A Cluster-Based Approach to XML Similarity Joins

Leonardo A. Ribeiro[*]
University of Kaiserslautern, Germany
AG DBIS, Department of Computer Science
ribeiro@cs.uni-kl.de

Theo Härder
University of Kaiserslautern, Germany
AG DBIS, Department of Computer Science
haerder@cs.uni-kl.de

Fernanda S. Pimenta[†]
UFRGS – Institute of Informatics
Porto Alegre, Brazil
fspimenta@inf.ufrgs.br

ABSTRACT

A natural consequence of the widespread adoption of XML as standard for information representation and exchange is the redundant storage of large amounts of persistent XML documents. Compared to relational data tables, data represented in XML format can potentially be even more sensitive to data quality issues because structure, besides textual information, may cause variations in XML documents representing the same information entity. Therefore, correlating XML documents, which are similar in content an structure, is a fundamental operation. In this paper, we present an effective, flexible, and high-performance XML-based similarity join framework. We exploit structural summaries and clustering concepts to produce compact and high-quality XML document representations: our approach outperforms previous work both in terms of performance and accuracy. In this context, we explore different ways to weigh and combine evidence from textual and structural XML representations. Furthermore, we address user interaction, when the similarity framework is configured for a specific domain, and updatability of clustering information, when new documents enter datasets under consideration. We present a thorough experimental evaluation to validate our techniques in the context of a native XML DBMS.

Categories and Subject Descriptors

H.2m [**Database Management**]: Miscellaneous—*Entity Resolution*; I.5 [**Pattern Recognition**]: Clustering—*Similarity measures*

General Terms

Design, Experimentation

[*]Supported by CAPES/Brazil under grant BEX1129/04-0
[†]Work done at the University of Kaiserslautern, Germany.

Keywords

similarity joins, XML, similarity measures, entity resolution, xml databases, clustering

1. INTRODUCTION

The quality of data stored in databases inevitably degrades over time. Common reasons for this behavior include data entry errors, such as typographical errors and misspellings, incomplete information, and varying conventions during integration of multiple data sources storing overlapping information. A common consequence of poor data quality is the appearance of the so-called *fuzzy duplicates*, i.e., multiple and non-identical representations of a real-world entity. Complications caused by such redundant information abound in common business practices, e.g., misleading downstream analysis due to erroneously inflated statistics, inability of correlating information related to the same customer, and repeated mailing operations. The problem of identifying fuzzy duplicates is commonly referred to as the *entity resolution* (ER) problem (also known as record linkage or near-duplicate identification).

As natural consequence of the widespread adoption of XML as standard for information representation and exchange, large amounts of persistent XML documents are redundantly stored in various places. Therefore, data represented in XML format potentially influences and jeopardizes the data quality delivered to applications even more compared to the relational world, because structure, besides textual information, may cause variations in XML documents representing the same information entity. Due to the much greater modeling flexibility of the XML model, even documents containing the same information and following the same schematic rule (i.e., a DTD or XML Schema) may be arranged in quite different structures. Moreover, because mapping flexibility and support of schema evolution are prime motivations for using XML, structural changes are likely to happen very often in many application domains.

Example 1. Consider the illustration in Figure 1, which shows an XML tree containing patient demographic information from a hospital. Although subtrees a) and b) refer to the same patient, it would be extremely difficult for an ER application to correctly identify them as fuzzy duplicates. The data in subtree a) is arranged by `patient`*, while the data of subtree b) is arranged by* `study`*. Further, there is the extra element* `relatives` *in subtree a). Moreover, there are typos and use of different abbreviation conventions between the content of the two subtrees.*

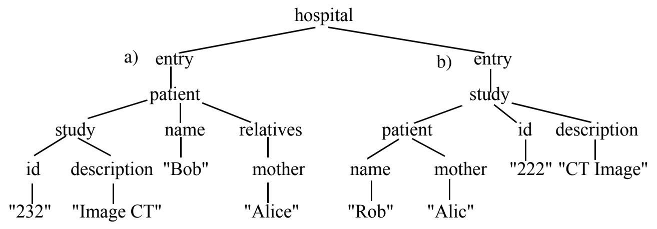


Figure 1: Heterogeneous XML data collection

Similarity join is a fundamental operation to support ER applications. This particular kind of join operation employs a predicate of type *sim* $(r, s) \geq \gamma$, where r and s are operand objects to be matched, *sim* is a *similarity function*, and γ is a constant threshold. In typical ER processes, pairs satisfying a similarity join can be classified as potential duplicates and selected for later analysis.

1.1 Specific XML-related Requirements

To perform similarity joins on XML datasets, the similarity predicate should address the hierarchical structure of the documents. For such kinds of similarity matching in XML tree structures, a large body of work is already available [3, 2, 6, 12, 17, 25, 33]. However, as far as identification of fuzzy duplicates is concerned, all previous work suffers from some combination of the following problems:

- *Limited support for textual similarity*: Many approaches focus on the structural similarity only; text nodes are either stripped away, before the matching process is initiated, or only simple equality operations on text values are considered. While structural information certainly conveys useful semantic information, most of the discriminating power of real-life XML data, i.e., the items of identifying information, which allow to distinguish documents in a collection from each another (e.g., keys), is assumed to be represented by text nodes. As a result, the resulting similarity notion is often too "loose" for ER applications or ineffective for datasets having poor textual quality.
- *Common Schema Assumption*: Some methods require that the documents to be matched follow the same schema, e.g., to locate the textual description information of objects. However, large parts of today's XML datasets are non-schematic or with multiple or evolving schemas. In these common scenarios, the effort of a unifying pre-step for structure alignment, when heterogeneous and ad-hoc XML documents are present, is prohibitively time-consuming.
- *Scalability*: Several similarity functions proposed have an expensive runtime complexity or, even worse, do not easily lend themselves to effective optimizations such as derivation of bounds or minimization of pairwise comparisons (candidate pruning). This limitation excludes such functions from being used in operations on large datasets.

An alternative way to overcome the first drawback is to decouple textual and structural information by considering them as distinct *document representations*. Such a separation is also important, because text and structure may have different requirements regarding their units of representation, i.e., their *tokens*, including aspects of feature identification and of weighting their relative importance. The use of multiple document representations brings up the issue of combining similarity evaluation results, which is commonly referred to as the *combination of evidence* problem.

The conceptual separation of text and structure for XML document representation does not exclude, however, the exploitation of structure to further improve the quality of textual similarity results. To this end, a common approach in XML retrieval is to qualify each textual unit by the XPath context, i.e., the root-to-leaf navigational path, in which it appears [7]. For ER applications, such *context segmentation* is crucial for the following reasons: first, to prevent token comparison among unrelated concepts and, second, to restrict the comparison of fields to those which are important for the resolution process, for example, while `author` and `title` are very significant in an XML database about publications, `year_of_publication` is expected to be much less informative. Moreover, it avoids distortion of feature statistics, e.g., a feature can appear in several fields but with different frequencies.

The ability of *approximately* matching navigational paths is very important for dealing with structural heterogeneity. In this context, the similarity evaluation should account not only for tokens appearing under *identical* paths, but also for the case when these paths are *similar* according to some similarity notion. A commonly used heuristic for the comparison of hierarchically structured data is to exploit the nesting level and to assign higher scores for elements matching at lower nesting depths. Finally, performance optimizations need to address two aspects to deal with large datasets. First, repeated path similarity evaluations should be avoided for tree patterns appearing several times in documents. Second, the set of tokens derived from XML trees can be potentially large requiring substantial overhead for pairwise comparisons.

1.2 Our Contribution

This paper makes the following contributions. We present an effective, flexible, and high-performance XML-based similarity join framework addressing all the issues above. We start by defining a set-overlap-based similarity function for XML paths. By using the nesting level of path elements in a monotonically decreasing way, we are able to obtain meaningful results for frequent types of path variations. We then apply this function for clustering the structural summary of all paths in a target collection of XML documents. The resulting set of path clusters is used to equip the original structural summary with the so-called *Path Cluster Identifiers* (PCIs), which identify all the paths belonging to the same cluster. Using PCIs, we are able to easily generate textual and structural representations of a document while keeping fine-granular feature statistics constrained at the cluster level. In this context, we explore approaches to weight structural tokens and to combine different representations of a document, namely techniques for combination of evidence at the feature and score levels. We also devise a cluster prototype for each path cluster, i.e., a structure subsuming all paths contained in a cluster. This structure is then used to match (partial) paths against its cluster prototype. The benefit of cluster prototypes is two-fold: users can specify which part of the documents will constitute their textual representation with vague knowledge about the underlying structure and existing clusters can be updated, as operations such as *insertion of a new document* add a new path to the structural summary. We quantify the accuracy and runtime performance of our approaches in the context of a native XML database management system.

