Windowed pq-grams for approximate joins of data-centric XML

Augsten, Nikolaus ; Böhlen, Michael H ; Dyreson, Curtis ; Gamper, Johann

Abstract: In data integration applications, a join matches elements that are common to two data sources. Since elements are represented slightly different in each source, an approximate join must be used to do the matching. For XML data, most existing approximate join strategies are based on some ordered tree matching technique, such as the tree edit distance. In data-centric XML, however, the sibling order is irrelevant, and two elements should match even if their subelement order varies. Thus, approximate joins for data-centric XML must leverage unordered tree matching techniques. This is computationally hard since the algorithms cannot rely on a predefined sibling order. In this paper, we give a solution for approximate joins based on unordered tree matching. The core of our solution are windowed pq-grams which are small subtrees of a specific shape. We develop an efficient technique to generate windowed pq-grams in a three-step process: sort the tree, extend the sorted tree with dummy nodes, and decompose the extended tree into windowed pq-grams. The windowed pq-grams distance between two trees is the number of pq-grams that are in one tree decomposition only. We show that our distance is a pseudo-metric and empirically demonstrate that it effectively approximates the unordered tree edit distance. The approximate join using windowed pq-grams can be efficiently implemented as an equality join on strings, which avoids the costly computation of the distance between every pair of input trees. Experiments with synthetic and real world data confirm the analytic results and show the effectiveness and efficiency of our technique.

DOI: https://doi.org/10.1007/s00778-011-0254-6

Windowed pq-Grams for Approximate Joins of Data-Centric XML

Nikolaus Augsten · Michael Böhlen · Curtis Dyreson · Johann Gamper

Abstract In data integration applications, a join matches elements that are common to two data sources. Since elements are represented slightly different in each source an approximate join must be used to do the matching. For XML data, most existing approximate join strategies are based on some ordered tree matching technique, such as the tree edit distance. In data-centric XML, however, the sibling order is irrelevant, and two elements should match even if their subelement order varies. Thus, approximate joins for data-centric XML must leverage unordered tree matching techniques. This is computationally hard since the algorithms cannot rely on a predefined sibling order.

In this paper we give a solution for approximate joins based on *unordered* tree matching. The core of our solution are *windowed pq-grams,* which are small subtrees of a specific shape. We develop an efficient technique to generate windowed pq-grams in a three-step process: sort the tree, extend the sorted tree with dummy nodes, and decompose the extended tree into windowed pq-grams. The windowed pq-gram distance between two trees is the number of pq-grams that are in one tree decomposition only. We show that our distance is a pseudo-metric and empirically demonstrate that it effectively approximates the unordered tree edit distance. The approximate join using windowed pq-grams can be efficiently implemented as an equality join on strings, which avoids the costly computation of the distance between every pair of input trees. Experiments with synthetic and real world data confirm the analytic results and show the effectiveness and efficiency of our technique.

Nikolaus Augsten, Free University of Bozen-Bolzano (Italy), E-mail: augsten@inf.unibz.it · Michael Böhlen, University of Zürich (Switzerland), E-mail: boehlen@ifi.uzh.ch · Curtis Dyreson, Utah State University (USA), E-mail: curtis.dyreson@usu.edu · Johann Gamper, Free University of Bozen-Bolzano (Italy), E-mail: gamper@inf.unibz.it

1 Introduction

When XML data from different sources is integrated, data items that correspond to the same real world object must be matched. Exact matches often fail due to inconsistent representations and missing global keys, and approximate matching techniques must be applied. For instance, when companies merge, their customer data needs to be integrated, but the companies may have different ways to represent such data. As another example, an internet shop may want to enrich its product description with data provided by third parties, which each have slightly different descriptions for the same product.

Since an XML document can be modeled as an ordered labeled tree, one way to approximately match a pair of XML documents is to compute the *minimal tree edit distance* between the documents [12, 19, 26]. The tree edit distance between two trees is the minimum number of node insertions, deletions, and renamings that transform one tree into the other. Although order is important in document-centric scenarios (e.g., paragraph tags in XHTML), most applications of data-centric XML ignore the sibling order when data items are matched. This makes tree matching more difficult since all sibling permutations need to be considered. While the *ordered* tree edit distance (which relies on a predefined sibling order) can be computed in polynomial time, the *unordered* tree edit distance (which ignores the sibling order) is NP-complete [44].

In many other applications data are modeled as unordered trees, and the similarity between different trees must be computed. In software engineering, branches of code that have evolved independently must be compared [15]. In computational biology, unordered trees are used to model glycans [2], the evolution of genes [44], and the linage of cells or immunoglobulin gene mutants [21]. Backup and file syn-

chronization tools compute differences between directory structures of file systems [9].

This paper develops an efficient, approximate join for data-centric XML, where the sibling order is ignored. Our solution is based on *windowed pq-grams,* which are small subtrees of a specific shape. We develop a technique to systematically generate the set of windowed pq-grams in a three-step process: sort the tree, extend the sorted tree with dummy nodes, and decompose the extended tree into windowed pq-grams. The windowed pq-gram distance between two trees is the number of windowed pq-grams that are in one tree decomposition only. Two trees are similar if they share many windowed pq-grams, and dissimilar otherwise.

A windowed pq-gram is a small subtree of the original tree and consists of a *stem* (a path with p nodes) and a *base* (a sequence of q sibling nodes). While stems are obviously invariant to the sibling order, the main challenge is to define bases that are independent of the ordering of siblings. In this paper we identify three core properties that such bases should fulfill, namely preservation of structure information, preservation of children information, and preservation of sibling information, and we show how to construct such bases.

The windowed pq-gram distance approximates the unordered tree edit distance, which is defined as the minimal edit sequence that transforms one tree into the other, while allowing zero cost permutations of the siblings between edit operations, disregarding in this way the sibling order. The tree sorting approach used for windowed pq-grams is not applicable to other common distances between ordered trees. In particular, the ordered tree edit distance in combination with tree sorting cannot be used to approximate the unordered tree edit distance (cf. Section 4). The permutations between the edit operations permit powerful edit sequences that cannot be expressed with a single permutation of the input trees.

We provide an algorithm to compute the windowed pq-gram distance in $O(n \log n)$ time (n is the number of tree nodes) and to approximately join two sets of trees using windowed pq-grams. Most joins that are based on distance measures, such as the tree edit distance, must evaluate the distance between every pair of input trees using a nested loop join since there is no effective way to sort sets of trees or partition them into buckets. By serializing windowed pq-grams, our algorithm reduces the approximate join to an equality join on strings and can take advantage of well-known join optimization techniques.

To summarize, the paper makes the following contributions:

- We define the *windowed pq-gram distance* for unordered tree matching. Windowed pq-grams are computed in three steps: sort, extend, and decompose the tree. We prove that the sorting error is independent of the tree size, making sorting a valid approach for windowed pq-grams.
- We formulate three *desiderata* for tree decompositions that are invariant to the sibling order: the decomposition shall not change the impact of edges (preservation of structure information), shall not change the Jaccard similarity between the children of nodes (preservation of children information), and shall encode information about siblings (preservation of sibling information). We investigate these properties for windowed pq-grams and show for which values of p (stem size), q (base size), and w (window size) windowed pq-grams are optimal.
- We propose the *windowed pq-gram join* that joins similar trees. Our join algorithm does not need to compute the distance between all tree pairs, but reduces the distance join to an equality join on strings and therefore can take advantage of well-known join optimization techniques.
- We *analytically show the scalability* of our approach. The number of windowed pq-grams is bound by $O(n)$, where n is the number of tree nodes, and the pq-gram distance is computed in $O(n \log n)$ time.
- Our *extensive experiments* confirm the analytic results. Windowed pq-grams scale to very large trees, and the pq-gram join clearly outperforms the standard distance join. Windowed pq-grams are effective for joining real world XML and for approximating the unordered tree edit distance.

The rest of the paper is organized as follows. Section 2 presents related work. We motivate the approximate join of data-centric XML in Section 3 and discuss the impact of the sibling order on the tree distance computation in Section 4. Section 5 states desiderata for tree decompositions that ignore the sibling order. Windowed pq-grams are introduced in Section 6. Section 7 investigates the properties of windowed pq-grams and shows how to choose optimal windowed pq-grams. Section 8 provides algorithms, which are experimentally evaluated in Section 9. In Section 10 we draw conclusions and point to future work.

2 Related Work

Most papers that investigate techniques to compare XML documents represent the XML data as trees with labeled nodes. Tree matching techniques are applied to compute the similarity between trees. A well known distance function for trees is the tree edit distance, which is defined as the minimum number of edit operations (node insertion, deletion and renaming) that transform one tree into another [33]. The best known ordered tree edit distance algorithms [11, 14, 24, 43] have at least $O(n^3)$ runtime for trees with n nodes. The unordered tree edit distance problem is NP-complete [44].

Guha et al. [19] present an approximate XML join based on the ordered tree edit distance. They give upper and lower bounds for the tree edit distance that can be computed in $O(n^2)$ time and use reference sets to take advantage of the fact that the tree edit distance is a metric, thereby reducing the actual number of distances to compute in a join. Guha et al. [19] do not address joins of unordered XML.

Garofalakis and Kumar [18] discuss approximate joins in the context of data streaming applications. They focus on performing a match in a limited amount of space and present an efficient approximation of the ordered tree edit distance. Unordered tree matching is not addressed.

pq-Grams were introduced by Augsten et al. [4] as an effective and efficient approximation of the ordered tree edit distance. The pq-gram distance provides a lower bound of the ordered tree edit distance [6]. Ribeiro and Härder [29] introduce extended pq-grams for ordered trees that consider in addition to the structural similarity between trees also the string similarity between leaf nodes.

Windowed pq-grams extend pq-grams to approximate the unordered tree edit distance for data-centric XML. This paper extends our previous work on joining data-centric XML [3] with desiderata for tree decompositions, a declarative definition of windowed pq-grams, proofs of the optimality of windowed pq-grams, and empirical results that show that the ordered tree edit distance fails to approximate the unordered tree edit distance for data-centric XML.

Ribeiro et al. [30] present a distance for data-centric XML that considers both the tree structure and the text values of the leaf nodes. The paths of all trees in a dataset are clustered and the structure of a tree is represented by the set of cluster identifiers in which its paths participate. The text values are represented by their q-grams [36]; two text values can match only if their paths participate in the same cluster. The user configures the method with a weighting factor for the path similarity, a cutoff threshold for the hierarchical clustering, and a path query that selects a subset of path clusters on which the text value similarity is evaluated. These parameters depend on the application and are non-trivial to choose. Windowed pq-grams are defined by the shape parameters p and q, and the window size w. We show the optimal values for all parameters.

Tatikonda and Parthasarathy [34] introduce embedded pivots for computing the distance between unordered trees. An embedded pivot consists of two nodes and their least common ancestor, unless the least common ancestor is one of the two nodes. The distance between the trees is the normalized intersection between their sets of embedded pivots. The size of the pivot set is quadratic in the number of tree nodes, which limits the scalability of the method. We show in our experiments that windowed pq-grams are faster and outperform embedded pivots in terms of matching quality in approximate joins on real world data.

Tekli et al. [35] review research in XML similarity and partition the research into three categories: *edit-based*, *information retrieval-based*, and *other*. Edit-based approaches utilize dynamic programming and have a high complexity [43], while the information-retrieval approaches target ranked XML querying [1]. The "other" category, in which windowed pq-grams fall, includes tag and edge similarity techniques. Tag similarity [8] discards the structure of XML and evaluates the similarity of the tag names. Edge similarity [25] uses a weight function and computes a minimal weight, maximal matching between the edges of two XML trees in $O(n^3)$ time. Windowed pq-grams, which are (overlapping) subtrees, consider both tags and edges, and the distance is computed in $O(n \log n)$ time.

In change detection scenarios two versions of the same document are given and the difference is computed. The distance measures proposed for change detection are evaluated between pairs of documents. When used as a join predicate, there is no obvious way to avoid an expensive nested loop join. Most research in this area relies on a predefined document order [10, 12, 26]. Cobéna et al. [12] take advantage of existing element IDs, which cannot be assumed for joins of data from different sources. Chawathe et al. [9] present a heuristics for the unordered tree edit distance that runs in $O(n^3)$ time and for many cases in $O(n^2)$. The X-Diff algorithm by Wang et al. [38] allows leaf and subtree insertion and deletion, and node renaming. To achieve $O(n^2 \times f_{max} \log f_{max})$ runtime (f_{max} is the maximum fanout of the nodes) they match only nodes with the same path to the root node.

Weis and Naumann [39] propose an XML similarity measure for a duplicate detection framework. In the worst case, all pairs of elements must be compared. Puhlmann et al. [28] improve the efficiency by applying the Sorted Neighborhood method to nested objects. Both approaches assume a known, common schema of the matched documents and require a configuration step. No join algorithm using the proposed similarity measure is presented.

Sanz et al. [31] develop a similarity-based inverted index to identify regions of XML documents that are similar to a given pattern. Adjacent regions are merged into new regions if the new region better matches the pattern than each of the merged regions. The merging algorithm relies on the sibling order. Joins are not addressed.

A core operation in XML query processing is to find all occurrences of a twig pattern [7,22], which, in common with approximate join techniques, concerns identifying patterns in a tree. Our aim is different. We split the tree into subtrees to calculate the distance between trees, not to answer queries.

Several papers deal with the related, but different problem of detecting the structural similarity between XML documents [13,16,20,27]. Two documents are considered struc-

turally similar if they are valid for a similar DTD. The text content of the elements and attribute values are ignored.

3 Motivation

When integrating data, different representations of the same real world object need to be matched. For instance, when companies merge, their customer data will need to be integrated, but the companies may have different ways to represent customer data. As another example, an internet shop may want to enrich its product description with data provided by third parties, each of which have slightly different descriptions for the same product. A third scenario is integrating citations extracted from the reference section of text documents (e.g., merging DBLP and Citeseer data).

In our application scenario we consider building an online database about music CDs that integrates data from two sources: a song lyric store and CD warehouse.[1] The integrated database will store the artists and songs of an album, information about individual songs such as the lyrics, guitar tabs, and information about the artists.

