.

2. *Std-Apd X Usd-Apd.* In this experiment we consider structure dependent queries *Std-Apd*, in which the user does not consider the order, against user dependent queries *Usd-Apd*, in which the user informs the order of values collections. Using metrics that consider the order, we get the top data trees to have a similarity score equal to 1. Figure 9 shows the result.

3. *Usd-Gnl X Usd-Apd.* Corroborating previous related work, we reafirm that using an application-domain dependent metric for atomic value increases the precision of the results. Figure 9 depicts the results showing that for the set of queries *Usd-Apd* the top data trees have similarity equal to 1.

The results for the movie rental domain have been quite similar to those shown in this paper. However, for lack of space we do not present them here.

5. RELATED WORK

The problem of handling with different representation of the same piece of data became the focus of important work in several research communities. They have different applications areas, from data integration [6, 16, 20] until query processing having similarity string joins [13, 14, 12].

In the database community, some approaches have been proposed to handle query with vague conditions. In the VAGUE System [18] the vector model is refined to include a proximity operator. The ranking of the results is based on the Euclidean distance and the similarity between attributes is done by using specific atomic metrics for each attribute. In [13] the proposal is to use the well known cosine similarity metric for string matching and to propose a SQL implementation of a sampling-based strategy to compute similarity text join in a RDBMS.

An important approach that considers similarity joins is WHIRL[6]. The proposal is based on the motivation of integrate information from structured (relations) information sources that contain textual information (documents). They assume that all data are stored in relations, and the attributes of each relation are fragments of text, whose data model they call STIR (*Storing Texts In Relations*). This approach differs of our in the sense we are not interested in large fragments of text but in short strings.

In the context of XML documents, some approaches [15, 4, 3] have already been proposed extending the vector model, where each node has a vector of terms. Some proposals extend XML query languages, such as XQL [11] and XML-QL [10], in the sense of querying XML documents through a search engine using the XML language query processor.

The main distinguishing aspect of our work from all above is that we propose specific metric for different structures of complex XML nodes. As opposed to the majority of related work described here we do not extend the vectorial model for combining the atomic similarity values. Corroborating in previous work [20, 18] applying specific application-domain dependent metrics for the atomic nodes increases the accuracy of the results. We believe some features of the structure of complex nodes can be used, such heterogeneity, i.e whether the node children are of the same type or not, providing a distinction between tuple and collection, and the order of the children nodes of a collection.

6. CONCLUSION AND FUTURE WORK

We have proposed a stepforward method for supporting vague query arguments, providing a precise ranking of the query results. Our main contribution is the definition of a method for querying XML data considering the multiple levels in the structure. On considering this feature, we have defined metric for complex XML elements considering the distinction between tuple and collection, and the order of the children nodes of a collection.

Our proposal could be used in the context of several existing applications. A naive-user interface might be generated by reading an XML Schema, and by filling the form the scored pattern tree might be internally generated. Web services having applications requesting data by providing the services server with parameters for some services, could support the similarity metrics proposed in this paper. A query processor that handles similarity operators might use the matching process as an access method. A data warehouse application might use the matching process for fetching the incoming objects against the reference object.

7. REFERENCES

[1] Al-Khalifa, S., Yu, C., and Jagadish, H. V. Querying structured text in an XML database. In *ACM SIGMOD* (2003), ACM Press, pp. 4–15.

[2] Baeza-Yates, R., and Ribeiro-Neto, B. *Modern Information Retrieval.* Addison Wesley, 1999.

[3] Carmel, D., Efrati, N., Landau, G. M., Maarek, Y., and Mass, Y. An extension of the vector space model for querying XML documents via XML fragments. In *Workshop On XML and Information Retrieval, SIGIR* (2002).

[4] Carvalho, J. P., and da Silva, A. S. Finding similar identities among objects from multiple web sources. In *ACM International Workshop on Web Information and Data Management* (2003, poster).

[5] Chaudhuri, S., Ganjam, K., Ganti, V., and Motwani, R. Robust and efficient fuzzy match for online data cleaning. In *ACM SIGMOD* (2003).

[6] Cohen, W. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Informations Systems 18*, 3 (2000), 288–321.

[7] Cohen, W., Ravikumar, P., and Fienberg, S. A comparison of string metrics for matching names and records. In *KDD-2003 Workshop on Data Cleaning and Object Consolidation* (2003).

[8] Del Bimbo, A. *Visual information retrieval.* Morgan Kaufmann Publishers Inc., 1999.

[9] Dorneles, C. F., Lima, A. E. N., Heuser, C. A., da Silva, A., and de Moura, E. S. Acessing xml data by allowing imprecise query arguments. Tech. Rep. RP-342, UFRGS, 2004.

[10] Florescu, D., Kossmann, D., and Manolescu, I. Integrating keyword search into xml query processing. In *WWW Conf. on Computer networks* (2000).

[11] Fuhr, and Grossjohann. XIRQL: an extension of XQL for information retrieval. In *ACM SIGIR Workshop On XML and Information Retrieval* (2000).

[12] Gravano, L., Ipeirotis, P. G., Jagadish, H. V., Koudas, N., Muthukrishnan, S., and Srivastava, D. Approximate string joins in a database (almost) for free. In *VLDB* (2001).

[13] Gravano, L., Ipeirotis, P. G., Koudas, N., and Srivastava, D. Text joins in an rdbms for web data integration. In *WWW* (2003), ACM Press.

[14] Jin, L., Li, C., and Mehrotra, S. Efficient similarity string joins in large data sets. In *VLDB* (April 2002).

[15] Kamps, J., Marx, M., de Rijke, M., and Sigurbjrnsson, B. XML retrieval: What to retrieve? In *ACM SIGIR* (2003).

[16] Lawrence, S., Giles, C. L., and Bollacker, K. D. Autonomous citation matching. In *Intl. Conf. on Autonomous Agents* (Seattle, Washington, May, 1-5 1999).

[17] Lew, M. S., Ed. *Principles of Visual Information Retrieval.* Springer, 2001, ch. Feature Similarity, pp. 121–143.

[18] Motro, A. Vague: A user interface to relational datbases that permits vague queries. *ACM Transactions on Office Information Systems 6*, 3 (July 1988), 187–214.

[19] Nadvorny, C. F., and Heuser, C. A. Twisting the metric space to achieve better metric trees. In *Brazilian Symp. on Database, SBBD* (2004).

[20] Tejada, S., Knoblock, C. A., and Minton, S. Learning object identification rules for information integration. *Information Systems 26* (2001), 607–633.

[21] Tejada, S., Knoblock, C. A., and Minton, S. Learning domain-independent string transformation weights for high accuracy object identification. In *ACM SIGKDD* (2002).