The rest of this paper is organized as follows. After having outlined in Section 2 the related work, Section 3 gives the preliminaries. In Section 4, we introduce our approach to XML path clustering used to derive token-based representations from documents, whereas we present the corresponding similarity functions in Section 5. Aspects of the decomposition of a document into structural and textual representations are covered in Section 6. Performance and accuracy results for this novel approach are described in Section 7, before we wrap up with the conclusions in Section 8.

2. RELATED WORK

Entity resolution is normally conducted in a very complex process involving several stages, including data preparation, candidate generation, data matching, and merging of results [36]. Similarity joins can play different roles in this process [21]. Besides proving direct input to some classification model, similarity joins can, for example, be used to reduce the number of candidate pairs considered by other, usually much more expensive similarity measures—an operation known as blocking. As another example, similarity joins can be used to support training-set construction for machine learning approaches [5].

Weis and Naumann [35] address ER in the context of XML datasets. Relevant content of each target object—either predefined or heuristically identified—is extracted from XML data and stored in a relation (object description), and string similarity is applied to identify duplicates. This approach assumes all XML documents complying to the same schema; thus, its usage in ad-hoc or "schema-later" settings is impractical.

A very large body of work deals with XML structural similarity [12, 17, 25]. Most of these contributions address the problem of identifying documents valid for a similar DTD and have some of the drawbacks mentioned in Section 1.1. The tree edit distance [33]—and its many variants, e.g. [25]— is a popular measure for comparing trees. While there are algorithms that compute the edit distance between ordered trees in polynomial time, those for unordered trees are known to be NP-complete [37]. In data-centric XML, however, sibling order is unimportant and therefore such documents are better modeled as unordered trees. Tackling this problem, Augsten et al. [2] present the concept of *windowed pq-grams*. Informally, the *pq*-grams of a tree are its subtrees having a specific shape [3]. Windowed *pq*-grams extend *pq*-grams to deal with unordered trees. Another alternative is based on the concept of path shingles [6]. The path-shingles approach generates tokens by prepending each element node in a tree with the full path from the root to this node. Both techniques convert a tree into a set of tokens and could be used in our similarity join framework. We compare our method with these approaches in Section 7.3.

Guha et al. [13] presented a framework for XML similarity joins based on tree edit distance. To improve scalability, the authors optimize and limit distance computations by using lower and upper bounds for the tree edit distance as filters and a pivot-based approach for the partitioning of the metric space. However, these computations are still in $O(n^2)$ and the reduction of pairwise distance calculations heavily depends on a good choice of document "pivots" (reference set, in their nomenclature).

Here, we assume that element tag labels are drawn from a common vocabulary. This assumption is often reasonable for real-world heterogeneous XML stores, where structure can be arbitrarily arranged, but there is some understanding of a set of interesting tags [23]. When not the case, the underlying vocabularies have to be standardized before our similarity join algorithm takes place. But, vocabulary or ontology integration is beyond the scope of this paper.

Exploitation of structure has been extensively investigated in XML search [20]. In this context, the user may have only limited knowledge about the underlying structure of an XML collection. Therefore, similarly to our context, text nodes have to be located approximately, i.e., the path to the specific node does not need to match exactly the structural constraints expressed in the query. In [7], fuzzy and partial match of term contexts (root-to-leaf paths) is supported. A shortcoming of this approach is that path similarity has to be computed every time a query is evaluated. In our approach, path similarity is computed only once in a pre-processing step as described in Section 4.2.

Clustering of tree-structured data is another active research topic [22, 19, 1, 11]. Common approaches compute the similarity within a cluster by exploiting additional knowledge sources (e.g., a thesaurus) and context information [22] or frequent structural patterns in a data collection [1]. Our work is similar in spirit to the bag-of-tree-paths model proposed in [19], which represent tree structures by a set of paths. Moreover, approaches combining structure and content to achieve a specific clustering of XML documents have been proposed [11]. Here, we do not cluster XML documents directly; rather, we cluster (much smaller) structural summaries to derive compact document representations. Furthermore, because we aim at supporting ER processes, we have to support approximate matching on strings to account for textual deviations.

3. BACKGROUND

Before we introduce our approach, we make the underlying assumptions explicit and review the most important definitions of the subject area.

3.1 XML Data Model

XML documents are modeled as unordered labeled trees. We distinguish between element nodes and text nodes, but not between element nodes and attribute nodes; each attribute is child of its owning element. Disregarding other node types such as Comment, we consider only data of string type. Given a node n, $\delta(n)$ corresponds to the label of n, if n is an element node, or to n's value, if n is a text node. Henceforth, when no confusion arises, we use n to represent an (text) element node as well its (value) label.

3.2 Textual XML Representation

A common approach in IR is to characterize a document by a *profile*, i.e., a compact representation of its content. This process is known as *indexing* and we refer to the units of representation as *tokens*. In this paper, our representation of XML documents comprises textual and structural information. In the following, we define textual units in isolation (ignoring context) and defer the definition of structural units and context-aware textual representation to Section 5.1 and Section 5.2, respectively.

In IR, tokens are typically identified by words. For ER systems, however, *similarity matching* has to be applied to deal with text deviations and, therefore, more fine-granular representation units are necessary. The concept of q-gram is widely used in this context. Informally, a q-gram of a string s is a substring of s of size q. The *q-gram profile*, denoted by $I^q(s)$, is the set of q-grams obtained from a string s. The common procedure used to produce a q-gram profile consists on "sliding" a window of size q step by step over the characters of s. It is useful to (conceptually) extend each string s by prefixing and suffixing it with $q-1$ *null* characters (a *null* character is a special symbol not present in any string). Thus, each character contained in s participates in exactly q (of a total of $size(s)+q-1$) q-grams.

3.3 Indexing and Weighting

Many experiments consistently revealed that not all tokens are equally important for similarity evaluation [30, 8]. The process of quantifying this importance is referred to as *weighting*. Particularly, tokens that appear very frequently in a collection of documents contribute to the discrimination of them to a lesser degree, whereas rare tokens carry more content information and, thus, are more discriminative. For this reason, the *inverse document frequency* (*idf*) is normally used as token weight. The *idf* of a token t is inversely proportional to the total number of documents f_t, in which t appears in a collection of N documents. A typical *idf* weight is given by $idf(t) = ln(1 + N/f_t)$. The *term frequency* (*tf*), i.e., the frequency of a token in a document, is also used for weighting. Given a token t appearing $f_{d,t}$ times in a document, its *tf* weight can be computed as $tf(t) = 1 + ln(f_{d,t})$. The product of both term statistics constitutes the well-known *tf-idf* weighting scheme.

The utility of the *tf* weight for string-to-string similarity comparison has been questioned [14]. In IR, an user-formulated query is compared to normally much larger documents and it is reasonable to consider the frequency of a query token in a document as an indication that the document is relevant to the query. However for string-to-string comparison, the intuition is just the opposite: frequency discrepancy of the same token present in two strings should decrease their similarity. Moreover, ER systems usually perform comparisons on much shorter strings, e.g., author name fields. A simple solution consists of converting a profile I, composed by a bag of tokens, into a profile I_a, composed by a set of *annotated tokens*, by appending an ordinal number to each occurrence of a token. For example, the profile $I = \{a, b, b\}$ is converted to $I_a = \{(a,1), (b,1), (b,2)\}$. Annotated tokens have $f_{d,t} = 1$ and their weights are determined by the *idf* weight only. Furthermore, divergence in term frequencies of a token will penalize the similarity of the related strings by a stronger degree, because subsequent occurrences of the referenced token have decreased the document frequency in a collection (and, therefore, increased *idf* weight).