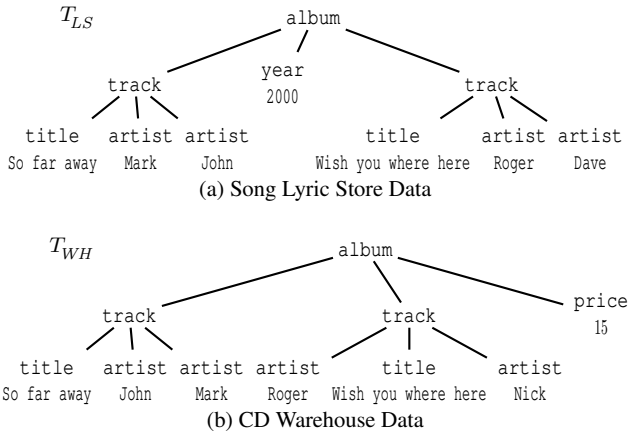
Example 1 Figure 1 shows tree representations of two different XML documents. Both represent data about the same song album. Yet exact, ordered tree matching would not consider the items as the same for a number of reasons. The song lyric store has an element `year` that is absent from the CD warehouse. The CD warehouse has a price for the album. For one track the databases list different artists. Also the document order of elements differs, i.e., the two documents have different sibling orders.

One way to match items from the two sources is to *join* the documents. The join attribute is (the part of) the XML document that represents the album. Two albums match if they are similar. The join condition cannot be equality since the data items that represent the same album in the different databases may not match exactly. The following XQuery expression returns all `album` pairs that are within distance `$tau`.

```
for $a in doc("lyricstore.xml")//album,
    $b in doc("warehouse.xml")//album
where dist($a,$b) <= $tau
return <match>{$a}{$b}</match>
```

In this XQuery expression, `$a` and `$b` are bound to elements of the sets `doc("lyricstore.xml")//album` and `doc("warehouse.xml")//album`, respectively.

[1] We do not assume that the sources use a common schema, but we assume a common vocabulary to describe the data; the problem of integrating data vocabularies or ontologies is separate from matching the data. Terms in one source can be converted to the vocabulary of the second source prior to matching. We focus on the data matching problem.



(a) Song Lyric Store Data

(b) CD Warehouse Data

Fig. 1 Two XML Trees Representing the Same Album.

Each `album` element is a (small) XML document itself. The distance function, `dist`, is a user-defined function that returns the distance between a pair of XML documents.

More generally, the approximate XML join between two sets of XML documents is defined as follows [19].

Definition 1 (Approximate XML Join) Given two sets of XML documents, F_1 and F_2, a distance measure, $\mathrm{D}(T_i, T_j)$, between documents $T_i \in F_1$ and $T_j \in F_2$, and a threshold τ. The approximate XML join computes all pairs $(T_i, T_j) \in F_1 \times F_2$, such that $\mathrm{D}(T_i, T_j) \leq \tau$.

Our goal is to find a distance function for labeled trees that ignores the sibling order, is effective for data-centric XML, and can be computed efficiently. We use this function as the basis for a scalable approximate join.

4 Ordered and Unordered Tree Matching

We first introduce basic concepts of tree matching and then discuss the impact of the sibling order on the tree distance computation.

XML and Trees: In order to use approximate tree matching techniques for XML, we represent an XML document as a rooted, labeled tree. Each node in the labeled tree is a pair (i, l), where i is the node index and l is the node label. A node in the tree represents an XML element (or attribute). The node index is any number that identifies the node in the document, such as the ordinal position of the element in document order. The node label is a (tag,value)-pair, where *tag* is the name of the element and *value* is the text content of the corresponding element. If the corresponding element contains only sub-elements and no content, then the node value is the empty string, ϵ. An edge connects an element node with each of its subelements (or attributes).

The *label function* $\lambda(n)$ maps a node $n = (i, l)$ to its label l. While nodes are unique within a tree, node labels

are not. To simplify the discussion, we refer to a node by its label and omit node indexes and empty values. $N(T)$ denotes the nodes and $E(T)$ the edges of a tree T. We write $n \in T$ if n is a node of tree T.

Ordered and Permuted Trees: Throughout the paper we assume ordered trees. In an ordered tree the children of a node form a sequence. Ordered trees that differ only in the sibling order are *permutations* of each other. A permutation of a tree T is denoted as $\pi(T)$, the set of all permutations of T as $\Pi(T)$. In a *sorted* tree the siblings are lexicographically ordered by their node labels.

Example 2 Figure 2 shows an ordered tree T together with two different permutations, $\pi_1(T)$ and $\pi_2(T)$. Permutation $\pi_2(T)$ is a sorted tree.

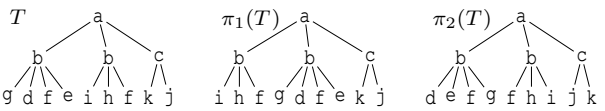


Fig. 2 Permuted Trees.

The Ordered Tree Edit Distance: The ordered edit distance (TED) between two trees is the minimum cost of a sequence of edit operations that transforms one tree into another, using the three standard edit operations: *delete* a node and connect its children to its parent maintaining the sibling order; *insert* a new node between an existing node, p, and a subsequence of consecutive children of p; and *rename* the label of a node.

The choice of a specific cost model for the edit operations is orthogonal to the research in this paper; in our examples we assume the unit cost model, which defines the cost of each edit operation to be 1.

Example 3 Figure 3 illustrates the tree edit operations. T_1 is transformed into T_2 by inserting node `x` as the second child of node `a`, substituting 3 children starting with position 2; the substituted children become children of the new node `x`. Deleting node `x` transforms T_2 back to T_1. Renaming node `a` to `z` transforms T_2 into T_3.

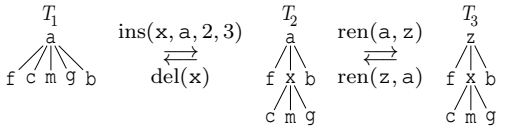


Fig. 3 Tree Edit Operations.

The ordered tree edit distance can be solved in polynomial time [14, 33, 43], and efficient approximations have been presented in the literature [6, 18, 40].

The Unordered Tree Edit Distance: The tree edit distance problem is NP-complete [44] when the sibling order is ignored since after each edit step all tree permutations must be considered. The tree edit distance that ignores the sibling order is the unordered tree edit distance. It is defined as the minimum cost of a sequence of edit operations on ordered trees, where each edit operation is optionally preceded or followed by zero cost permutations.

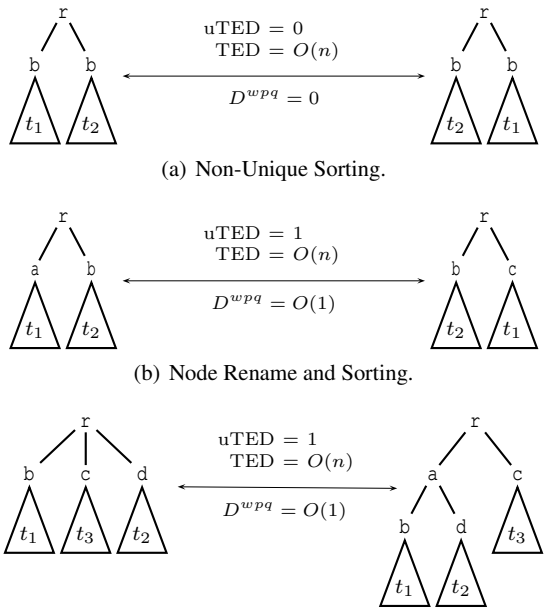
Definition 2 (Unordered Tree Edit Distance) Let T_x and T_y be ordered trees, $s = (T_1 = T_x, T_2, \ldots, T_k = T_y)$, $k \geq 2$, a sequence of ordered trees, $\pi_s \in \Pi(T_1) \times \Pi(T_2) \times \ldots \times \Pi(T_k)$ a permutation of the trees in s, and $\pi_{s,i}(T_i)$ the permutation of tree T_i in π_s. The *unordered tree edit distance* (uTED) between T_x and T_y is defined as

$$\mathrm{uTED}(T_x, T_y) = \min_{s,\pi_s} \sum_{i=1}^{|s|-1} \mathrm{TED}(\pi_{s,i}(T_i), T_{i+1}).$$

Finding the permutations of two ordered trees that yield the smallest tree edit distance is non-trivial, and it is obviously not feasible to compute the edit distance between all permutations of two ordered trees.

As an alternative, consider an approach that sorts the siblings of both trees by their string labels. This heuristic fails for the ordered tree edit distance. When the siblings of a tree are sorted, then the subtrees rooted in the siblings are also sorted. Therefore two subtrees that should match may appear in a different order in the two sorted trees, for example, because the root nodes of the subtrees have different labels or there are siblings with the same labels but different sorting. The ordered tree edit distance restores the subtree order and moves back subtrees node by node. A subtree can be of size $O(n)$, where n is the number of tree nodes. Thus, even if the unordered tree edit distance is zero or a small constant (for example, a single renamed node), the tree edit distance between the respective sorted trees may be $O(n)$.

Example 4 Consider the trees in Figure 4(a). The two children of the root node have the same label, and the label sort is not unique. Although both trees are sorted, the subtrees t_1 and t_2 are swapped in the two trees. The unordered tree edit distance is zero since they differ only in the sibling order. The ordered tree edit distance is roughly the tree size since the subtrees t_1 and t_2 must be moved back node by node. In Figure 4(b) the unordered tree edit distance is 1 (renaming the root node of t_1 from `a` to `c`), but the ordered tree edit distance is $O(n)$ since the renaming changes the subtree order. In Figure 4(c) the ordered tree edit distance cannot insert node `a` with children `b` and `d` into the left tree since `b` and `d` are not consecutive siblings; the subtree order must be restored before the insert can take place. The unordered tree edit distance is 1 since `a` is inserted in a permutation of the left tree in which `b` and `d` are consecutive siblings.



(a) Non-Unique Sorting.

(b) Node Rename and Sorting.

(c) Node Insertion and Sorting.

Fig. 4 Tree Sorting Changes the Subtree Order.

In this paper we show that sorting trees is a valid approach for windowed pq-grams, which will be introduced in Section 6. We show that the permutation of a constant number of siblings changes only a constant number of windowed pq-grams. Thus, we can sort the trees and compute the windowed pq-grams on the sorted trees. The number of windowed pq-grams that are in one tree decomposition only gives a distance measure, termed windowed pq-grams distance (D^{wpq}), which approximates the unordered tree edit distance (compare uTED and D^{wpq} in Figure 4).

5 Desiderata for Tree Decompositions

Tree decompositions provide an alternative to the edit based distances discussed in the previous section. Tree decompositions split a tree into small pieces, and trees with many common pieces are considered similar. Our solution, windowed pq-grams, is a tree decomposition for unordered trees. The efficient join algorithm for windowed pq-grams (cf. Section 8.2), which reduces the distance join to an equality join on windowed pq-grams, is applicable to tree decompositions in general, but not to edit based distances.

In this section we first provide an overview about different tree decomposition techniques, including pq-grams, and then proceed to formulate three properties that tree decompositions for unordered tree matching must satisfy.

5.1 Overview of Tree Decompositions

A *tree decomposition*, $X(T)$, is a set of subtrees or subtree sequences, also termed a *snippet*, of a possibly preprocessed tree T. The preprocessing typically extends the original tree with dummy nodes, yielding an *extended tree*. A dummy node is a node with a special label ($\star$), which is different from all labels in the original tree. A tree decomposition is used to define and compute the distance between two trees. Two trees are similar if their decompositions have many snippets in common. Different tree decompositions have been proposed in the literature and are described next (see also Fig. 5):

- *Binary Branches* [40]: A binary branch is a snippet that consists of a node, its first child, and its right sibling. The preprocessing adds the following dummy nodes to the original tree T: a parent and a right sibling to the root node, a node after the last child of each node, and a child to each leaf. For each non-dummy node of the extended tree T^{bb} one binary branch is produced.
- *pq-Grams* [4]: A pq-gram is a besom shaped subtree that consists of an anchor node, q consecutive children, and the $p-1$ closest ancestors of the node. The preprocessing adds $p-1$ dummy ancestors to the root, $q-1$ dummy children before and after the children of each node, and q dummy children to each leaf node. For each non-dummy node of the extended tree T^{pq} all possible pq-grams are produced.
- *Path Shingles* [8]: A path shingle is a sequence of node chains. The tree is decomposed into all paths that contain the root node, the paths are ordered by the preorder numbers of their leaf nodes, and k consecutive paths form a path shingle of length k.
- *Valid Subtrees* [17]: The valid subtrees have different shapes and are produced in a sequence of parsing steps. In parsing step i, $i \geq 1$, a new tree T_i is obtained by contracting nodes of tree T_{i-1} into nodes of T_i. The parsing starts with $T_0 = T$ and stops if $|N(T_i)| = 1$. Nodes are contracted as follows: contiguous sequences of leaf children are split into blocks of length two or three; then the blocks are contracted, a leaf node is contracted with its parent if it is the leftmost leaf child that is not adjacent to any other leaf child; and finally, node chains are contracted into a single node. Each node of a tree T_j, $j \geq 0$, is a contracted subtree of the original tree T, and the contracted subtree is a snippet of the decomposition.

Example 5 Figure 5 shows the different decompositions of the example tree T. The binary branch decomposition produces six snippets, each consisting of two small subtrees with two and one nodes, respectively. The ordered pq-gram decomposition splits the tree into 13 subtrees of the same shape. The path shingle decomposition produces five snippets, each consisting of two subtrees that are node chains.

The valid subtree decomposition produces 13 valid subtrees in total. The left column in Figure 5(e) shows the trees T_i that are produced in each parsing step by contracting nodes from T_{i-1}. Each node of a parse tree T_i represents a subtree in the original tree T. In our illustration, this subtree is encoded in the respective node labels. The right column shows the valid subtrees that are produced in each step. The valid subtrees of T_0 are the nodes of the original tree. The two contracted leaf nodes in Step 1 are connected with a dummy parent (i.e., a node that does not exist in the original tree) to form a tree. T_3 consists of a single node with a valid subtree that is equal to T.

The decomposition of a tree into snippets requires particular attention when the sibling order is ignored, as we will discuss and illustrate below. To facilitate this discussion, we first introduce some notation.

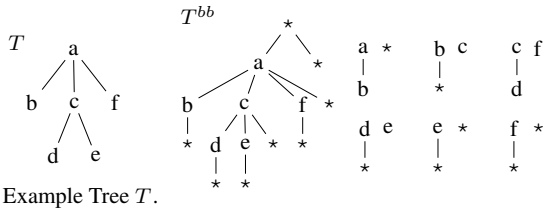
Let $C(n,T)$ denote the *children of a node* $n \in T$, and $C_\lambda(n,T) = \{\lambda(x) : x \in C(n,T)\}$ the corresponding bag of labels. We define a wildcard label "$\cdot$" that matches all other labels. The *conditional labeling function*, $\lambda(n,C)$, returns the wildcard label if node n is not in a set of nodes C, i.e., $\lambda(n,C) = \lambda(n)$ if $n \in C$ and $\lambda(n,C) =$ "$\cdot$" if $n \notin C$.

The *snippets of node* $n \in T$ are the snippets that contain at least two child nodes of n in the extended tree T^{ex}, i.e., $S(n,T) = \{s \in X(T) : |C(n,T^{ex}) \cap N(s)| \geq 2\}$. Note that the snippets of a node are affected by the sibling order.

Example 6 Consider the example tree T in Figure 5(a). The snippets of the root node, a, are: the second and the third binary branch in the first row and the last binary branch in the second row in Figure 5(b); all pq-grams in the first row in Figure 5(c); the second and the last path shingle in Figure 5(d); and the first valid subtree of Steps 2 and 3 in Figure 5(e).

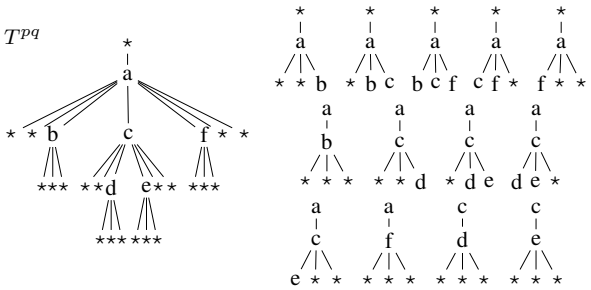
We use a linear encoding and represent a snippet as a tuple $s = (n_1, n_2, \ldots, n_{|s|})$ of its nodes in preorder. With $\lambda(s) = (\lambda(n_1), \lambda(n_2), \ldots, \lambda(n_{|s|}))$ we denote the node labels of a snippet, called its *label tuple*. While a snippet is unique within a tree, different snippets may yield identical label tuples. To simplify the notation, we represent a node by its label and a snippet by the concatenation of its node labels, e.g., the node $(1, \mathtt{a})$ is denoted as $\mathtt{a}$, the snippet $((1,\mathtt{a}),(2,\mathtt{b}),(6,\mathtt{f}))$ as $\mathtt{abf}$.

A *conditional label tuple*, $\lambda(s,C)$, is the tuple of conditional labels for snippet s. $S_\lambda(n,T) = \{\lambda(s, C(n,T^{ex})) : s \in S(n,T)\}$ is the bag of all conditional label tuples of node $n \in T$. Notice that the conditional label tuples in $S_\lambda(n,T)$ convey only information about the horizontal structure (sibling relations), which are relevant for our discussion about order. The labels of the nodes that represent the vertical structure (parent-child relationship) are substituted by wildcard labels. Thus, conditional label tuples differ only if the labels of the siblings differ.

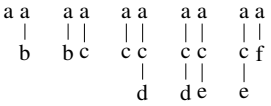


(a) Example Tree T.

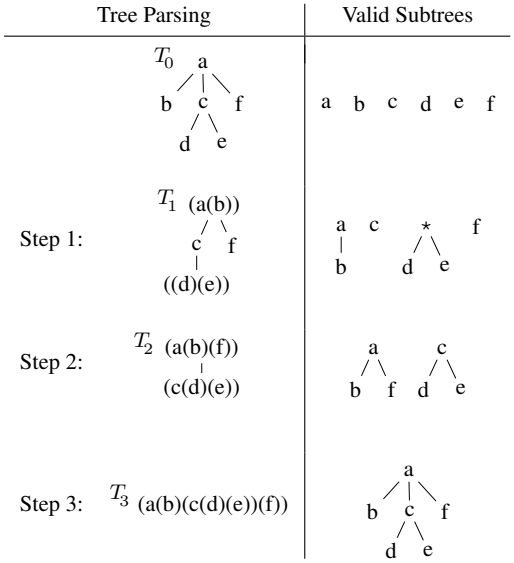
(b) Extended Tree and Binary Branches.



(c) Extended Tree and Ordered pq-Grams ($p = 2$, $q = 3$).



(d) Path Shingles ($k = 2$).



(e) Tree Parsing and Valid Subtrees.

Fig. 5 Tree Decompositions: Binary Branches, pq-Grams, Path Shingles, and Valid Subtrees.

Example 7 Consider Figure 5. The conditional label tuples of node a in tree T for respectively, binary branches (bb), pq-grams (pq), path shingles (ps), and valid subtrees (vs), are:

- $S^{bb}_\lambda(a, T^{bb}) = \{\mathtt{b{\cdot}c}, \mathtt{c{\cdot}f}, \mathtt{f{\cdot}\star}\}$
- $S^{pq}_\lambda(a, T^{pq}) = \{\mathtt{{\cdot}{\cdot}\star\star b}, \mathtt{{\cdot}{\cdot}\star bc}, \mathtt{{\cdot}{\cdot}bcf}, \mathtt{{\cdot}{\cdot}cf\star}, \mathtt{{\cdot}{\cdot}f\star\star}\}$
- $S^{ps}_\lambda(a, T) = \{\mathtt{{\cdot}b{\cdot}c}, \mathtt{{\cdot}c{\cdot}{\cdot}f}\}$
- $S^{vs}_\lambda(a, T) = \{\mathtt{{\cdot}bf}, \mathtt{{\cdot}bc{\cdot}{\cdot}f}\}$

5.2 Desired Properties for Unordered Tree Decompositions

Based on our discussion about unordered tree matching in Section 4, we identify three core properties that tree decompositions should satisfy when sibling order is ignored:

- *Preservation of Structure Information*: Each edge of a tree appears in the same number of snippets in the decomposition of the tree.
- *Preservation of Children Information*: The overlap of the snippets of two nodes from different trees should be the same as the overlap of their children.
- *Preservation of Sibling Information*: If the children of two parent nodes u and v are partitioned differently, i.e., the parent of some children is changed, the combined snippets of u and v are different.

In the sequel these properties are formally defined and error measures are introduced. We give examples that illustrate the effect of the decomposition properties in Section 7.

5.2.1 Preservation of Structure Information

The structure of a tree is given by its edges. The goal is to decompose a tree in such a way that all edges have the same impact on the tree distance. When an edge appears in one tree only, none of the snippets that contain that edge matches a snippet in the other tree, and the distance between the trees increases. Thus, the importance of an edge depends on the number of snippets that contain the edge. The structure information is preserved if each edge of the original tree appears in the same number of snippets.

Definition 3 (Preservation of Structure Information) Given a tree T with edges $E(T)$. The decomposition $X(T)$ preserves the structure information iff $\exists k \forall e : e \in E(T) \Rightarrow |\{s : s \in X(T) \wedge e \in E(s)\}| = k$.

To measure the preservation of structure information we define the structure error. Let the *edge frequency* be the number of snippets in $X(T)$ that contain edge e, i.e., $\phi(e,T) = |\{s : s \in X(T) \wedge e \in E(s)\}|$; the average edge frequency of a set E of edges is $\bar{\phi}(E,T) = \sum_{e \in E} \phi(e,T)/|E|$. Then the structure error is defined as follows.

Definition 4 (Structure Error) The *structure error*, ϕ_ϵ, of an edge $e \in E(T)$ is defined as the relative error of its edge frequency with respect to the average edge frequency in T,

$$\phi_\epsilon(e,T) = |1 - \phi(e,T)/\bar{\phi}(E(T),T)|.$$

The structure error is the deviation of the frequency of a particular edge from the average edge frequency. If $\phi_\epsilon = 0$ for all edges of a tree, the structure information is preserved.

5.2.2 Preservation of Children Information

The goal is to decompose two trees, T_1 and T_2, in such a way that the relative overlap between the snippets of two nodes, $n_1 \in T_1$ and $n_2 \in T_2$, is the same as the relative overlap between their children.

Definition 5 (Preservation of Children Information) Two decompositions $X(T_1)$, $X(T_2)$ preserve the children information iff for all pairs of non-leaf nodes $n_1 \in T_1$, $n_2 \in N(T_2)$ it holds that $S_\lambda(n_1,T_1) \uplus S_\lambda(n_2,T_2) \neq \emptyset$ and

$$\frac{|C_\lambda(n_1,T_1) \uplus C_\lambda(n_2,T_2)|}{|C_\lambda(n_1,T_1) \uplus C_\lambda(n_2,T_2)|} = \frac{|S_\lambda(n_1,T_1) \uplus S_\lambda(n_2,T_2)|}{|S_\lambda(n_1,T_1) \uplus S_\lambda(n_2,T_2)|}.$$

To measure the preservation of the children information we introduce the children error, which is based on the Jaccard similarity [37]. Assume bags X and Y such that $X \neq \emptyset$ or $Y \neq \emptyset$. The Jaccard similarity between X and Y is

$$J(X,Y) = 2|X \uplus Y|/|X \uplus Y|. \quad (1)$$

Definition 6 (Children Error) Let $n_1 \in T_1$ and $n_2 \in T_2$ be two non-leaf nodes. The *children error*, ε, is defined as

$$\varepsilon(n_1,T_1,n_2,T_2) = |J(C_\lambda(n_1,T_1),C_\lambda(n_2,T_2)) - J(S_\lambda(n_1,T_1),S_\lambda(n_2,T_2))| \quad (2)$$

if $S_\lambda(n_1,T_1) \uplus S_\lambda(n_2,T_2) \neq \emptyset$, and $\varepsilon = 1$ otherwise.

The children error ε ranges between 0 and 1. If $\varepsilon = 0$ for all pairs of non-leaf nodes, the children information is fully preserved.

5.2.3 Preservation of Sibling Information

The goal is to decompose a tree such that the combined snippets of two nodes u and v are different if a child of u is moved to become a child of v.

Definition 7 (Preservation of Sibling Information) The decomposition $X(T_1)$ of a tree T_1 preserves the sibling information iff for the decomposition $X(T_2)$ of any tree T_2 that results from T_1 by moving a child n from its parent u to a new parent v the following holds:

$$S_\lambda(u,T_1) \uplus S_\lambda(v,T_1) \neq S_\lambda(u,T_2) \uplus S_\lambda(v,T_2).$$

The sibling information is relevant to detect changes that are not visible from the parent-child relationships between nodes. For example, if a node is moved to another parent with the same label, then the edges in the old and in the new tree have identical labels. Snippets that encode only parent-child relationships (e.g., snippets that consist of a single edge) cannot detect such a node move. If the moved node has siblings with different labels before and after the move, the node move can be detected using sibling information encoded in the snippets of the parent node. For instance, let

n be a child that is moved from parent u to parent v, i.e., $C(u,T_2) = C(u,T_1) \setminus \{n\}$, and $C(v,T_2) = C(v,T_1) \cup \{n\}$. Then the node move is detected if at least one snippet changes, i.e., the union of the snippets of $u \in T_1$ and snippets of $v \in T_1$ is different from the union of snippets of $u \in T_2$ and snippets of $v \in T_2$

Intuitively, the sibling information is measured as the number of different sibling pairs that appear in the snippets. The set of all (unordered) sibling pairs encoded by the snippets of a node n is $\text{encpairs}(n,T) = \{\{a,b\} : \exists_{s \in S(n,T)}(a,b \in C(n,T^{ex}) \cap N(s)), a \neq b\}$. The set of all sibling pairs that can be formed for the children of a node n is $\text{allpairs}(n,T) = \{\{a,b\} : a,b \in C(n,T), a \neq b\}$.

Not all sibling pairs are relevant for the sibling information. The same pair may be encoded twice, for example, in two different snippets and with different order. Snippets formed from the same sibling pair are duplicates and store redundant information. Sibling pairs with a dummy node provide no sibling information. The set of relevant sibling pairs is $\text{encpairs}(n,T) \cap \text{allpairs}(n,T)$.

We define the snippet recall and snippet precision to measure the preservation of sibling information.

Definition 8 (Snippet Recall) Consider a node $n \in T$ with $f = |C(n,T)| \geq 2$ children. The *snippet recall*, ρ, is defined as the ratio of relevant sibling pairs encoded by the snippets of n to the number of all possible pairs, i.e.,

$$\rho(n,T) = \frac{|\,\text{encpairs}(n,T) \cap \text{allpairs}(n,T)|}{|\,\text{allpairs}(n,T)|}. \quad (3)$$

With $f \geq 2$ children we can form $|\,\text{allpairs}(n,T)| = \binom{f}{2} = \frac{f(f-1)}{2}$ pairs. $\rho = 1$ if all possible pairs of children of n are in the snippets of n, $\rho = 0$ if none of the possible pairs is encoded. Snippets with low recall may not encode relevant sibling pairs and thus miss node moves.

Definition 9 (Snippet Precision) Consider a node $n \in T$. The *snippet precision*, π, is defined as the ratio of the relevant sibling pairs to the sibling pairs encoded by the snippets of n, i.e.,

$$\pi(n,T) = \frac{|\,\text{encpairs}(n,T) \cap \text{allpairs}(n,T)|}{|\,\text{encpairs}(n,T)|}. \quad (4)$$

$\pi = 1$ if the snippets contain no dummy nodes. A low precision, i.e., many snippets with dummy nodes, decreases the weight of the original nodes.

6 Windowed *pq*-Grams

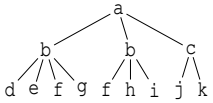
pq-Grams were introduced as an approximation of the ordered tree edit distance [4]. A *pq*-gram is a small subtree of a specific shape composed of two parts: a stem that consists of an anchor node with $p-1$ ancestors and a base that consists of q consecutive children of the anchor node. Stems are node chains of length p and invariant to order. The strategy for choosing stems in ordered tree matching carries over to unordered tree matching. The bases of *pq*-grams are formed by consecutive siblings and are sensitive to order. Hence, a different strategy is required, when the sibling order shall be ignored.

In this section we first introduce *windowed pq-grams,* which are independent of the sibling order and therefore well-suited for unordered tree matching. Then we define the windowed *pq*-gram distance and show that it is a pseudo-metric.

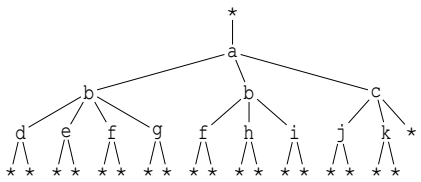
6.1 Constructing Windowed *pq*-Grams

The construction of windowed *pq*-grams is a 3-step process as illustrated in Figure 6:

a) *sort* the tree,
b) *extend* the sorted tree, and
c) *decompose* the extended tree into windowed *pq*-grams.