3.4 Set-Overlap-Based Similarity Functions

Set-overlap-based similarity functions evaluate the similarity of two objects based on the number of tokens they have in common. It is well-known that similarity functions are highly domain-dependent, where even functions performing well on average may poorly perform on some datasets. Therefore, an important property of this class of functions is versatility: it is possible to obtain diverse notions of similarity by measuring the overlap differently [31]. Next, we formally define the weighted version of the well-known Jaccard similarity, which will be used as building block for other similarity functions defined further in this paper; the weighted Jaccard has been shown to provide competitive effectiveness against several other popular (and often more computationally expensive) measures [8].

Definition 1. Let R be a set of tokens; $w_R(t)$ be the weight of token t relative to R according to some weighting scheme. Let the weight of R be given by $wt(R) = \sum_{t\in R} w_R(t)$. Similarly, consider a set S. The weighted Jaccard similarity (WJS) of R and S is defined as follows:

$$WJS(R,S) = \frac{wt(R\cap S)}{wt(R) + wt(S) - wt(R\cap S)} \tag{1}$$

where the set overlap of R and S is given by:

$$wt(R\cap S) = \sum_{t\in R\cap S} min(w_R(t), w_S(t)).$$

3.5 Set Similarity Joins

In the database community, there are two main processing models for efficient evaluation of joins employing set-overlap-based similarity functions. The first uses an *unnested* representation of sets in which each set element is represented together with the corresponding object identifier. Query processing commonly relies on relational database machinery: equi-joins supported by clustered indexes are used to identify all pairs sharing tokens, or elements of set surrogates, the so-called signature schemes [9], whereas grouping and aggregation operators together with UDFs are used for the final evaluation [9]. In the second model, an index is built for mapping tokens to the list of objects containing that token [31, 4]—for self-joins, such index can be dynamically performed as the query is processed. The index is then probed for each object to generate the set of candidates which will be later evaluated against the overlap constraint. Previous work has shown that approaches based on indexes consistently outperform relational-based approaches [4] (see also [14] for selection queries). As primary reason, a query processing model based on indexes provides superior optimization opportunities. A major method for that is to use index reduction techniques [4]; such techniques significantly minimizes the number of tokens to be indexed as well as the number of tokens that need to remain indexed as the set similarity join is processed [29].

3.6 Combination of Evidence

In the ER context, similarity matching results obtained from different combinations of factors such as document representation, similarity function, and weighting schemes can be viewed as multiple evidence that two objects are duplicates or not. Such pieces of evidence have to be merged in a meaningful manner before being used as input to some classification model. This is also a concern for document retrieval systems, which very commonly employ more than one retrieval and representation model for improving effectiveness or have to combine results from several search engines—the latter is known as the *meta-search problem.* In general, solutions proposed for the ER context are applicable to the IR context, and vice-versa. Common approaches consist of training classifiers—based on methods such as decision trees, Support Vector Machines (SVMs), and logistic regression—to learn a combination model [5, 10] or applying simpler functions such as the simple average on the (normalized) similarity values or on induced ranking to obtain a conciliated result [10]. When using a single similarity function and weighting scheme, it is possible to combine different document representations at the token level. Using a single collection statistics from all document representations, a linear combination weighted by the corresponding representations can be employed to form token weights [26]. Alternatively, token weights can be directly obtained from token collection statistics relative to each document representation in isolation [7]. Finally, multiple sources of evidence can be combined at token generation time. For example, one can produce tokens that jointly capture structural and textual properties of an XML tree [28].

4. XML PATH CLUSTERING

In this section, we first define a similarity function for navigational XML paths which is used in a cluster method to group paths of a structural summary. We then employ these clusters to derive representations of XML documents.

4.1 Path Similarity Function

We consider paths formed by element nodes. Token sets can be derived from paths by converting each element node in a given path into an annotated token. The motivation of using annotated tokens is different from that discussed in Section 3.3. Here, token annotation serves to deal with paths containing element recursion, e.g., paths containing multiple occurrences of the same node. For example, consider two paths p_1 = `/a/b/a` and p_2 = `/a/b` and their corresponding profile of annotated tokens $I_a(p_1) = \{(a,1),(b,1),(a,2)\}$ and $I_a(p_2) = \{(a,1),(b,1)\}$. Therefore, we have $I_a(p_1) \cap I_a(p_2) = \{(a,1),(b,1)\}$. Note that in this way, the matching of tokens derived from recursive labels is done from low to high nesting levels.

Further, a weighting scheme can be applied to express the relative node significance in a path. In hierarchically structured data, more general concepts in the corresponding domain are normally placed at lower nesting depths. Mismatches between two paths on such low-level concepts may suggest that the information contained in them are semantically more "distant". Therefore, an intuitive heuristics is to assign higher importance to nodes at lower nesting depths. Finally, given two paths represented as weighted sets, their similarity can be assessed using a set-overlap-based similarity function as defined in 3.4. Next, we formally define these concepts.

Definition 2. Let $p = (n_0, \ldots, n_n)$ be a path of element nodes, where node n_i is at nesting level i. The level-based weighting scheme assigns a weight to each path element, i.e., $LWS(p) = (n_0, w_0; \ldots; n_n, w_n)$ where:

$$w_i = e^{\beta i} \tag{2}$$

and $\beta \leq 0$ is a decay rate constant.

Definition 3. Let p_1 and p_2 be two paths. The weighted path similarity (WPS) between p_1 and p_2 is given by:

$$\mathit{WPS}(p_1, p_2) = \mathit{WJS}(\mathit{LWS}(p_1), \mathit{LWS}(p_2)) \tag{3}$$

using the weighted Jaccard defined in Equation 1.

Example 2. Consider p_1 = ***`patient/relatives/mother`*** *and p_2 =* ***`study/patient/mother`****, two (partial) paths appearing in the document shown in Figure 1. Applying LWS with decay rated $\beta = -0.1$, we obtain the following weighted sets: $\mathit{LWS}(p_1) = \{(p,1),(r,0.904),(m,0.818)\}$ from p1, and $\mathit{LWS}(p_2) = \{(s,1),(p,0.904),(r,0.818)\}$ from p2. The two sets share the tokens `patient` and `mother`; the weight value of 0.904 for the token `patient` is returned by the minimum function. Thus, the weighted overlap is 1.723. The weighted norm of $\mathit{LWS}(P_1)$ and $\mathit{LWS}(P_2)$ is 2.723. Thus, $\mathit{WPS}(p_1,p_2) = 1.723/(2.723 + 2.723 - 1.723) = 0.462$.*

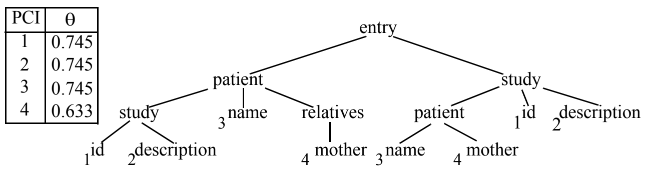
We observe that, owing to the strictly decreasing weighting rule of *LWS*, *WPS* may yield non-intuitive results, when applied to long paths. For example, depending on the value of decay rate β, two long paths sharing a smaller portion of element nodes at higher nesting levels, but having a larger part of unrelated nodes at lower levels, will have higher similarity according to *WPS*. One solution consists of making the weights constant from some level on. Therefore, the weighted norms in the denominator are kept sufficiently large to decrease the final result. On the other hand, empirical studies provide evidence that the vast majority of XML data worldwide has less than 8 levels [10] and, therefore, the above effect is hardly an issue in practice.

4.2 XML Representation based on Path Cluster Identifiers

We now exploit *WPS* to cluster *path classes*, i.e., paths that occur at least once in at least one document in a collection. All path classes in an XML dataset are typically maintained by index structures called *structural summaries* (also known as *1-Index* [24] or *path synopsis* [15]; see [24] for a formal definition). For example, the structural summary of the document shown in Figure 1 is depicted in Figure 2 (disregard the node identifiers and the accompanying table for the moment). Path summaries are instrumental for efficient XPath query evaluation;[1] therefore they are pervasively supported by XML DBMSs. Here, our goal is to exploit path summaries to derive compact structural surrogates of XML trees: we use a cluster method to group similar path classes and then the resulting cluster information to generate the representation of documents.

In the following, we describe the path clustering process in detail. Given a structural summary of a document collection, we start by specifying a target label $tg(l)$, corresponding to the entity to be matched (e.g., `entry` in Figure 1). Let $P^{tg(l)}$ be the set of all path classes relative to $tg(l)$, i.e., paths in the structural summary having $tg(l)$ as root label. In case of nested occurrences of $tg(l)$, we consider only the paths rooted by the topmost occurrence. We then use *WPS* to generate a *proximity matrix* containing all pairwise similarity values of $P^{tg(l)}$. This similarity matrix is the input for a cluster method to generate a set of path clusters (partitions) on $P^{tg(l)}$. In this paper, we use the UPGMA

[1]In reference [15], path summaries are exploited for designing space-economic storage models.



PCI	θ
1	0.745
2	0.745
3	0.745
4	0.633

Figure 2: Structural summary equipped with PCIs

Agglomerative Hierarchical Clustering method with a user-specified threshold as cutting point in the dendrogram [18].[2] We denote the set of path clusters generated from $P^{tg(l)}$ at cutting threshold θ with $PC_{\theta}^{tg(l)}$. A *Path Cluster Identifier* (PCI) is used to distinguish the path clusters in $PC_{\theta}^{tg(l)}$. Finally, we identify all path classes $p \in PC$ in a structural summary by assigning to them the corresponding PCI. For ease of notation, let the i be the corresponding PCI of a path cluster $pc^i \in PC_{\theta}^{tg(l)}$. Figure 2 illustrates for $tg(l) = \texttt{entry}$, $\theta = 0.6$, and decay rate of $\beta = -0.1$ for *LWS*. The values in the box on the left are the similarity values at which the clusters were formed.

After having equipped a structural summary with PCIs, we are able to automatically derive a token-based structural representation of XML documents. For this purpose, we decompose a document into a set of paths and, for each path p, the corresponding PCI is used as a structural token, i.e., i such that $p \in pc^i$. As a consequence, the structure of the document is represented as a set of PCIs, each token denoting an appearance of a path cluster element in the document. In Figure 2, note that subtrees, a) and b) are represented by the same PCI token set $\{1, 2, 3, 4\}$; as result, they would have maximum similarity according to any set-overlap-based similarity function, even though they have no path in common. This latter observation highlights a salient feature of our PCI-based representation: equality matching of single tokens incorporates similarity matching of whole paths for free. The actual path comparison is done only once during the clustering process thereby avoiding repeated path similarity computations when evaluating the similarity join operation. Next, we describe our PCI-based similarity functions in detail.

5. XML TREE SIMILARITY

In this section, we first define the subsets of $PC_{\theta}^{tg(l)}$ that are used to produce token sets for structure and content; then we describe the corresponding similarity functions. These functions are combined to define similarity functions for XML trees. (In the following, we omit the target label and the threshold cutting point in the notation for a set of path clusters to avoid clutter, i.e., $PC_{\theta}^{tg(l)} = PC$.)

Definition 4. Let C be a collection of XML documents; PC is a cluster set generated from the structural summary of C. We decompose PC into two disjoint sets, PC_s and PC_t, such that $PC_s \cup PC_t = PC$;[3] PC_t and PC_s do not form a partition, however, because one or the other set can be empty. We call PC_s the set of structural path clusters and PC_t the set of textual path clusters.

[2]Note that the specific clustering method is orthogonal to all the methods used in this paper. Of course, other techniques such as K-means and DBSCAN are applicable.

[3]Details of this decomposition are given in Section 6.

5.1 Structural Similarity

Given an XML document d from C, let P_d^s be the set of paths of d where each path is member of some path cluster in PC_s, i.e., $P_d^s = \{p : p \in pc \text{ and } pc \in PC_s\}$. Given P_d^s, we use the corresponding set of PCIs to obtain the profile of annotated tokens $I_a^s(D)$ of D. PCI tokens are generated as described in Section 4.2.

Definition 5. Let d_1 and d_2 be two XML documents and let $I_a^s(d_1)$ and $I_a^s(d_2)$ be their respective structural profile. The structural similarity function (PCI-S) is given by:

$$PCI\text{-}S\,(d_1, d_2) = WJS\left(I_a^s(d_1), I_a^s(d_2)\right) \qquad (4)$$

We now discuss the weighting scheme used for structural tokens. In the definition of *PCI-S*, we use annotated tokens, which rules out *tf*-based weighting schemes. However, it is possible to make a different rationale for structural tokens than that for textual tokens (Section 3.3). For example, because data collections associated with an XML schema may carry loose constraints on the data, e.g., allowing repeating and optional elements, node frequency discrepancies can naturally emerge among documents. In this sense, the *tf* weighting component, as defined in Section 3.3, or any other weighting formula that dampens the effect of token frequency in a document provides a good compromise. However, while this particular notion of similarity is reasonable when dealing with the identification of XML documents valid for a similar DTD [12, 17, 25], it is not clear whether or not a similarity evaluation based on such an approach would produce meaningful results in the ER context. We empirically evaluate different weighting schemes for structural tokens in Section 7.3.

5.2 Textual Similarity

Given a document d from C, let t^i be a text node of d appearing in a path cluster $pc^i \in PC_t$ (or more precisely, in a path $p \in pc^i$). Let $I_a^q\left(t^i\right)$ be the annotated set of q-grams of t^i as defined in Section 3.2. We convert $I_a^q\left(t^i\right)$ to $PCI^q\left(I_a^q\left(t^i\right), i\right)$, the set of the so-called *pci-grams* of t^i, by appending i, the PCI value of the path cluster pc^i, to each q-gram $q \in I_a^q\left(t^i\right)$. Further, let T_d^t be the set of text nodes t^i of d appearing in some path cluster $pc^i \in PC_t$. The profile of annotated *pci*-grams of d, denoted as PCI_d^q, is given by $PCI_d^q = \cup_{t^i \in T_d^t} PCI^q\left(I_a^q\left(t^i\right), i\right)$.

Definition 6. Let d_1 and d_2 be two XML documents, and let PCI_{d1}^q and PCI_{d2}^q be their respective textual profile. The textual similarity function, denoted as PCI-T, is given by:

$$PCI\text{-}T\,(d_1, d_2) = WJS\left(PCI_{d1}^q, PCI_{d2}^q\right) \qquad (5)$$

Note that, besides being used to compose the textual representation of a document, PCI also serves to approximately locate text nodes. For example, in Figure 1, we are able to compare the text value of `mother` in subtrees a) and b) because their respective paths have associated the same $PCI = 4$ (see Figure 2). Further, we consider only the *idf* weighting scheme for textual tokens. The collection statistics of a token t^i is constrained at the cluster path pc^i, i.e., f_{t^i} corresponds to the number of documents where token t appears in pc^i.

5.3 Similarity Combination

We now consider ways to combine textual and structural similarity. In this paper, we examine two approaches: score-level combination (SLC) and token-level combination (TLC). The first is the linear combination of the resulting scores of *PCI-S* and *PCI-T*. In this approach, the similarity functions are evaluated independently, possibly using different weighting schemes, and their result is combined using weights that can be either hand-tuned or obtained from some learning model. The second approach performs the union of the structural and textual profile of a document thereby obtaining a unified representation. In this approach, we use the same (*idf*) weighting scheme for both textual and structural tokens and a single similarity function. Note that textual tokens would have lower frequencies than structural tokens (and therefore greater *idf* weights). Next, we formally define both strategies.