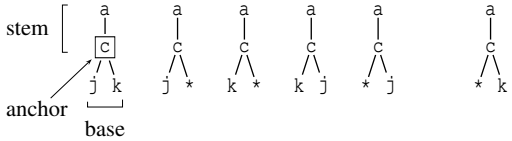
(a) Sorted Tree T^{sort}.



(b) Extended Tree T^{wpq}.

W(c,T) = { jk⋆ jk⋆, j k⋆j k⋆, jk ⋆jk ⋆ }

(c) Window for Anchor Node c.



(d) Windowed *pq*-Grams for Anchor Node c.

Fig. 6 Construction of Windowed *pq*-Grams ($w=3$, $p=q=2$).

6.1.1 Sorting the Tree

In the first step we sort the tree by ordering the siblings lexicographically by their node labels.

Definition 10 (Sorted Tree) A tree T is *sorted* iff its siblings are ordered and for each sibling pair, $n = (i, l)$ and $n' = (i', l')$, the order satisfies $l < l' \Rightarrow n < n'$.

In XML trees the labels are (tag,value)-pairs, and for two labels $l = (t, v)$ and $l' = (t', v')$ we define $l < l' \Leftrightarrow t < t' \vee (t = t' \wedge v < v')$. Because of nodes with identical labels the sorting is not unique. However, all possible sorts of a tree yield identical windowed pq-grams and are equivalent for our purpose. Figure 6(a) shows T_{sort}, the sorted permutation of example tree T of Figure 2.

6.1.2 Extending the Sorted Tree

The second step extends the sorted tree with *dummy nodes*.

Definition 11 (Extended Tree) Let T be a sorted tree, $p > 0$ and $q > 0$ be the parameters determining the shape of the windowed pq-grams, $w \geq q$ be the window size, and f be the fanout of a node. The *extended tree*, T^{wpq}, is defined as T extended with dummy nodes ($\bullet_i$) as follows:

- *root:* $p-1$ ancestors are prepended to the root node;
- *leaves:* q children are added to each leaf node;
- *siblings:* $w-f$ siblings are appended to each child sequence $(c_1, \ldots, c_f)$ of size $0<f<w$, yielding $(c_1, \ldots, c_f, \bullet_1, \ldots, \bullet_{w-f})$.

Dummy nodes have a special label, $\lambda(\bullet_i) = \star$, which is the same for all dummy nodes. The number of dummy nodes depends on the size of the window that we use to generate windowed pq-grams. Figure 6(b) shows the extended tree T^{wpq} for $w=3$ and $p=q=2$.

6.1.3 Decomposing into Windowed pq-Grams

The third step is to define a window on a tree and the windowed pq-grams based of the extended tree. With $K(n, T)$ we denote the sequence of all children of node $n \in T$. Given two sequences a and b, $c = a \circ b$ denotes the concatenation of a and b, and $a \subseteq b$ denotes a being a (possibly non-consecutive) subsequence of b.

Definition 12 (Window) Given a tree T with extended tree T^{wpq}, a window size w, a node $n \in T$ that is a non-leaf in T, and the children sequence $K(n, T^{wpq})$ in the extended tree, a *window W over the children of node $n \in T$* is a consecutive subsequence of $K(n, T^{wpq}) \circ K(n, T^{wpq})$ of length w. $W(n, T)$ denotes the set of all windows over the children of n. If n is a leaf in T, no window is defined for n and $W(n, T) = \emptyset$.

Intuitively, windows are produced by shifting a pattern of length w over the children of a node. The nodes covered by the pattern form a window. Patterns that overhang on the right are wrapped (concatenation of the children sequence).

Definition 13 (Windowed pq-Grams) Let T be a tree with extended tree T^{wpq}. An ordered tree G is a windowed pq-gram of T iff

(a) G is a permuted subtree of T^{wpq} with q leaf and p non-leaf nodes,
(b) all leaf nodes of G are children of a single node with fanout q, called the *anchor node* a,
(c) $K(a, G) = K(a, T^{wpq})$ or $\exists W \in W(a, T)$ such that $K(a, G) \subseteq W$.

Conditions (a) and (b) in Definition 13 define the shape of the pq-grams, Condition (c) defines the nodes that appear in the bases. The base of a leaf anchor node consists of dummy nodes (left term in the disjunction), all other bases are (possibly non-consecutive) subsequences of a window.

Example 8 Figure 6(c) illustrates the construction of the window for node c. The concatenated children sequences is $K(\mathtt{c}, T) = \mathtt{jk}\star\mathtt{jk}\star$, which gives the windows $W(\mathtt{c}, T) = \{\mathtt{jk}\star, \mathtt{k}\star\mathtt{j}, \star\mathtt{jk}\}$. Figure 6(d) shows all windowed pq-grams for $p = q = 2$ that can be formed for T^{wpq} and anchor node c. The respective conditional label tuples are $S^{wpq}_{\lambda}(\mathtt{c}, T) = \{\cdot\cdot\mathtt{jk}, \cdot\cdot\mathtt{j}\star, \cdot\cdot\mathtt{k}\star, \cdot\cdot\mathtt{kj}, \cdot\cdot\star\mathtt{j}, \cdot\cdot\star\mathtt{k}\}$.

Dummy nodes, windows, and the concatenation of children sequences guarantee that each node of a tree is in the same number of bases produced from one tree, thus giving each node the same weight. Dummy nodes prevent a node from appearing twice in the same window when the two children sequences are concatenated. The concatenation guarantees that each node appears in all w positions of a window exactly once, independent of the number of left and right siblings. Only bases within windows are formed, thus each node is in the same number of bases.

6.2 Windowed pq-Grams Profile, Index, and Distance

Based on windowed pq-grams, we proceed to define an index for and a distance between unordered trees.

Definition 14 (Windowed pq-Gram Profile and Index) Let T be a tree, $p > 0$, $w \geq q > 0$. The *windowed pq-gram profile* of T, $X^{wpq}(T)$, is the set of all windowed pq-grams of T. The *windowed pq-gram index* of T, $X^{wpq}_{\lambda}(T)$, is the bag of all label tuples of windowed pq-grams of T, i.e.,

$$X^{wpq}_{\lambda}(T) = \biguplus_{G \in X^{wpq}(T)} \lambda(G).$$

The following theorem shows that the size of the pq-gram profile is bound by $O(n)$ for a tree with n nodes, which makes pq-grams scalable for large trees (both in terms of memory requirements of the pq-gram index and in terms of computation of the pq-gram distance).

Theorem 1 (Linear Profile Size) *Let T be a tree with n nodes and let the base size $q > 1$ and the window size $w \geq q$ be constants, then the size of the windowed pq-gram profile of T is linear in the tree size, $|X^{wpq}(T)| \leq nq\binom{w}{q}$. If all non-leaf nodes of T have fanout $f \geq w$, then $|X^{wpq}(T)| = (n-1)\binom{w-1}{q-1} + l$, where l is the number of leaves.*

Proof We first show $|X^{wpq}(T)| \leq nq\binom{w}{q}$: Consider an insert operation that transforms tree T_i into tree T_{i+1} by inserting a new leaf node, x, as a child of a node p (fanout f_p in T_i). The leaf insertion transforms the windowed pq-grams of T_i with anchor node p, $G_p(T_i) \subseteq X^{wpq}(T_i)$, to the windowed pq-grams $G_p(T_{i+1}) \subseteq X^{wpq}(T_{i+1})$, and new windowed pq-grams with anchor node x, $G_x(T_{i+1}) \subseteq X^{wpq}(T_{i+1})$, are introduced. The profile increases by

$$|X^{wpq}(T_{i+1})| - |X^{wpq}(T_i)| = |G_p(T_{i+1})| + |G_x(T_{i+1})| - |G_p(T_i)|.$$

We distinguish the three cases illustrated in Figure 7 and show that inserting a new leaf, x, as a child of a *leaf* node of T_i (Case 1 in Figure 7) adds the largest number of windowed pq-grams to the profile, i.e., $w\binom{w-1}{q-1} \geq \binom{w-1}{q-1} + 1 \geq 1$. To show $w\binom{w-1}{q-1} \geq \binom{w-1}{q-1} + 1$ we divide by $\binom{w-1}{q-1} > 0$ and get $w \geq 1 + \frac{(q-1)!(w-q)!}{(w-1)!}$. Since $w \geq 2$ we need to show $\frac{(q-1)!(w-q)!}{(w-1)!} \leq 1$. We expand the factorials, substitute $a_i = w - q + i$, and use $a_i \geq i$:

$$\begin{aligned}\frac{(q-1)!(w-q)!}{(w-1)!} &= \frac{1 \cdot 2 \cdot \ldots \cdot (q-1)}{(w-q+1)(w-q+2)\ldots(w-1)} \\ &= \frac{1 \cdot 2 \cdot \ldots \cdot (q-1)}{a_1 a_2 \ldots a_{q-1}} \leq \frac{1 \cdot 2 \cdot \ldots \cdot (q-1)}{1 \cdot 2 \cdot \ldots \cdot (q-1)} = 1\end{aligned}$$

Any tree T of size n can be constructed from a tree T_0 that consists only of a root node by inserting $n-1$ leaves. Thus, the profile size of $T = T_n$ is at most

$$\begin{aligned}|X^{wpq}(T)| &= |X^{wpq}(T_0)| + \sum_{i=1}^{n-1}[|X^{wpq}(T_{i+1})| - |X^{wpq}(T_i)|] \\ &\leq 1 + (n-1)w\binom{w-1}{q-1} \leq nw\binom{w-1}{q-1} = nq\binom{w}{q}.\end{aligned}$$

Next we show $|X^{wpq}(T)| = (n-1)\binom{w-1}{q-1} + l$ for $f \geq w$: For trees with fanout $f \geq w$, each non-root node is the first node of exactly $\binom{w-1}{q-1}$ bases, all l leaf nodes are the anchor node of exactly one base, these two sets have no overlap, and there are no other pq-grams. □

The windowed pq-gram-distance between two trees is computed from the number of windowed pq-grams that the indexes of the compared trees have in common.

Definition 15 (Windowed pq-Gram Distance) Let T_1 and T_2 be two trees with index $X^{wpq}_\lambda(T_1)$ and $X^{wpq}_\lambda(T_2)$, respectively. The *windowed pq-gram distance*, $\mathrm{D}^{wpq}(T_1, T_2)$, between T_1 and T_2 is defined as

$$\begin{aligned}\mathrm{D}^{wpq}(T_1, T_2) = |X^{wpq}_\lambda(T_1) \uplus X^{wpq}_\lambda(T_2)| - \\ 2|X^{wpq}_\lambda(T_1) \Cap X^{wpq}_\lambda(T_2)|.\end{aligned} \quad (5)$$

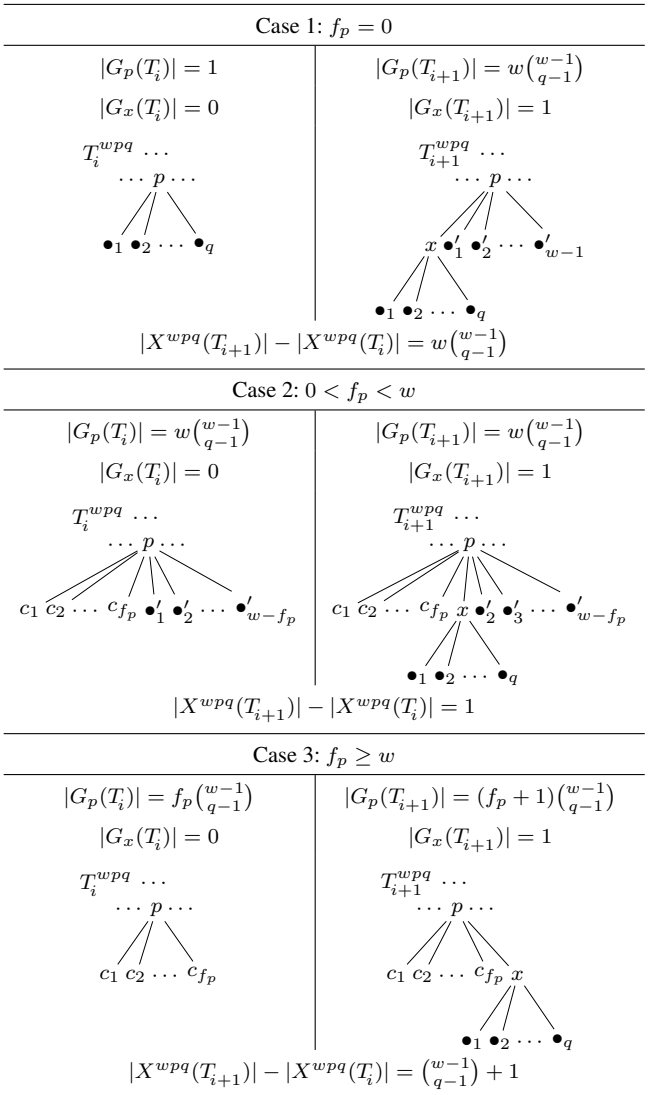


Fig. 7 Profile Increase after Inserting a Leaf (Proof of Theorem 1).

Next, we show that the windowed pq-gram-distance is a pseudo-metric. A pseudo-metric is essential for many similarity search algorithms since it can be used to efficiently prune the search space [42]. Different from a metric, in a pseudo-metric non-identical trees may be at distance zero.

Theorem 2 (Pseudo-Metric) *The windowed pq-gram distance, D^{wpq}, is a pseudo-metric, i.e., for any trees T_1, T_2, and T_3, the following holds:*

1. non-negativity*:* $\mathrm{D}^{wpq}(T_1, T_2) \geq 0$
2. reflexivity*:* $T_1 = T_2 \Rightarrow \mathrm{D}^{wpq}(T_1, T_2) = 0$
3. symmetry*:* $\mathrm{D}^{wpq}(T_1, T_2) = \mathrm{D}^{wpq}(T_2, T_1)$
4. triangle inequality*:* $\mathrm{D}^{wpq}(T_1, T_3) \leq \mathrm{D}^{wpq}(T_1, T_2) + \mathrm{D}^{wpq}(T_2, T_3)$

Proof The *non-negativity* of the windowed pq-gram distance follows from $|X^{wpq}_\lambda(T_1) \uplus X^{wpq}_\lambda(T_2)| \geq 2|X^{wpq}_\lambda(T_1) \Cap X^{wpq}_\lambda(T_2)|$, the *reflexivity* follows from $T_1 = T_2 \Rightarrow$

$X^{wpq}_\lambda(T_1) = X^{wpq}_\lambda(T_2)$, the *symmetry* follows from the symmetry of bag union and bag intersection. *Triangle inequality*: We follow a proof idea by Yianilos [41] and partition the indexes into disjoint subsets as shown in Figure 8.