Definition 7. Let d_1 and d_2 be two XML documents. Let λ_t and λ_s be weights such that $\lambda_t + \lambda_s = 1$. The score-level similarity combination (SLC) between d_1 and d_2 is given by:

$$SLC\left(d_1, d_2\right) = \lambda_s \times PCI\text{-}S\left(d_1, d_2\right) + \lambda_t \times PCI\text{-}T\left(d_2, d_2\right) \tag{6}$$

Definition 8. Let d_1 be an XML document. The combined representation of d_1, denoted as $C_a^{d_1}$, is given by $C_a^{d_1} = T_a^s(d_1) \cup PCI_{d1}^q$. Similarly, consider a document d_2. The token-level similarity combination (TLC) between d_1 and d_2 is given by:

$$TLC\left(d_1, d_2\right) = WJS\left(C_a^{d_1}, C_a^{d_2}\right) \tag{7}$$

6. DECOMPOSITION OF TEXTUAL AND STRUCTURAL REPRESENTATIONS

So far, we have described how we accomplish XML path clustering and how we compute textual and structural similarity. We are now ready to discuss the process of determining the sets PC_t and PC_s. Because this decomposition of the original set of clusters defines which parts of an XML document will deliver textual or structural tokens, it is a key design decision. As previously mentioned, textual information has more discriminating power than structural information in XML documents, in general. However, simply using all textual information available will hardly be effective. Issues such as lack of independence are known to negatively affect the quality of duplicate detection results when more fields than needed are used [36]. Moreover, textual tokens such as q-grams significantly increase the space overhead, thereby directly having negative impact on performance. Therefore, the ability to select specific parts of the textual information from XML documents is a fundamental requirement in our similarity join framework.

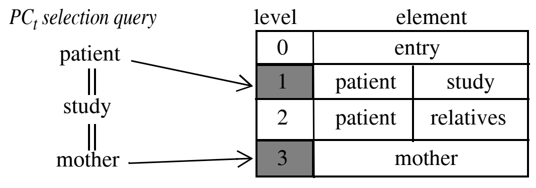
The above considerations about restrictions on the use of textual information bring similar concerns about structural information. The first question is whether to use it at all. The role of structure has been intensively explored in the context of XML retrieval. There exists empirical evidence that using structure in queries improves precision at low recall levels, but hurts performance at high recall levels [20]. In the ER context, the number of relevant answers is expected to be substantially smaller in comparison to XML retrieval.[4] In fact, because common real-world datasets contain small amounts of fuzzy duplicates, most XML trees have either no duplicates or only a small number of them. Therefore, the fraction of relevant elements for which the use of structure has been shown to enhance precision in XML retrieval (first retrieved elements), matches the typical recall range in the ER context. Furthermore, from a semantic viewpoint, structure has special importance regardless of its discriminative power. For example, two XML documents having a completely different structure would certainly be classified as non-duplicates. Finally, by representing the structure by a set of PCIs, we are able to derive very small token sets causing only a little performance penalty.

The definition of the best decomposition configuration is critically dependent on the application domain. Hence, we let users specify the PC_t set by issuing the so-called *PC_t selection queries*. This query consists of one or more simple path expressions that are approximately matched against the set of clusters. The top-k answers, i.e., the k cluster sets with highest similarity scores, are selected to constitute PC_t; the remaining cluster sets form PC_s. Because users are likely to have only vague knowledge about the underlying structure of the data collection, we support exploratory queries for cluster decomposition, before the actual similarity join evaluation takes place. Currently, we return several pieces of statistical information about the top-k results, such as cluster frequency in the collection and mean text value length of text nodes appearing in the cluster set. The user can therefore reformulate the query if, for example, the returned cluster set appears only in a few subtrees or is related to a part of the document with predominance of lengthy free text. We plan to enhance user guidance on text-cluster set selection with information-theoretic techniques [32].

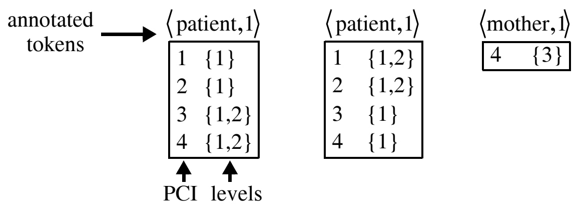
To avoid the burden of comparing path queries with all path classes of a cluster, we employ a *path cluster prototype*, a structure subsuming all elements of a cluster. Path queries are then matched with cluster prototypes rather than with each path class. To this end, we adopt a simple but effective level-based structure in which all labels appearing at the same level are kept together. Figure 3a shows the prototype of the path cluster with *PCI* 4 in our running example. The comparison between a *PC_t selection query* and a prototype is done using the *WPS* function, similarly to that of regular paths. The main difference is that only a single label match is allowed per level. For example, in the *PC_t selection query* shown in Figure 3a, the `study` component cannot be matched with the cluster prototype because of the previous match of `patient` at level 1.

We design a specialized inverted list index to represent the set of cluster prototypes and efficiently match PC_t selection queries against them. An inverted list is constructed for each annotated label appearing in a cluster prototype. Figure 3b shows the inverted lists of the annotated labels appearing in the example of Figure 3a. The space requirement of the index is $O\left(L \times P \times D\right)$, where L is the number of distinct annotated labels in the dataset—L can be greater than the number of distinct labels due to recursion, P is the number of distinct paths, and D is the maximum

[4]A common approach to evaluate the effectiveness of ER algorithms consists in considering a document from the dataset as a query and the number of true duplicates of this document as the set of relevant answers. Therefore, the use of standard IR evaluation measures is straightforward. We use this approach in this paper (see Section 7.2).



(a) Cluster prototype (PCI 4)



(b) Inverted list of cluster prototypes

Figure 3: Path cluster prototype matching

path length among all documents. Clearly, this analytical bound is "loose" and, in practice, the space consumption is expected to be much smaller. Moreover, the vast majority of XML datasets present fairly moderate values for L, P, and D, e.g., less than 200 path classes and maximum depth of 8 [15]. Therefore, for such datasets, the dictionary as well as the inverted list can be kept memory-resident.

Furthermore, we can ensure truly interactive responses even under concurrent requests by employing well-known IR optimizations such as the *max-score* strategy [34]. The order of evaluation of the inverted lists follows the order of a PC_t selection query components. This means that the upper bound for the weight of matching labels decreases monotonically as the query is evaluated (recall that we use the minimum as aggregation function in Equation 1) and therefore it is easy to compute the highest score that a cluster prototype containing all remaining components of the PC_t selection queries can achieve. Thus, we can promptly stop the query evaluation and return the current top-k candidate as soon as there is no other candidate in the intermediate result whose highest possible score is higher than the score of the kth candidate. In practice, only the inverted lists associated with the first query label components will need to be evaluated. Note that we are interested only in the top-k results; their complete score is unimportant.

Besides PC_t selection queries matching for cluster set decomposition, our inverted list representation of cluster prototypes is also important for (PCI-equipped) path summary maintenance in the presence of incremental path class updates. New path classes are matched against the index to automatically select the most similar path cluster. Therefore, the need of complete re-clustering is avoided after a new document is stored, or it can be postponed to be done off-line. A pre-defined threshold θ' is used to define a minimum value of closeness between a new path class and cluster prototypes. If no cluster prototype is returned with similarity to the new path class not less than θ', a new cluster is created and the path class is assigned to it. Typically, θ' is defined with the same value of the cutting threshold θ that was used to generated the initial set of path clusters (see Section 4.2).

7. EXPERIMENTS

After having presented our cluster-based approach to similarity joins, we are ready to show our experimental results. Our goals are: a) to evaluate the effectiveness of our PCI-based similarity functions for tree-structured data and compare it with other state-of-the-art token-based similarity functions; b) to investigate different weighting schemes for structural tokens; c) to evaluate different approaches for the combination of structural and textual similarities; d) to measure the performance of our similarity join framework on large datasets.

7.1 Datasets and Setup

For our empirical study, we used three real-world XML datasets: *Nasa*[5] containing astronomical data and two protein sequence databases, *SwissProt*[6], and *PIR-PSD*[7] (*PSD* for short). For all of them, we obtained sets of XML documents by deleting the root node of each XML dataset. The resulting documents are structurally heterogeneous. *Nasa* is irregular, has large trees, deep nesting levels, and is more document-centric (larger textual content per subtree). *SwissProt* and *PSD* are both more regular than *Nasa*, but the trees of *SwissProt* are larger and more shallow than those of *PSD*; in addition, *SwissProt* has the largest number of distinct tags. Detailed statistics describing the datasets are given in Table 1.