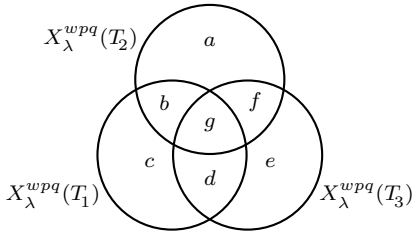


Fig. 8 Disjoint Subsets of $X^{wpq}_\lambda(T_1)$, $X^{wpq}_\lambda(T_2)$, and $X^{wpq}_\lambda(T_3)$.

The lowercase letters in the figure are the cardinalities of the respective subsets. Thus, $\mathrm{D}^{wpq}(T_1,T_2) = c+d+a+f$, $\mathrm{D}^{wpq}(T_2,T_3) = a+b+d+e$, $\mathrm{D}^{wpq}(T_1,T_3) = b+c+e+f$. The triangle inequality holds:

$$\begin{aligned} \mathrm{D}^{wpq}(T_1,T_3) &\leq \mathrm{D}^{wpq}(T_1,T_2) + \mathrm{D}^{wpq}(T_2,T_3) \\ b+c+e+f &\leq c+d+a+f+a+b+d+e \\ 0 &\leq 2a+2d \end{aligned}$$ □

To account for the size of the trees, the windowed pq-gram distance can be normalized by dividing the right-hand side of Equation 5 by $|X^{wpq}_\lambda(T_1) \uplus X^{wpq}_\lambda(T_2)| - |X^{wpq}_\lambda(T_1) \upharpoonright X^{wpq}_\lambda(T_2)|$. This normalization was shown to preserve the metric properties [6].

We conclude with the proof that tree sorting is a valid approach for windowed pq-grams. Sorting a tree changes the order in which its subtrees appear. However, the windowed pq-gram distance is independent of the size of the reordered subtrees. Only the windowed pq-grams that contain the root nodes of the reordered subtrees in the bases change. This core property qualifies windowed pq-grams for unordered tree matching and sets the windowed pq-gram distance apart from other distance measures such as the ordered tree edit distance.

Theorem 3 (Local Effect of Reordering Subtree) *Assume a sorted tree T_1 is transformed into a tree T_2 by changing the order of the $f \geq w$ children of a node n. The reordering affects at most $O(f)$ windowed pq-grams, i.e.,*

$$|X^{wpq}_\lambda(T_1) \setminus X^{wpq}_\lambda(T_2)| \leq O(f).$$

Proof The stems are independent of the sibling order. Only the windowed pq-grams with reordered nodes in the base change: there are $f\binom{w-1}{q-1}$ windowed pq-grams with anchor node n, and the constants w and q are independent of the size of the trees T_1 and T_2. □

7 Properties of Windowed pq-Grams

In this section we show that windowed pq-grams preserve the three properties introduced in Section 5, and we show the optimal choice of the parameters p (stem size), q (base size), and w (window size). Specifically, stems of size $p = 1$ have structure error zero (Lemma 1), and bases of size $q = 2$ have a smaller children error than larger bases (Lemma 2), but they detect the same node moves (Lemma 3). For $q = 2$ we provide snippet recall and precision (Lemma 4), and we provide a window size w that optimizes both recall and precision (Theorem 4).

To illustrate the properties of windowed pq-grams (as introduced in Section 6), we also investigate three alternative and straightforward approaches to construct pq-grams:

- *All-permutation pq-grams:* All possible children permutations of length q form a base.
- *Consecutive children pq-grams:* The children are sorted, and each subsequence of length q of the sorted children forms a base.
- *Single leaf pq-grams:* Each child node is a base of length $q = 1$.

Similar to windowed pq-grams, $q - f$ dummy node children are added to each non-leaf node with fanout $f < q$, and each leaf is the anchor node of a single pq-gram with q dummy nodes in the base. We show that these approaches fail to meet the three requirements for tree decompositions.

7.1 Preservation of Structure Information and Stem Size

To preserve structure information, the structure error should be zero for all edges in the tree, i.e., all edges of the original tree should appear in the same number of windowed pq-grams. For $p = 1$ the frequency of an edge $e = (n, c)$ is equal to the number of bases in which child c appears. Thus, all nodes (except the root node) should appear in the same number of bases. This is not true for all-permutation pq-grams. There are $\frac{f!}{(f-q)!}$ permutations of length q over $f \geq q$ children and each child appears in $\frac{q}{f}\frac{f!}{(f-q)!} = O(f^{q-1})$ bases. Thus, for all-permutation pq-grams the pq-grams produced from children with many siblings disproportionally contribute to the total number of pq-grams. As a result, changes covered by these pq-grams are amplified, other changes are disregarded.

Example 9 Consider the trees in Figure 9. Both T_y and T_z are at the same distance $\mathrm{uTED} = 1$ from T_x (one rename operation), but at different distances for all-permutation pq-grams ($q = 3$):

- $|X^{ap}_\lambda(T_x) \uplus X^{ap}_\lambda(T_y)| = 282$
- $|X^{ap}_\lambda(T_x) \upharpoonright X^{ap}_\lambda(T_y)| = 80$

- $\mathrm{D}^{ap}(T_x, T_y) = 122$
- $|X^{ap}_{\lambda}(T_x) \uplus X^{ap}_{\lambda}(T_z)| = 282$
- $|X^{ap}_{\lambda}(T_x) \uplus X^{ap}_{\lambda}(T_z)| = 134$
- $\mathrm{D}^{ap}(T_x, T_z) = 14$

The reason for the different distances is the number of bases in which the renamed nodes appear in the two trees. Node d is in a set of $f = 6$ children and appears in $\frac{q}{f}\frac{f!}{(f-q)!} = 60$ bases, i.e., the edge frequency is $\phi^{wpq}((\mathtt{b},\mathtt{d}), T_x) = 60$. Node m has only a few siblings, and $\phi^{wpq}((\mathtt{c},\mathtt{m}), T_x) = 6$.

The windowed pq-gram bases ($w = 3$, $q = 2$) that contain d and m are {hd, id, de, df} and {nm, om, mn, mo}, respectively, thus $\phi^{wpq}((\mathtt{b},\mathtt{d}), T_x) = \phi^{wpq}((\mathtt{c},\mathtt{m}), T_x) = 4$, and the distances are identical: $\mathrm{D}^{wpq}(T_x, T_y) = \mathrm{D}^{wpq}(T_x, T_z) = 10$.

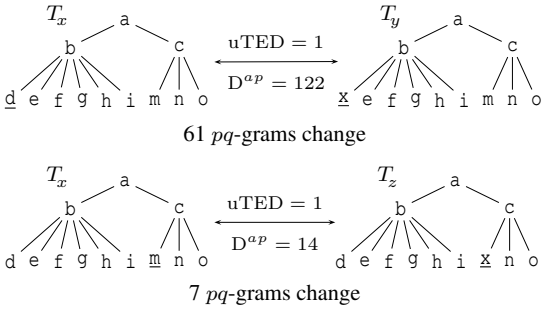


Fig. 9 All-Permutations pq-Grams Do Not Preserve Structure Information.

The following lemma shows that the structure error of windowed pq-grams with stems size $p = 1$ is zero.

Lemma 1 (Optimal Stem Size) *Let T be a tree and $X^{wpq}(T)$ be the set of all windowed pq-grams of T with $p = 1$ and $w \geq q > 0$. Then the structure error is zero for all edges in the tree, i.e.,*

$$\forall e \in E(T) : \phi^{wpq}_{\epsilon}(e, T) = 0. \tag{6}$$

Proof We show that the edge frequency is the same for all edges $e \in E(T)$: $\phi^{wpq}(e, T) = q\binom{w-1}{q-1}$. In a windowed pq-gram with $p = 1$ any edge $e = (n, b)$ connects the anchor node n and a base node b. Thus, the frequency of edge e is equal to the number of bases in which b appears. Case $q = 1$: Each node forms exactly one base, thus $\phi^{wpq}(e, T) = 1$ and (6) holds. Case $q \geq 2$: Node b appears in w windows, and only bases formed in these windows contain b. With b in the first window position $\binom{w-1}{q-1}$ bases are formed. For each of the other $w - 1$ positions $\binom{w-2}{q-2}$ bases that contain b and the first node of the window are formed. We get $\phi^{wpq}(e, T) = \binom{w-1}{q-1} + (w-1)\binom{w-2}{q-2} = q\binom{w-1}{q-1}$. □

Larger stems of size $p \geq 2$ explicitly store ancestor-descendant relationships of depth p. They give more weight to nodes with many children, and the structure error can be larger than zero. In some situations it is beneficial to accept a non-zero structure error for the additional information encoded the stems. For example, for the address trees discussed in Section 9.3.2, larger stems are more effective.

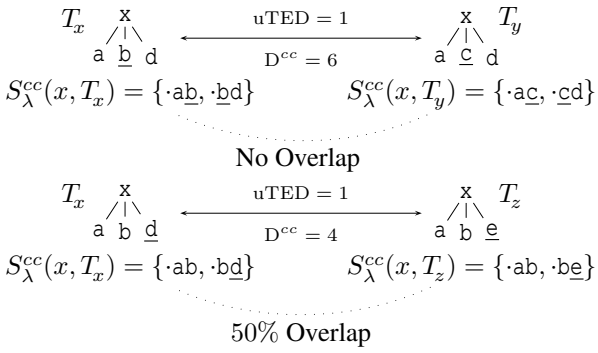


Fig. 10 Consecutive Children pq-Grams Do Not Preserve Children Information.

7.2 Preservation of Children Information and Base Size

To preserve children information, the children error should be small, and it should be independent of the sorting order of the children labels. This is not the case for consecutive children pq-grams, where the children error varies along with the labels of the non-matching children. The children error is high if the non-matching children in the sorted children sequence appear between matching children, and low otherwise. Since the children error depends on the labels of the changed nodes, the distance for consecutive children pq-grams depends on the node labels, too.

Example 10 Consider the trees in Figure 10. We form consecutive children pq-grams for the children of x in T_x, T_y, and T_z, respectively. With $C(x, T_x) = \{\mathtt{a},\mathtt{b},\mathtt{d}\}$, $C(x, T_y) = \{\mathtt{a},\mathtt{c},\mathtt{d}\}$, and $C(x, T_z) = \{\mathtt{a},\mathtt{b},\mathtt{e}\}$ the children error is determined as follows ($q = 2$):

- $J(C(x, T_x), C(x, T_y)) = J(C(x, T_x), C(x, T_z)) = 2/3$
- $J(S^{cc}_{\lambda}(x, T_x), S^{cc}_{\lambda}(x, T_y)) = 0$ (no overlap)
- $\varepsilon^{cc}(x, T_x, x, T_y) = 2/3$
- $J(S^{cc}_{\lambda}(x, T_x), S^{cc}_{\lambda}(x, T_z)) = 1/2$ (50% overlap)
- $\varepsilon^{cc}(x, T_x, x, T_z)) = 1/6$

Though T_y and T_z are at the same distance $\mathrm{uTED} = 1$ from T_x, the distances for consecutive children pq-grams differ:

- $|X^{cc}_{\lambda}(T_x) \uplus X^{cc}_{\lambda}(T_y)| = 10,\ |X^{cc}_{\lambda}(T_x) \uplus X^{cc}_{\lambda}(T_y)| = 2$
- $D^{cc}(T_x, T_y) = 6$
- $|X^{cc}_{\lambda}(T_x) \uplus X^{cc}_{\lambda}(T_z)| = 10,\ |X^{cc}_{\lambda}(T_x) \uplus X^{cc}_{\lambda}(T_z)| = 3$
- $D^{cc}(T_x, T_z) = 4$

For windowed pq-grams ($w = 3$, $q = 2$), we get

- $S^{wpq}(x, T_x) = \{\cdot\mathtt{ab}, \cdot\mathtt{ad}, \cdot\mathtt{bd}, \cdot\mathtt{ba}, \cdot\mathtt{da}, \cdot\mathtt{db}\}$
- $S^{wpq}(x, T_y) = \{\cdot\mathtt{ac}, \cdot\mathtt{ad}, \cdot\mathtt{cd}, \cdot\mathtt{ca}, \cdot\mathtt{da}, \cdot\mathtt{dc}\}$

- $S^{wpq}(x, T_z) = \{\cdot\texttt{ab}, \cdot\texttt{ae}, \cdot\texttt{be}, \cdot\texttt{ba}, \cdot\texttt{ea}, \cdot\texttt{eb}\}$
- $J(S_\lambda^{wpq}(x, T_x), S_\lambda^{wpq}(x, T_y)) = 1/3$
- $J(S_\lambda^{wpq}(x, T_x), S_\lambda^{wpq}(x, T_z)) = 1/3$
- $\varepsilon(x, T_x, x, T_y) = \varepsilon(x, T_x, x, T_z) = 1/3$
- $\mathrm{D}^{wpq}(T_x, T_y) = \mathrm{D}^{wpq}(T_x, T_z) = 10$

That is, the children error between T_x and T_y is the same as between T_x and T_z.

The following lemma shows that the children error for bases of size $q = 2$ is smaller than for bases of size $q > 2$.

Lemma 2 (Optimal Base Size) *Let n be a node with f children and assume one of the following edit sequences that transform T_1 into T_2:*

a) k insertions of new children of n
b) k children of n are renamed ($k \leq f$)
c) k children of n are deleted ($k \leq f$)

For mutually different children labels and a window size $w \leq \min(|C(n, T_1)|, |C(n, T_2)|)$, windowed pq-grams with bases of size 2, denoted as $S^{wpq=2}(n, T_1)$, have equal or smaller children error than pq-grams with bases of size $q > 2$, denoted as $S^{wpq>2}(n, T_1)$, i.e.,

$$\varepsilon^{wpq=2}(n, T_1, n, T_2) \leq \varepsilon^{wpq>2}(n, T_1, n, T_2) \tag{7}$$

Proof We first show that (7) follows from (8) below, then we show (8). We use the following abbreviations: $C_\lambda = C_\lambda(n, T_1)$, $\tilde{C}_\lambda = C_\lambda(n, T_2)$, $S_\lambda^{=2} = S_\lambda^{wpq=2}(n, T_1)$, $\tilde{S}_\lambda^{>2} = S_\lambda^{wpq>2}(n, T_2)$, etc.

$$\frac{|C_\lambda \uplus \tilde{C}_\lambda|}{|C_\lambda \uplus \tilde{C}_\lambda|} \geq \frac{|S_\lambda^{=2} \uplus \tilde{S}_\lambda^{=2}|}{|S_\lambda^{=2} \uplus \tilde{S}_\lambda^{=2}|} \geq \frac{|S_\lambda^{>2} \uplus \tilde{S}_\lambda^{>2}|}{|S_\lambda^{>2} \uplus \tilde{S}_\lambda^{>2}|} \tag{8}$$

$$\overset{(1)}{\Rightarrow} J(C_\lambda, \tilde{C}_\lambda) \geq J(S_\lambda^{=2}, \tilde{S}_\lambda^{=2}) \geq J(S_\lambda^{>2}, \tilde{S}_\lambda^{>2}) \tag{9}$$

$$\Rightarrow J(S_\lambda^{=2}, \tilde{S}_\lambda^{=2}) \geq J(S_\lambda^{>2}, \tilde{S}_\lambda^{>2})$$

$$\Leftrightarrow J(S_\lambda^{=2}, \tilde{S}_\lambda^{=2}) - J(C_\lambda, \tilde{C}_\lambda) \geq J(S_\lambda^{>2}, \tilde{S}_\lambda^{>2}) - J(C_\lambda, \tilde{C}_\lambda)$$

$$\overset{(9)}{\Rightarrow} |J(S_\lambda^{=2}, \tilde{S}_\lambda^{=2}) - J(C_\lambda, \tilde{C}_\lambda)| \leq |J(S_\lambda^{>2}, \tilde{S}_\lambda^{>2}) - J(C_\lambda, \tilde{C}_\lambda)|$$

$$\overset{(2)}{\Rightarrow} (7)$$

In order to show (8) we compute $|S_\lambda \uplus \tilde{S}_\lambda|$ and $|S_\lambda \pitchfork \tilde{S}_\lambda|$ for any $q \geq 2$.

$|S_\lambda \uplus \tilde{S}_\lambda|$: we systematically form all windowed pq-grams with anchor node n in T_1 and T_2 as follows: the children are sorted, a window of size w is shifted over the resulting sequence, and for each window all subsequences of length q form a base of a pq-gram. For each window $\binom{w-1}{q-1}$ bases (i.e., pq-grams) are produced, thus

$$|S_\lambda \uplus \tilde{S}_\lambda| = |C_\lambda \uplus C_\lambda| \binom{w-1}{q-1} \tag{10}$$

$|S_\lambda \pitchfork \tilde{S}_\lambda|$: We call $C_\lambda \pitchfork \tilde{C}_\lambda$ the unchanged labels and $(C_\lambda \setminus \tilde{C}_\lambda) \uplus (\tilde{C}_\lambda \setminus C_\lambda)$ the changed labels. For all pq-grams $s \in S$ in the case of rename and deletion, and for all pq-grams $s \in \tilde{S}$ in the case of insertion the following holds: The label tuple of s (with wildcard stem) is in $S_\lambda \pitchfork \tilde{S}_\lambda$ iff its base does not contain a changed label. Thus we can compute $|S_\lambda \pitchfork \tilde{S}_\lambda|$ by counting bases without changed labels.

Consider how the window is shifted over the sorted sibling sequence to produce the bases. If the window starts with a changed label, no pq-gram produced for this window position is in $S_\lambda \pitchfork \tilde{S}_\lambda$. Only if the window starts with an unchanged label, pq-grams for $S_\lambda \pitchfork \tilde{S}_\lambda$ are produced. We number the unchanged labels with the integers $1 \leq i \leq |C_\lambda \pitchfork \tilde{C}_\lambda|$ and define k_i to be the number of changed labels that appear in the window that starts with the unchanged label number i. Then $\binom{w-k_i-1}{q-1}$ of the $\binom{w-1}{q-1}$ pq-grams produced at window position i are in $S_\lambda \pitchfork \tilde{S}_\lambda$:

$$|S_\lambda \pitchfork \tilde{S}_\lambda| = \sum_{i=1}^{|C_\lambda \pitchfork \tilde{C}_\lambda|} \binom{w-k_i-1}{q-1} \tag{11}$$

With (10) and (11) we compute $|S_\lambda \pitchfork \tilde{S}_\lambda| / |S_\lambda \uplus \tilde{S}_\lambda|$, $q \geq 2$:

$$\begin{aligned}
\frac{|S_\lambda \pitchfork \tilde{S}_\lambda|}{|S_\lambda \uplus \tilde{S}_\lambda|} &\overset{(10)(11)}{=} \frac{1}{|C_\lambda \uplus \tilde{C}_\lambda|} \sum_{i=1}^{|C_\lambda \pitchfork \tilde{C}_\lambda|} \frac{\binom{w-k_i-1}{q-1}}{\binom{w-1}{q-1}} \\
&= \frac{1}{|C_\lambda \uplus \tilde{C}_\lambda|} \sum_{i=1}^{|C_\lambda \pitchfork \tilde{C}_\lambda|} \frac{\frac{(w-k_i-1)!}{(q-1)!(w-k_i-q)!}}{\frac{(w-1)!}{(q-1)!(w-q)!}} \\
&= \frac{1}{|C_\lambda \uplus \tilde{C}_\lambda|} \sum_{i=1}^{|C_\lambda \pitchfork \tilde{C}_\lambda|} \Big[\frac{(w-k_i-1)!}{(w-1)!} \frac{(w-q)!}{(w-k_i-q)!}\Big]
\end{aligned} \tag{12}$$

Only the term $\frac{(w-q)!}{(w-k_i-q)!}$ depends on q. We expand the term:

$$\frac{(w-q)!}{(w-k_i-q)!} = (w-k_i-q+1)(w-k_i-q+2)\dots(w-q)$$

Each factor on the right side of the equation increases with smaller q values, thus $\frac{(w-q)!}{(w-k_i-q)!}$ is larger for smaller q, and the right-hand inequality of (8) follows:

$$\frac{|S_\lambda^{=2} \pitchfork \tilde{S}_\lambda^{=2}|}{|S_\lambda^{=2} \uplus \tilde{S}_\lambda^{=2}|} \geq \frac{|S_\lambda^{>2} \pitchfork \tilde{S}_\lambda^{>2}|}{|S_\lambda^{>2} \uplus \tilde{S}_\lambda^{>2}|}$$

Next we show the left-hand inequality of (8):

$$\frac{|C_\lambda \pitchfork \tilde{C}_\lambda|}{|C_\lambda \uplus \tilde{C}_\lambda|} \geq \frac{|S_\lambda^{=2} \pitchfork \tilde{S}_\lambda^{=2}|}{|S_\lambda^{=2} \uplus \tilde{S}_\lambda^{=2}|}$$

For $q = 2$:

$$\frac{|S_\lambda^{=2} \pitchfork \tilde{S}_\lambda^{=2}|}{|S_\lambda^{=2} \uplus \tilde{S}_\lambda^{=2}|} \overset{(12)}{=} \frac{1}{|C_\lambda \uplus \tilde{C}_\lambda|} \sum_{i=1}^{|C_\lambda \pitchfork \tilde{C}_\lambda|} \frac{w-k_i-1}{w-1}$$

From $k_i \geq 0$ we follow $\frac{w-k_i-1}{w-1} \leq 1$.

$$\Rightarrow \sum_{i=1}^{|C_\lambda \pitchfork \tilde{C}_\lambda|} \frac{w-k_i-1}{w-1} \leq |C_\lambda \pitchfork \tilde{C}_\lambda|$$

$$\Rightarrow \frac{|S_\lambda^{=2} \pitchfork \tilde{S}_\lambda^{=2}|}{|S_\lambda^{=2} \uplus \tilde{S}_\lambda^{=2}|} \leq \frac{|C_\lambda \pitchfork \tilde{C}_\lambda|}{|C_\lambda \uplus \tilde{C}_\lambda|}$$

□

7.3 Preservation of Sibling Information and Window Size

Single leaf pq-grams do not preserve sibling information and therefore fail to detect node moves to another parent if the ancestors in the old and the new position have identical labels. Ancestors with identical labels are frequent in data-centric XML (e.g., all `title` elements have the ancestors `track` and `album` in the XML of Figure 1). pq-Grams with larger bases encode sibling information and can detect node moves since nodes with homonymous ancestors may have siblings with different labels. A node move is detected if at least one of the pq-grams changes.

Example 11 Single leaf pq-grams cannot distinguish trees T_x and T_y in Figure 11 since $X^{sl}_\lambda(T_x) = X^{sl}_\lambda(T_y)$ and therefore $\mathrm{D}^{sl}(T_x, T_y) = 0$. The ancestors of the moved node `d` have identical labels, which yields identical stems. No sibling information is encoded, and snippet recall and snippet precision are zero. Windowed pq-grams encode sibling information and distinguish the two trees. The moved node `d` has a sibling `c` in T_x but not in T_y, thus the snippet `bcd` exists only in T_x and distinguishes it from T_y, and $\mathrm{D}^{wpq}(T_x, T_y) = 8$ ($w = 3$, $p = 1$, $q = 2$). The snippets of node $(2,\mathtt{b})$ in Figure 11 are $S^{wpq=2}_\lambda((2,\mathtt{b}), T_x) = \{\cdot\mathtt{cd}, \cdot\mathtt{c}\star, \cdot\mathtt{d}\star, \cdot\mathtt{dc}, \cdot\star\mathtt{c}, \cdot\star\mathtt{d}\}$. Snippets $\cdot\mathtt{cd}$ and $\cdot\mathtt{dc}$ are duplicates and all other snippets contain dummy nodes, thus $\mathrm{encpairs}((2,\mathtt{b}), T_x) = \{\{\mathtt{c},\mathtt{d}\}, \{\mathtt{c},\star\}, \{\mathtt{d},\star\}\}$, $\mathrm{allpairs}((2,\mathtt{b}), T_x) = \{\{\mathtt{c},\mathtt{d}\}\}$, snippet recall $\rho((2,\mathtt{b}), T_x) = 1$ (the snippets encode all pairs of children of n), and snippet precision $\pi((2,\mathtt{b}), T_x) = \frac{1}{3}$ (1 of 3 encoded pairs are relevant for detecting node moves).

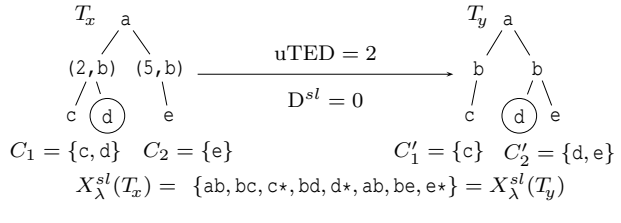


Fig. 11 Single Leaf pq-Grams Do Not Preserve Sibling Information.

Lemma 3 (Node Move Detection) *Assume node n is moved from parent u to parent v: $C(u,T_2) = C(u,T_1)\setminus\{n\}$ and $C(v,T_2) = C(v,T_1)\cup\{n\}$. If for window size w the node move is detected by pq-grams with bases of size $q > 2$ then it is also detected by pq-grams with bases of size 2, i.e.,*

$$S^{wpq>2}_\lambda(u,T_1) \uplus S^{wpq>2}_\lambda(v,T_1) \neq S^{wpq>2}_\lambda(u,T_2) \uplus S^{wpq>2}_\lambda(v,T_2)$$
$$\Rightarrow$$
$$S^{wpq=2}_\lambda(u,T_1) \uplus S^{wpq=2}_\lambda(v,T_1) \neq S^{wpq=2}_\lambda(u,T_2) \uplus S^{wpq=2}_\lambda(v,T_2)$$

Proof If there is a base of any size $q \le w$ that contains the moved node n and a sibling m that distinguishes $C(u,T_1)$ from $C(v,T_2)$, then either there is a window that has m as the first node and contains n, or vice versa. In either case for $q = 2$ a base containing both n and m is formed, and the node move is detected. □

Lemma 4 (Recall and Precision) *Let n be a node with $f \ge 2$ children, $S^{wpq=2}(n,T)$ be the windowed pq-grams of n with base size $q = 2$ and window size $w \ge q$. Snippet recall, $\rho^{wpq=2}(n,T)$, and snippet precision, $\pi^{wpq=2}(n,T)$, are*

$$\rho = \begin{cases} 2\frac{w-1}{f-1} & w < \frac{f+1}{2} \\ 1 & w \ge \frac{f+1}{2} \end{cases} \quad \textit{and} \quad \pi = \begin{cases} 1 & w < \frac{f+1}{2} \\ \frac{f-1}{2(w-1)} & w \ge \frac{f+1}{2} \end{cases}$$

Proof We need to compute the number of node pairs that can be formed from the children of n, and the number of node pairs encoded by the bases. The number of node pairs that can be formed from $f \ge 2$ children is $\mathrm{allpairs}^{wpq=2}(n,T) = \binom{f}{q} = \frac{f(f-1)}{2}$. *Small windows* ($w < \frac{f+1}{2}$) produce bases without duplicates and dummy nodes, thus $\mathrm{encpairs}^{wpq=2}(n,T) = f(w-1)$ is equal to the total number of bases of n. With (3) and (4), $\rho = 2\frac{w-1}{f-1}$ and $\pi = 1$. *Large windows* ($w \ge \frac{f+1}{2}$): All possible sibling pairs are encoded by the $f(w-1)$ bases of n, thus $\mathrm{encpairs}^{wpq=2}(n,T) = \frac{f(f-1)}{2}$, $\rho = 1$, and $\pi = \frac{f-1}{2(w-1)}$. □

7.4 Optimal Windowed pq-Grams

The following theorem specifies the optimal parameters p, q, and w for windowed pq-grams such that the three properties for tree decompositions are maximally preserved.