For all experiments, we produced "fuzzy" copies of these datasets by performing *random* transformations on content and structure. Injected errors on text nodes consist of word swappings and character-level modifications (insertions, deletions, and substitutions). In all experiments, we applied 1–5 such modifications for each dirty copy. These textual modifications simulated typical data quality issues, such as data entry errors. The structural modifications consist of node insertions and deletions, node inversions, and relabeling of nodes. Insertion and deletion operations follow the semantics of the tree edit distance [33], while node inversions switch the position between a node and its parent. Relabeling only changes the node's name (with a DTD-valid substitute). Additionally, we also applied modifications at the subtree level (deletions of entire subtrees) and at the path level (path deletions). We define *error extent* as the percentage of nodes from a subtree which were affected by the set of structural modifications. We considered as affected the node which received the modification (e.g., a rename) as well as all its descendants. With these modifications, we aimed at simulating kinds of structural heterogeneity which commonly arises in the context of semi-structured data management, such as heterogeneity owing to optional elements, schema evolution, and data integration. Finally, we classify the fuzzy copies generated from each data set according to the error extent used, i.e., we have low (10%), moderate (30%), and dirty (50%) error datasets.

We conducted all experiments and performance measurements using our prototype XML database system called XTC (XML Transactional Coordinator) [16]. All tests were run on an Intel Xeon Quad Core 3350 2.66-GHz Intel Pentium IV computer (two 3.2-GHz CPUs, about 2.5-GB main memory, Java Sun JDK 1.6.0) as the XDBMS server ma-

[5]http://www.cs.washington.edu/research/xmldatasets/
[6]http://us.expasy.org/sprot/
[7]http://pir.georgetown.edu/

Table 1: Dataset statistics

dataset	#subtrees	avg #nodes per subtree	# distinct tags/paths	avg/max path length	structure/ content ratio	avg/max node string size	avg/max tree string size
Nasa	2435	371	69/73	6,76/7	1,43	30,45/12668	4646,68/153588
SwissProt	50000	187	98/191	3,82/4	1,22	10,06/325	845,72/24029
PSD	262525	149	66/72	5,99/6	1,31	17,13/15535	1109,74/19074

chine where we configured XTC with a DB buffer of 250 8-KB-sized pages.

7.2 Evaluation Metrics

To evaluate the accuracy of similarity functions, we used our join algorithms as selection queries, i.e., as the special case where one of the join partners has only one entry. We proceeded as follows. Each dataset was generated by first randomly selecting 500 subtrees from the original dataset and then generating 4500 duplicates from them (i.e., 9 fuzzy copies per subtree), all together totaling 5000 subtrees. As the query workload, we randomly selected 100 subtrees from the generated dataset. For each queried input subtree T, the trees T_R in the result returned by our similarity join are ranked according to their calculated similarity with T. In this experiment, we did not use any threshold parameter and, therefore, the rank returned is *complete*. Finally, during data generation, we kept track of all duplicates generated from each subtree; those duplicates form a partition and carry the same identifier called *duplicate ID*. For each tree T, the set of corresponding relevant trees are those having the same duplicate ID. Here, we report the *non-interpolated Average Precision* (AP), which is given by:

$$AP = \frac{1}{\#relevanttrees} \times \sum_{r=1}^{n} [P(r) \times rel\,(r)] \qquad (8)$$

where r is the rank, n the number of subtrees returned. $P(r)$ is the number of *relevant* subtrees ranked before r, divided by the total number of subtrees ranked before r, and $rel\,(r)$ is 1, if the subtree at rank r is relevant and 0 otherwise. This measure emphasizes the situation where more relevant document fragments are returned earlier. The mean of the AP over the query workload is reported (abbreviated to MAP in our experimental charts). We also experimented with several other evaluation metrics such as the *11-point interpolated average precision* and the *F1* measure and obtained consistent results.

For performance experiments, we evaluated the algorithms according to their overall execution time (recorded as the average over 5 runs) and according to the token set size produced, which has a direct impact on the execution time.

7.3 Structural Similarity Results

In this first experiment, our objective was two-fold: 1) evaluating the effectiveness of our structural similarity measure in identifying fuzzy duplicates and comparing it to competing approaches; and 2) comparing different weighting strategies for structural tokens. To evaluate the contribution of the structural similarity measure in isolation, we ignored all text nodes in this experiment, i.e., we made $PC_t = \varnothing$.

We compared our PCI-based similarity function (*PCI-S*) against two other token-based approaches: windowed *pq*-grams (*WPQ*) and path-shingles (*PS*). For the two competing approaches, we used the parameters as recommended by the authors, i.e., $p = q = 2$ and $w = 3$ for *WPQ* and a window of size 1 for *PS*. In this and all subsequent experiments, we used a decay rate of 0.1 for *LWS* and a threshold of 0.4 as cutting point for the dendogram. We observed the best results with this configuration. Finally, we also considered an additional similarity function (*PATH*), which simply used the weighted Jaccard formulation (*WJS*)—as defined in Section 3.4—between the sets of root-to-leaf paths of two trees. Thus, we could observe the specific effects of our path similarity approach to the result quality.

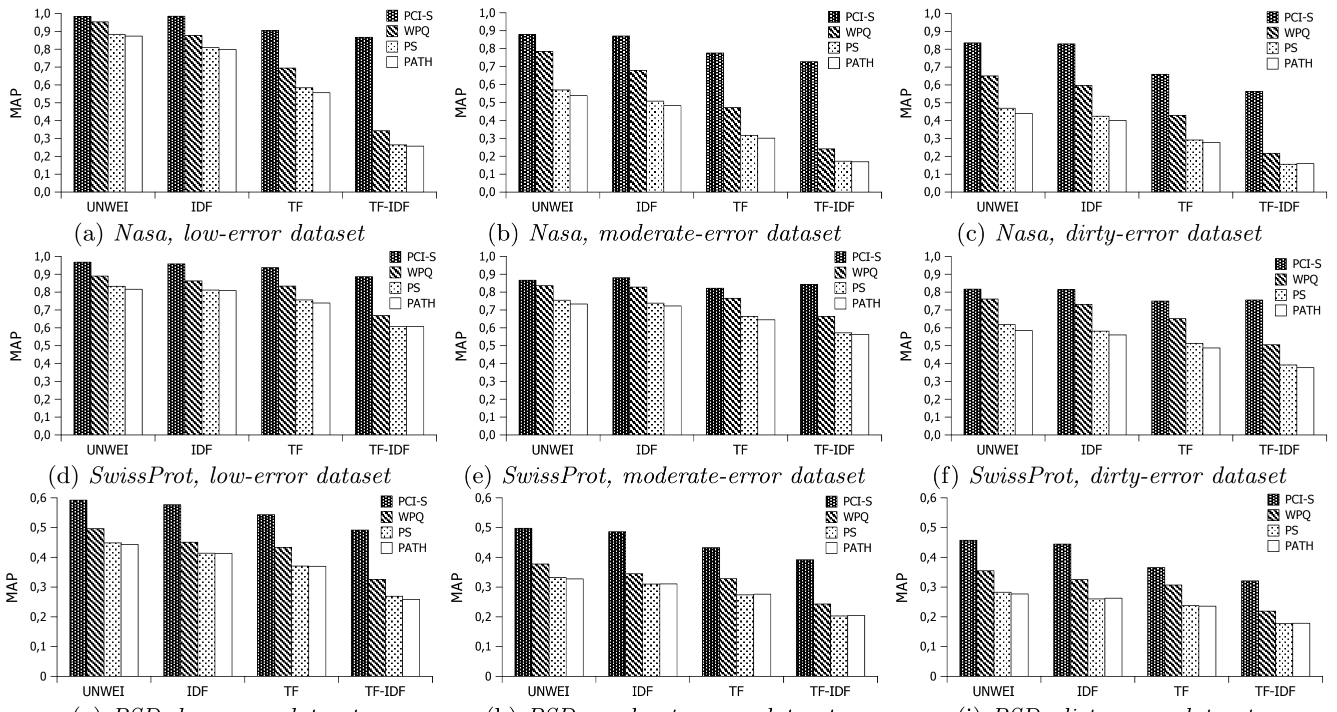
Further, we present results for four different weighting schemes: unweighted (*UNWEI*), inverse document frequency (*IDF*), term frequency (*TF*), and *tf-idf* (TF-IDF). For *UNWEI* and *IDF*, we used *WJS*. On the other hand, *TF* and *TF-IDF* are based on local statistics (term frequency). As a result, token weights can vary among different sets. Using *minimum* as aggregation function ensures that similarity results are in the interval $[0, 1]$. However, we experienced unstable results with this formulation. Hence, we refrained from using Jaccard and, instead, employed the well-known cosine similarity with *TF* and *TF-IDF* weighting schemes [30]. We also tested the cosine version of *IDF*; however, we consistently observed superior results with Jaccard.