Theorem 4 *(Optimal Windowed pq-Grams) Assume a tree with fanout $f \ge 2$ for the non-leaf nodes. Windowed pq-grams with stem size $p = 1$, base size $q = 2$, and window size $w = \frac{f+1}{2}$ maximally preserve structure, children and sibling information:*

(a) *Structure Error:* $\phi_\epsilon = 0$ *for all edges*

(b) *Children Error:* $\varepsilon \le \begin{cases} \frac{k}{f} & \textit{for rename} \\ \frac{2k}{2f+k} & \textit{for insert} \\ \frac{2k}{2f-k} & \textit{for delete} \end{cases}$

(c) *Snippet Recall:* $\rho = 1$ *for* $w = \lceil\frac{f+1}{2}\rceil$

(d) *Snippet Precision:* $\pi = 1$ *for* $w = \lfloor\frac{f+1}{2}\rfloor$

Proof (a) Structure Error. Follows from Lemma 1.

(b) Children Error. Let C (pq-grams $S^{wpq=2}(n,T_1)$) be children with mutually different labels $C_\lambda(n,T_1)$, let C be transformed to $C(n,T_2)$ (pq-grams $S^{wpq=2}(n,T_2)$) by a sequence of k edit operations according to Lemma 2. k_i is the

number of changed nodes in the i-th window that starts with an unchanged node.

Insert:

- $|C_\lambda(n,T_1) \uplus C_\lambda(n,T_2)| = f$
- $|C_\lambda(n,T_1) \uplus C_\lambda(n,T_2)| = 2f + k$
- $|S_\lambda^{wpq=2}(n,T_1) \uplus S_\lambda^{wpq=2}(n,T_2)| \stackrel{(11)}{=} \sum_{i=1}^{f} \binom{w-1-k_i}{q-1} = f(w-1) - \sum_{i=1}^{f} k_i$
- $|S_\lambda^{wpq=2}(n,T_1) \uplus S_\lambda^{wpq=2}(n,T_2)| \stackrel{(10)}{=} (2f+k)\binom{w-1}{q-1} = (2f+k)(w-1)$

A new node can appear in at most $w-1$ different windows that start with a node from $C(n,T_1)$, contributing to $w-1$ k_i-values, thus $\sum_{i=1}^{f} k_i \leq k(w-1)$ and $|S_\lambda^{wpq=2}(n,T_1) \uplus S_\lambda^{wpq=2}(n,T_2)| \geq f(w-1) - k(w-1)$. With (2), $\varepsilon \leq \frac{2k}{2f+k}$.

Delete:

- $|C_\lambda(n,T_1) \uplus C_\lambda(n,T_2)| = f - k$
- $|C_\lambda(n,T_1) \uplus C_\lambda(n,T_2)| = 2f - k$
- $|S_\lambda^{wpq=2}(n,T_1) \uplus S_\lambda^{wpq=2}(n,T_2)| \stackrel{(11)}{=} (f-k)(w-1) - \sum_{i=1}^{f-k} k_i$
- $|S_\lambda^{wpq=2}(n,T_1) \uplus S_\lambda^{wpq=2}(n,T_2)| \stackrel{(10)}{=} (2f-k)(w-1)$

Rename:

- $|C_\lambda(n,T_1) \uplus C_\lambda(n,T_2)| = f - k$
- $|C_\lambda(n,T_1) \uplus C_\lambda(n,T_2)| = 2f$
- $|S_\lambda^{wpq=2}(n,T_1) \uplus S_\lambda^{wpq=2}(n,T_2)| \stackrel{(11)}{=} (f-k)(w-1) - \sum_{i=1}^{f-k} k_i$
- $|S_\lambda^{wpq=2}(n,T_1) \uplus S_\lambda^{wpq=2}(n,T_2)| \stackrel{(10)}{=} 2f(w-1)$

The children error follows from (2) and $\sum_{i=1}^{f-k} k_i \leq k(w-1)$.

(c) Recall and Precision. Follows immediately from Lemma 4 by setting $w = \frac{f+1}{2}$. □

The optimal size of w depends on the fanout f. For a tree that consists only of the root node and $n-1$ leaves, the optimal value is $w = (f+1)/2 = O(n)$. Even in this case, the optimal windowed pq-gram profile can not grow larger than $O(n^2)$ since for $q = 2$ and $f \geq w$ the profile size is $|X^{wpq}(T)| = (n-1)(w-1) + l$ (cf. Theorem 1).

Choosing bases of size $q > 2$ can be useful despite the higher base error. For example, larger bases give more weight to changes in the leaf nodes: The number of windowed pq-grams that are affected by a leaf change only depends on the number of bases in which the changed node appears (the number of stems is always one for leaves), and the number of affected bases increases with q (cf. Eq. (8)). For trees with fanout $f = O(n)$ (see above), bases of size $q > 2$ in combination with the optimal window size $w = (f+1)/2$ lead to large profiles. In this case it is not efficient to use the optimal window size, but a smaller constant must be used.

8 Algorithms

8.1 Building the pq-Gram Index

Algorithm 1 computes the windowed pq-gram profile X^{wpq} for $q = 2$ by recursively traversing the tree T in preorder. The algorithm is initialized with the root node n of T, the window size w, a stem of dummy nodes $(\bullet_1, \ldots, \bullet_p)$, and the empty profile $X^{wpq} = \emptyset$. Whenever the last child (in document order) of a node is reached, the children are sorted (dummy nodes to the end), and the windowed pq-grams are produced. The runtime is $O(n + f_{max} \log f_{max})$ for documents with n nodes, a maximal fanout of f_{max}, and constant window size. Our experiments confirm the analytic runtime result.

Algorithm 1: `getPQGrams`$(T, n, w, \text{stem}, X^{wpq})$

```
stem ← dequeue-first-element(stem) ∘ n;
if n is a leaf then return X^wpq ∪ {(T, stem ∘ (•,•))};
C ← ∅;
foreach child c of n do
    C ← C ∪ {c};
    X^wpq ← X^wpq ∪ getPQGrams(T, c, w, stem, X^wpq);
C ← C ∪ ⋃_{i=1}^{w-f} {•};
a ← sort-by-label(C);
for i ← 0 to |a| − 1 do
    for j ← i + 1 to i + w − 1 do
        X^wpq ← X^wpq ∪ {(T, stem ∘ a[i] ∘ a[j mod |a|])};
return X^wpq;
```

The index, X_λ^{wpq}, is computed by aggregating and counting the label tuples of the windowed pq-grams in the profile: $X_\lambda^{wpq} \leftarrow \Gamma_{tid,\lambda(pqg)\rightarrow pqg,\text{COUNT}(*)\rightarrow cnt}(X^{wpq})$. The runtime is $O(n \log n)$ (sorting the profile of size $O(n)$). The index of a forest is the union of the indexes of its trees.

To deal with node labels of different length, such as element names and text values in XML documents, we use a fingerprint hash function that maps a string s to a hash value $h(s)$ of fixed length that is unique with a high probability (e.g., the Karp-Rabin fingerprint function [23]). Instead of storing the label tuples of windowed pq-grams, we store the concatenation of the hashed labels. Note that the only operation we need to perform on the labels is to check equality.

Example 12 Figure 12 shows an example hash function and part of the windowed pq-gram indexes of the two XML documents in Figure 1, the music albums from the song lyric store (T_{LS}) and the CD warehouse (T_{WH}). We choose $p = q = 2$, $w = 3$, $\lambda(\bullet) = (\star,\star)$. The label tuple $((\star,\star),(\texttt{album},\epsilon),(\texttt{track},\epsilon),(\texttt{track},\epsilon))$ with hash value `9999 4100 3200 3200` appears twice in the index of T_{LS} and has two matches in the other index. The label tuple $((\texttt{album},\epsilon),(\texttt{year},\texttt{2000}),(\star,\star),(\star,\star))$ with the hash value

s	$h(s)$
*	99
ϵ	00
album	41
track	32
title	02
artist	11
year	54
price	19

s	$h(s)$
So far away	67
Mark	86
John	15
2000	97
15	73
Wish you where here	42
Roger	26
Dave	09
Nick	37

(a) Hash Function.

tid	pqg	cnt
...	...	...
T_{LS}	9999 4100 3200 5497	2
T_{LS}	9999 4100 3200 3200	2
T_{LS}	9999 4100 5497 3200	2
T_{LS}	4100 5497 9999 9999	1
T_{LS}	4100 3200 0267 1186	1
T_{LS}	4100 3200 0267 1115	1
T_{LS}	4100 3200 1186 1115	1
...	...	...

(b) *pq*-Gram Index of the Song Lyric Store.

tid	pqg	cnt
...	...	...
T_{WH}	9999 4100 3200 3200	2
T_{WH}	9999 4100 3200 1973	2
T_{WH}	9999 4100 1973 3200	2
T_{WH}	4100 1973 9999 9999	1
T_{WH}	4100 3200 0267 1115	1
T_{WH}	4100 3200 0267 1186	1
T_{WH}	4100 3200 1115 1186	1
...	...	...

(c) *pq*-Gram Index of the CD Warehouse.

Fig. 12 Implementation of the Windowed pq-Gram Index.

4100 5497 9999 9999 appears only once in the index of T_{LS} and has no match in the index of T_{WH}.

8.2 Approximate XML Join

Algorithm 2 computes the approximate join between two sets of trees, F_1 and F_2. The input to the algorithm are the windowed pq-gram indexes of the two sets, $X1_\lambda^{wpq}$ and $X2_\lambda^{wpq}$, respectively, and a threshold, $\tau < 1$. The algorithm returns all pairs $(T_i, T_j) \in F_1 \times F_2$ that satisfy $\mathrm{D}_n^{wpq}(T_i, T_j) \leq \tau$, where $\mathrm{D}_n^{wpq}(T_i, T_j)$ is the normalized windowed pq-gram distance between T_i and T_j. PS_1 and PS_2 are initialized with the profile sizes for the trees in F_1 and F_2, respectively, which are used for the normalization. In the selection predicate, $isec = |X_\lambda^{wpq}(T_i) \Cap X_\lambda^{wpq}(T_j)|$ and $size_1 + size_2 = |X_\lambda^{wpq}(T_i) \uplus X_\lambda^{wpq}(T_j)|$ for each pair of trees (T_i, T_j).

Algorithm 2: $\mathtt{pqGramJoin}(X1_\lambda^{wpq}, X2_\lambda^{wpq}, \tau)$

$\mathsf{PS}_1 \leftarrow \Gamma_{tid_1, \mathrm{SUM}(cnt_1) \to size_1}(X1_\lambda^{wpq})$;
$\mathsf{PS}_2 \leftarrow \Gamma_{tid_2, \mathrm{SUM}(cnt_2) \to size_2}(X2_\lambda^{wpq})$;
$A \leftarrow \rho_{tid/tid_1, cnt/cnt_1}(X1_\lambda^{wpq}) \bowtie \rho_{tid/tid_2, cnt/cnt_2}(X2_\lambda^{wpq})$;
$B \leftarrow \Gamma_{tid_1, tid_2, \mathrm{SUM}(\min(cnt_1, cnt_2)) \to isec}(A)$;
$C \leftarrow B \bowtie \mathsf{PS}_1 \bowtie \mathsf{PS}_2$;
$D \leftarrow \pi_{tid_1, tid_2}(\sigma_{1 - \frac{isec}{size_1 + size_2 - isec} \leq \tau}(C))$;
return D;

Example 13 Figure 13 illustrates Algorithm 2 and shows the windowed pq-gram join ($w = 3$, $p = q = 2$) between two sets of example trees, F_1 and F_2. To increase the readability, the node labels of the windowed pq-grams are shown instead of their hash values. The input to the algorithm are the windowed pq-gram indexes, $X1_\lambda^{wpq}$ and $X2_\lambda^{wpq}$, of the two forests, F_1 and F_2, respectively. First, the tables PS_1 and PS_2 are computed as shown in the figure. Then the indexes are joined (Table A). Next, the intersection of the pq-gram indexes is computed (Table B) unless the intersection is empty. The intersection result is extended with the size of the pq-gram profiles (Table C). Finally, all pairs of trees within windowed pq-gram distance τ are selected (Table D).

As pointed out by Guha et al. [19], hash and sort-merge joins do not carry over to approximate tree joins that use the edit distance since the distance function must be evaluated between every input pair. There is no effective way to sort trees or partition them into buckets with a hash function. The only approach readily applicable is the nested loop join [19].

This is different for the windowed pq-gram distance. For the calculation of the windowed pq-gram distance a tree is represented by its windowed pq-gram index. Instead of computing the distance between each pair of trees directly, we check for each windowed pq-gram in which pairs of trees it appears. We transform the distance-based join to an equality join on all windowed pq-grams represented as strings. The approximate join is computed by counting windowed pq-grams in the join result. We can apply well known techniques to optimize this join (e.g., sort-merge and hash join). In addition, the size of the intermediate join result can be reduced by using so-called prefix filters, which were developed for equality joins that compute intersections [32].

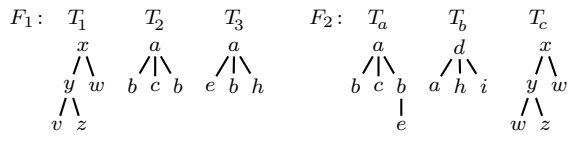
In the worst case the forests consist of identical copies of the same tree. Let N be the cardinality of the forest, n the number of nodes per tree. The indexes are of size $O(Nn)$ for a constant window size. In a sort-merge join the complexity of sorting the relations is $O(Nn \log(Nn))$. Each windowed pq-gram in one index matches $O(N)$ tuples in the other index. The overall complexity is $O(Nn(N + \log n))$. Note that for this worst case scenario the join result is of size $\Theta(N^2)$, thus no algorithm can improve on the quadratic runtime.

Different from the nested loop join, our join algorithm can take advantage of the diversity of trees in a forest. In particular, empty intersections between windowed pq-gram indexes are never computed. In the best case, when no two trees in the forest share windowed pq-grams, the runtime is $O(Nn \log(Nn))$ for index size $O(Nn)$.

9 Experiments

In this section we experimentally evaluate our solution and compare it to competing approaches. All algorithms are implemented as single-thread applications in Java 1.6 and run on a dual-core AMD64 server. We use MySQL 5.1 as a database server.

F_1: T_1 T_2 T_3 F_2: T_a T_b T_c



Index $X1^{wpq}_{\lambda}$

tid	pqg	cnt
T_1	*xyw	1
T_1	*xy*	1
T_1	*xw*	1
...	...	...
T_2	*abc	2
T_2	*abb	2
T_2	*acb	2
...	...	...
T_3	*aeb	1
T_3	*aeh	1
T_3	*abh	1
...	...	...