Figure 4 shows the results. In all the settings, *PCI-S* is the most effective. Moreover, *PCI-S* is the most resistant, when the error extent increases, and shows only slight accuracy degradation as the error extent increases from 10% to 50% (low to dirty dataset). On the other hand, the accuracy of the other three similarity measures, *WPQ*, *PS*, and *PATH*, drops significantly as the error extent increases; it practically decreases at the same rate. *WPQ* performs better than *PS* and *PATH*, but it is close to *PCI-S* only on low-error datasets.

To better understand the reasons for the superiority of *PCI-S* over the other approaches, we conducted additional experiments on datasets generated by only one kind of structural modification. We observed that the results favoring *PCI-S* are, in general, more prominent on datasets generated by node-level modifications, i.e., node insertion, rename, inversion, and deletion. This observation is not a surprise, because the other measures cannot capture modifications that change the root-to-leaf path of an element.

All similarity functions show better results on *Nasa*. This is expected because *Nasa* has the most irregular structure, therefore providing a good structural "signature" to identify a subtree. Accordingly, the worst results are observed for the *PSD* dataset, which has the most regular structure and many subtrees have a very similar shape. Interestingly, the accuracy gap between *PCI-S* and the other measures is larger in this restricted feature domain.

The better results of *PCI-S* are even more impressive when we observe the average token set size delivered by each representation in Table 2 . For all datasets, *PCI-S* produces



(a) *Nasa, low-error dataset* (b) *Nasa, moderate-error dataset* (c) *Nasa, dirty-error dataset*

(d) *SwissProt, low-error dataset* (e) *SwissProt, moderate-error dataset* (f) *SwissProt, dirty-error dataset*

(g) *PSD, low-error dataset* (h) *PSD, moderate-error dataset* (i) *PSD, dirty-error dataset*

Figure 4: MAP values for different structural similarity functions on different datasets

Table 2: Average structural token set size

	Nasa	*SwissProt*	*PSD*
PCI-S	**143**	**79**	**60**
PS	360	181	145
WPQ	693	295	262

token sets about 5 times shorter than *WPQ*. To be resistant to subtree permutations, *WPQ* has to produce an increased number of *pq*-grams [2], while for *PCI-S* the number of tokens is always bounded by the number of paths. Moreover, the use of Path Cluster Identifiers provides a very compact representation for free, without the need of using any codification method such as a fingerprint hash function.

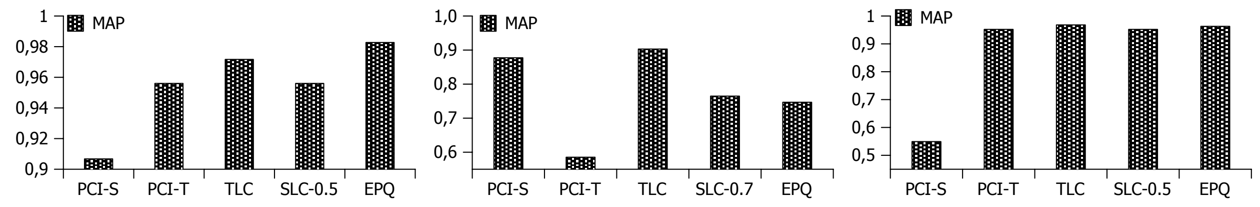
We now analyze the results of the comparison of weighting schemes. The main observation is that *UNWEI* and *IDF* perform consistently better than *TF* and *TF-IDF*. This result might be due to the use of Jaccard for *UNWEI* and *IDF*. In general, unweighted and *IDF*-weighted versions of Jaccard show very similar results for all similarity functions, with a slight advantage for the former. It seems that the distribution of structural tokens does not provide any significant advantage for *IDF*-weighting schemes. We also note that *TF* significantly outperforms *TF-IDF* in all token set representations. Similar results have been observed in the context of clustering web pages [27]. The common conjecture is that *TF-IDF* over-emphasizes the rarest terms. In our case, we believe that mismatches on such over-weighted (structural) tokens penalize too much the similarity result.

7.4 Accuracy Results for Text and Structure Combination

Here, we compare approaches to combine textual and structural similarity. We evaluate the methods presented in Section 5.3: score-level combination (*SLC*) and token-level combination (*TLC*). For *SLC*, we report the results using the *idf* weighting scheme with Jaccard as similarity function (as initially defined). We didn't observe any accuracy gain by using unweighted structural tokens. Also, normalizing the score results provided no accuracy enhancing, so the results reported were obtained without any normalization method. In addition, we also compare the *epq-grams* (*EPQ*) approach [28], an extension of *pq*-grams to capture textual information. Note that *pq*-grams and, in turn, *EPQ* can only handle ordered XML trees (in contrast to windowed *pq*-grams or our *PCI*-based methods). However, as we didn't apply node-swapping operations when generating the datasets, our comparison here is fair. The *EPQ* approach employs the so-called *path predicates*, a different mechanism to support selection of specific document parts used for textual representation. Path predicates support partial matching (e.g. `//a`), but not approximate matching as PC_t *selection queries*. See [28] for details. Finally, to better understand the effects of combining structural and textual similarity, we also report the results concerning the evaluation of *PCI-S* and *PCI-T* in isolation.

The following PC_t *selection queries* (path predicates) were used for the PCI-based methods (*epq*-grams): `/dataset/name`, (*Nasa*), `/Entry/Ref/Author` (*SwissProt*), and `/ProteinEntry/sequence` (*PSD*). We only report the results for moderate-error datasets due to space constraints. We used $q = 3$ for the PCI-based functions and $p = 2$ and $q = 3$ *EPQ*. In the experimental charts, we indicate for *SLC* the structural weight (λ_s) we used to achieve the best results. For example for *SwissProt*, we obtained better performance with a structural (textual) weight of 0.7 (0.3).

Figure 5 shows the results. *TLC* has the best results, in general, and exhibits good performance in all datasets.



(a) *Nasa* (b) *SwissProt* (c) *PSD*

Figure 5: Complete accuracy results on moderate-error datasets

Table 3: Average token set size for *TLC* and *EPQ*

	SwissProt	*PSD*
TLC	**198**	**432**
EPQ	417	903

Moreover *TLC* shows better accuracy than *PCI-T* and *PCI-S* in isolation in all settings. Even when *PCI-T* (on *SwissProt*) or *PCI-S* (on *PSD*) provided poor results, the overall accuracy of *TLC* never worsened. This result is similar to the findings of [26], which observed that good document representations tend to be robust when combined with other poorly performing representations. *SLC* obtained good results except for *SwissProt*. For this dataset, *PCI-T* also showed very poor results. A closer examination revealed that the clustering method grouped some unrelated paths in the same cluster. In this situation, *TLC* was able to leverage the results of *PCI-S* and didn't suffer any significant degradation w.r.t. accuracy. Finally, *EPQ* provided good results on *PSD* and *Nasa*; it was even able to slightly outperform *TLC* on *Nasa*. Similarly to the *PCI*-based methods on *SwissProt*, the path-predicate mechanism was not able to correctly locate its text nodes.

7.5 Performance Results

This last experiment quantifies the runtime performance of *TLC*, *SLC*, and *EPQ* for varying query thresholds. We used fuzzy copies of the *SwissProt* and *PSD* datasets, each copy generated with an error extent of 30% (moderate-error) and containing 100k subtrees. We implemented the *w-mpjoin* algorithm, a (self) set similarity join algorithm presented in [29]. For SLC, we first evaluated *PCI-T* and afterwards textitPCI-S; therefore this query was evaluated in conjunctive mode, where only the subtrees returned by *PCI-T* were considered by *PCI-S*. The input of *w-mpjoin* is a set collection sorted in increasing order of the set size; the elements of each set are sorted in decreasing order of the document frequency. We sorted the input in advance, so the input sorting time is not contained in the results.

The results are shown in Figure 6. The runtime performance of both *TLC* and *SLC* is practically identical. *EPQ* is slower than *TLC* and *SLC* by a factor of up to 2.5. This result reflects the average token-set size produced by each approach, which is given in Table 3. *EPQ* delivers about the double size of the PCI-based approaches on both datasets. All algorithms are about 6 times faster on the *SwissProt* dataset. The reason is that the token sets generated from *SwissProt* are smaller and, more importantly, have a wider set-size distribution, which is exploited by the *w-mpjoin* algorithm to optimize join processing [29].

7.6 Experimental Summary

PCI-S is clearly the structural XML representation of choice: it provides superior accuracy results, its performance degrades only moderately as the dataset level of " dirtyness" increases, and it is able to correlate reasonably well trees with restricted structural information. Moreover, *PCI-S* delivers much smaller token sets—bounded by the number of paths in a tree, the tokens lend themselves to very compact representations, and can be generated for free with support of a PCI-equipped structural summary.

We found the Jaccard similarity being superior to Cosine. Looking at evaluation of structural similarity in isolation, we didn't observe any significant gain using weighting schemes.

Regarding the combination of structural and textual similarity, *TLC* is the winner. It is able to leverage very well both kinds of similarity providing stable results even when the textual or structural similarity performs poorly. Moreover, *TLC* runs faster in our similarity join framework.

8. CONCLUSION

In this paper, we presented an XML-based similarity join framework, which is amenable to an integration into native XML database systems. We exploited structural summaries and clustering concepts to represent XML documents as token sets that capture common textual and structural deviations occurring in real-world XML datasets. Remarkably, our approach delivered substantially more compact token sets than competing approaches, while providing more accurate results. In this context, we explored different ways to weigh and combine evidence from textual and structural XML representations. Finally, we addressed user interaction, when the similarity framework is configured for a specific domain, and updatability of clustering information, when new documents enter datasets under consideration, by devising a cluster prototype structure represented as little memory-resident inverted lists.

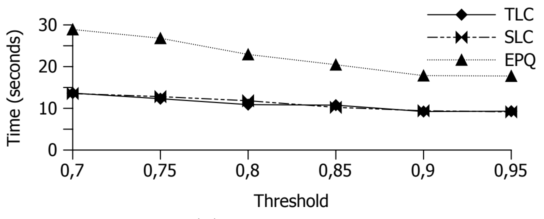
9. REFERENCES

[1] C. C. Aggarwal, N. Ta, J. Wang, J. Feng, and M. J. Zaki. Xproj: a framework for projected structural clustering of xml documents. In *Proc. KDD Conf.*, pages 46–55, 2007.

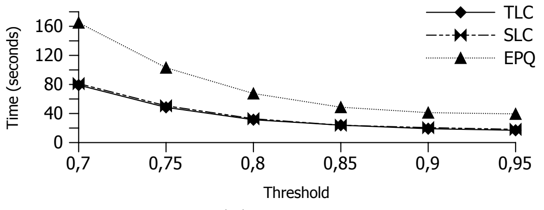
[2] N. Augsten, M. H. Böhlen, C. E. Dyreson, and J. Gamper. Approximate joins for data-centric xml. In *Proc. ICDE Conf.*, pages 814–823, 2008.

[3] N. Augsten, M. H. Böhlen, and J. Gamper. Approximate matching of hierarchical data using pq-grams. In *Proc. VLDB Conf.*, pages 301–312, 2005.

[4] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proc. WWW Conf.*, pages



(a) *SwissProt*



(b) *PSD*

Figure 6: Runtime experiments

131–140, 2007.

[5] M. Bilenko and R. J. Mooney. On evaluation and training-set construction for duplicate detection. In *Proc. KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 7–12, 2003.

[6] D. Buttler. A short survey of document structure similarity algorithms. In *Proc. Intl. Conf. on Internet Computing*, pages 3–9, 2004.

[7] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching xml documents via xml fragments. In *Proc. SIGIR Conf.*, pages 151–158, 2003.

[8] A. Chandel, O. Hassanzadeh, N. Koudas, M. Sadoghi, and D. Srivastava. Benchmarking declarative approximate selection predicates. In *Proc. SIGMOD Conf.*, pages 353–364, 2007.

[9] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proc. ICDE Conf.*, page 5, 2006.

[10] W. B. Croft. Combining approaches to information retrieval. *Advances in information retrieval*, 7:1–36, 2000.

[11] L. Denoyer and P. Gallinari. Overview of the inex 2008 xml mining track. In S. Geva, J. Kamps, and A. Trotman, editors, *Proc. INEX 2008*, LNCS, 2009.

[12] S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. Fast detection of xml structural similarity. *TKDE*, 17(2):160–175, 2005.

[13] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Integrating xml data sources using approximate joins. *TODS*, 31(1):161–207, 2006.

[14] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava. Fast indexes and algorithms for set similarity selection queries. In *Proc. ICDE Conf.*, pages 267–276, 2008.

[15] T. Härder, C. Mathis, and K. Schmidt. Comparison of complete and elementless native storage of xml documents. In *Proc. IDEAS Conf.*, pages 102–113, 2007.

[16] M. P. Haustein and T. Härder. An efficient infrastructure for native transactional xml processing. *DKE*, 61(3):500–523, 2007.

[17] S. Helmer. Measuring the structural similarity of semistructured documents using entropy. In *Proc. VLDB Conf.*, pages 1022–1032, 2007.

[18] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice Hall College Div, 1988.

[19] S. Joshi, N. Agrawal, R. Krishnapuram, and S. Negi. A bag of paths model for measuring structural similarity in web documents. In *Proc. KDD Conf.*, pages 577–582, 2003.

[20] J. Kamps, M. Marx, M. de Rijke, and B. Sigurbjörnsson. Articulating information needs in xml query languages. *TOIS*, 24(4):407–436, 2006.

[21] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *Proc. SIGMOD Conf.*, pages 802–803, 2006.

[22] M.-L. Lee, L. H. Yang, W. Hsu, and X. Yang. Xclust: clustering xml schemas for effective integration. In *Proc. CIKM Conf.*, pages 292–299, 2002.

[23] Z. H. Liu and R. Murthy. A decade of xml data management: An industrial experience report from oracle. In *Proc. ICDE Conf.*, pages 1351–1362, 2009.

[24] T. Milo and D. Suciu. Index structures for path expressions. In *Proc. ICDT Conf.*, pages 277–295, 1999.

[25] A. Nierman and H. V. Jagadish. Evaluating structural similarity in xml documents. In *Proc. WebDB Workshop*, pages 61–66, 2002.

[26] P. Ogilvie and J. P. Callan. Combining document representations for known-item search. In *Proc. SIGIR Conf.*, pages 143–150, 2003.

[27] D. Ramage, P. Heymann, C. D. Manning, and H. Garcia-Molina. Clustering the tagged web. In *Proc. WSDM Conf.*, pages 54–63, 2009.

[28] L. A. Ribeiro and T. Härder. Evaluating performance and quality of xml-based similarity joins. In *Proc. ADBIS Conf.*, pages 246–261, 2008.

[29] L. A. Ribeiro and T. Härder. Efficient set similarity joins using min-prefixes. In *Proc. ADBIS Conf. (to appear)*, 2009.

[30] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc. New York, 1983.

[31] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *Proc. SIGMOD Conf.*, pages 743–754, 2004.

[32] F. Sebastiani. Machine learning in automated text categorization. *CSUR*, 34(1):1–47, 2002.

[33] K.-C. Tai. The tree-to-tree correction problem. *JACM*, 26(3):422–433, 1979.

[34] H. R. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Information Processing Management*, 31(6):831–850, 1995.

[35] M. Weis and F. Naumann. Dogmatix tracks down duplicates in xml. In *Proc. SIGMOD Conf.*, pages 431–442, 2005.

[36] W. Winkler. Overview of record linkage and current research directions. Technical report, Statistical Research Division, U.S. Bureau of the Census, 2006.

[37] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters (IPL)*, 42(3):133–139, 1992.