Index $X2^{wpq}_{\lambda}$

tid	pqg	cnt
T_a	*abc	2
T_a	*abb	2
T_a	*acb	2
...	...	...
T_b	*dah	1
T_b	*dai	1
T_b	*dhi	1
...	...	...
T_c	*xyw	1
T_c	*xy*	1
T_c	*xw*	1
...	...	...

PS_1

tid_1	$size_1$
T_1	15
T_2	9
T_3	9

PS_2

tid_2	$size_2$
T_1	15
T_2	9
T_3	15

$A = X1^{wpq}_{\lambda} \bowtie X2^{wpq}_{\lambda}$

pqg	tid_1	cnt_1	tid_2	cnt_2
*xyw	T_1	1	T_c	1
xy	T_1	1	T_c	1
xw	T_1	1	T_c	1
*abc	T_2	2	T_a	2
*abb	T_2	2	T_a	2
*acb	T_2	2	T_a	2
ab**	T_3	1	T_a	2
...	...	...	...	...

$B = \Gamma_{tid_1,tid_2,\mathrm{SUM}(\min(cnt_1,cnt_2))\rightarrow isec}(A)$

tid_1	tid_2	$isec$
T_1	T_c	10
T_2	T_a	9
T_3	T_a	1

$C = \mathsf{PS}_1 \bowtie B \bowtie \mathsf{PS}_2$

tid_1	tid_2	$isec$	$size_1$	$size_2$
T_1	T_c	10	15	15
T_2	T_a	9	9	15
T_3	T_a	1	9	15

$D = \pi_{tid_1,tid_2}(\sigma_{1-\frac{isec}{size_1+size_2-isec}\leq\tau}(C))$

tid_1	tid_2
T_1	T_c
T_2	T_a

Fig. 13 pq-Gram Join ($\tau = 0.5$) on Example Tree Sets F_1 and F_2.

9.1 Scalability of Profile and Index Computation

We study the scalability of the windowed pq-gram index computation using synthetic data from the standard XMark benchmark [25], whose documents model online auction data and combine complex structures and realistic text. There is a linear relation between the size of the XMark documents (in MB) and the number of nodes in the respective XML trees. The depth of the trees does not vary with the size and is 11 for all documents. Our test data range between 100kB and 1.2GB (2k to 20M nodes).

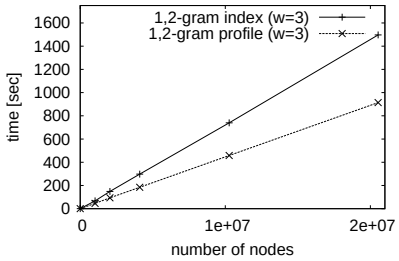


Fig. 14 Index Computation Time for Varying Document Size.

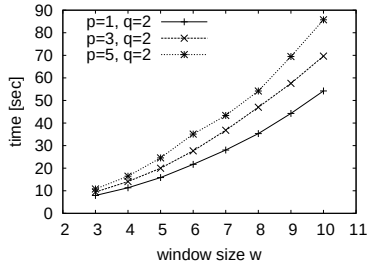
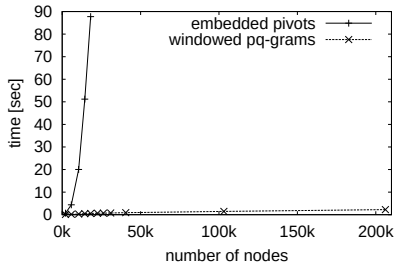


Fig. 15 Index Computation Time for Varying w and p.

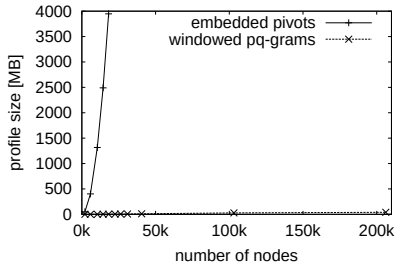
The index computation in Figure 14 includes the profile computation (Algorithm 1, $w = 3, p = 1, q = 2$) and the aggregation of duplicate pq-grams within each tree. The number of windowed pq-grams in the index is linear in the tree size and the index computation scales to very large trees.

In Figure 15 we study the runtime of the index computation as a function of the window size w for different stem sizes p ($q = 2$). We compute the pq-gram index for an XMark XML document of 5.75MB (103k nodes). The stem size p has little impact on the runtime; increasing p only adds nodes to each pq-gram, but does not increase the number of pq-grams in the index. The index cardinality grows with the window size w. The growth is linear if the window size is within the fanout of non-leaf nodes and is quadratic in the worst case. Although the fanout of many nodes in the tested XML document is smaller than the window size (61% of the nodes have a fanout of one, 90% have a fanout within 5, and the average fanout is 2.7), the runtime of the index creation scales well with the window size.

Next, we show the scalability of windowed pq-grams ($w = 3, p = 1, q = 2$) and compare them to embedded pivots [34]. Embedded pivots are snippets that consist of two nodes and their least common ancestor (cf. Section 2). We measure the runtime for decomposing XML documents of varying size (2k to 206k nodes) and the size of the resulting profile in megabytes. Figure 16 shows the results. As expected, windowed pq-grams are much faster than embedded pivots since there is only a linear number of windowed pq-



(a) Profile Computation Time.



(b) Profile Size.

Fig. 16 Windowed pq-Grams vs. Embedded Pivots.

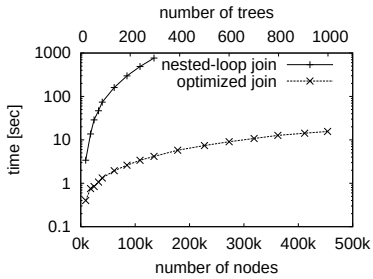


Fig. 17 Scalability of the Optimized Join with Windowed pq-Grams.

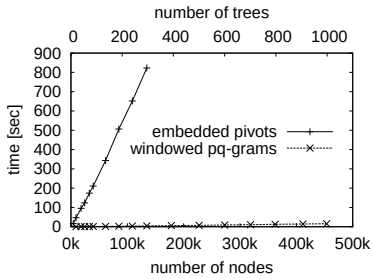


Fig. 18 Join Scalability of Windowed pq-Grams vs. Embedded Pivots.

grams but a quadratic number of embedded pivots. For example, decomposing a tree with 18k nodes into embedded pivots takes 87.7s and results in a profile of size 3.95GB; decomposing the same tree into windowed pq-grams takes 0.5s and produces a profile of size 2.7MB. The windowed pq-gram profile for the largest tree in Figure 16 (206k nodes) is computed in 2.2s and is of size 40MB.

Table 1 shows the size of the profiles and indexes for the three large real world XML databases described in detail in Section 9.3. Profile size and index size refer to the space used by the respective database tables, cardinality is the number of rows, i.e., the number of windowed pq-grams. Each label in a pq-gram is hashed to a 6 byte hash value. The index is up to a factor of three smaller than the profile since duplicate windowed pq-grams are stored only once. The index is at most about twice as large as the respective XML file; in the case of SwissProt, due to many duplicate windowed pq-grams, the cardinality of the index is even smaller than the number of the nodes in the XML database.

	XML File		wpq-Gram Profile		wpq-Gram Index	
	Size	Nodes	Size	Card.	Size	Card.
DBLP	531 MB	15.6M	1,052 MB	45.7M	725 MB	26.8M
SwissProt	110 MB	5.2M	348 MB	15.1M	97 MB	3.6M
TreeBank	82 MB	2.4M	187 MB	8.1M	146 MB	5.4M

Table 1 Profile and Index Size for Real World XML ($w = 3, p = 1$).

9.2 Approximate Join Based on Windowed pq-Grams

We compare the scalability of the optimized join (Algorithm 2) with the scalability of a join that computes the windowed pq-gram distance between each pair of trees (nested loop join). We produce two sets of synthetic trees, A and B. Each set consists of 1000 trees with 100 to 2434 nodes (454 nodes on average) and stores 10.9MB of data. The trees within a set are different, and each tree has a match in the other set. Figure 17 shows the wall clock time for joining subsets of A and B of different size. The optimized join computes only the distance between trees that have pq-grams in common. Unlike the nested loop join, it can take advantage of the diversity of the trees to avoid unnecessary distance computations. The runtime is close to linear.

The optimized join algorithm works for any tree decomposition. We compare the efficiency of our optimized join for windowed pq-grams with embedded pivots in Figure 18. For both approaches the join time is almost linear in the number of trees in the datasets. However, the scalability of embedded pivots is limited by the large index which is quadratic in the tree size. For example, joining two subsets with 300 trees takes 822.8s and produces an index with 50M tuples for embedded pivots vs. 4.2s and 0.38M tuples for windowed pq-grams.

9.3 Tree Matching

We investigate the use of windowed pq-grams in a tree matching scenario, where two sets of trees, F_1 and F_2, are given and a mapping $M^x \subseteq F_1 \times F_2$ is computed. An el-

ement of a mapping is a *pair*. The quality of the computed mapping, M^x, is evaluated with respect to a correct mapping $M^c \subseteq F_1 \times F_2$ that contains all pairs of trees that *should* match. We measure *precision*, $p = \frac{|M^x \cap M^c|}{|M^x|}$, (correctly computed pairs to total number of computed pairs) and *recall*, $r = \frac{|M^x \cap M^c|}{|M^c|}$, (correctly computed pairs to total number of correct pairs). The precision is high if the returned pairs are correct, the recall is high if the algorithm does not miss correct pairs. The F-measure, $F = \frac{2pr}{p+r}$, is a well-known performance measure [37] that considers both recall and precision. To account for the different tree sizes, we normalize all distances in our tree matching experiments to values between 0 and 1.

9.3.1 Matching Real World XML

We study the effectiveness of the windowed pq-gram distance for XML data. We use real world XML data sets, add noise (spelling mistakes, missing or additional elements), and we approximately join the original and the noisy set. We measure the effectiveness in terms of precision and recall.

Data Sets. We use the DBLP[2] (bibliography), the SwissProt[3] (protein sequence database), and the Treebank[4] (parts of speech tagged English sentences) XML databases. We split each database into a set of (sub)documents by deleting the root node, and we randomly choose 200 of the resulting documents for our experiments (requiring their size to be at least 15 nodes).

The resulting document sets are structurally very different: the DBLP subset contains small and flat documents (21 nodes and 2.1 levels on average) with a total of 4253 nodes, 3686 leaves, and 27 different tag names, the SwissProt documents are larger and deeper (98 nodes and 3.7 levels on average) with a total of 19529 nodes, 15975 leaves, and 84 different tag names, the Treebank documents have deep recursive structure (45 nodes and 10.9 levels on average, with a maximum of 22 levels) with a total of 8962 nodes, 5104 leaves, and 67 different tag names.

We add *noise* to the original documents by deleting, inserting, and renaming random nodes. Node deletions/insertions simulate missing/additional elements or attributes and modify the document structure. Renamed nodes represent different tag names or spelling mistakes in the text values. After each node edit operation the tree is permuted randomly. The permutation after each edit step is essential to model the edit operations of the unordered tree edit distance (cf. Section 4). The resulting noisy document is the *match* of the original document, all other noisy documents are *non-matches*. In our figures we show the noise as the percentage of changed nodes.

Distance Parameters. As suggested by Theorem 4, we use stems of size $p = 1$ and bases of size $q = 2$. For choosing the window size w, the theorem assumes a constant fanout for all non-leaf nodes in the tree. We approximate the constant fanout with the median of all non-leaf fanouts and compute $w = \max\{3, (f_{\text{median}} + 1)/2\} = 3$ for all datasets.

Distance between Matches and Non-Matches. Each original document has exactly one match. Figure 19 shows the average windowed pq-gram distance ($w = 3$, $p = 1$, $q = 2$) of the original documents to their match and to the closest non-match. The noise is increased in steps of 5%. The distance to the closest non-match shows that SwissProt documents are more similar to each other than the DBLP and Treebank documents. As expected, the windowed pq-gram distance between matches is zero for 0% noise and monotonically increases with the percentage of changed nodes. Since all documents are modified, also the distance to the non-matches increases with the number of changed nodes.

Measuring the Effectiveness. Figure 20 shows precision and recall for different noise levels and thresholds τ ($w = 3$, $p = 1$, $q = 2$). Increasing the threshold decreases the precision and increases the recall. If the threshold is too low for the noise level, then the mapping is empty, the recall is zero, and the precision is not defined.

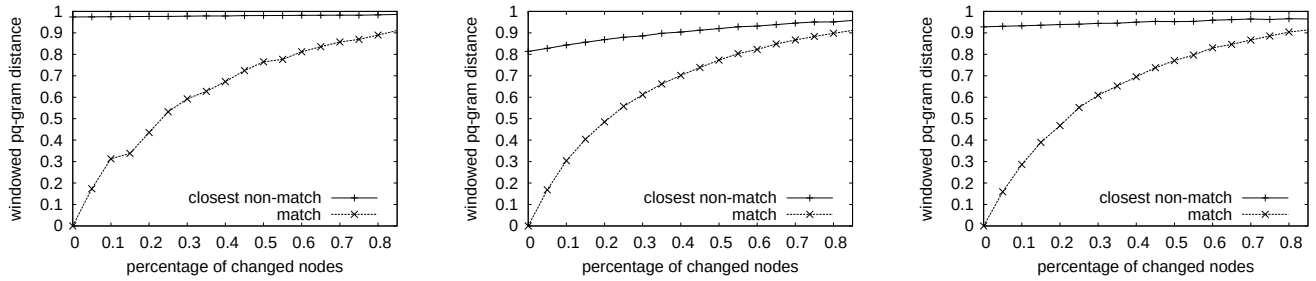
For DBLP and Treebank the precision is high, thus the threshold can be increased to $\tau = 0.7$ to get recall values of more than 95% for 20% noise. For SwissProt the precision drops as we increase the threshold. The SwissProt documents are clustered into groups of very similar documents (protein variants). The clustering of the data is evident from the precision values in Figure 20(b) for 0% noise (approximate self join): Already for $\tau = 0.3$ some documents form a pair with other documents rather than just themselves; for $\tau = 0.5$ approximately 10% of the documents form a pair with other documents.

We improve the result for SwissProt by using a variable threshold. Each document is paired with its nearest neighbor. If a document has more than one nearest neighbor, the document is not paired and does not appear in the mapping. Figure 21(b) shows the results for the SwissProt database. The algorithm returns precise pairs, and even for 45% noise we miss less than 3% of the correct pairs. The results for DBLP and SwissProt are similar.

9.3.2 Matching Address Data

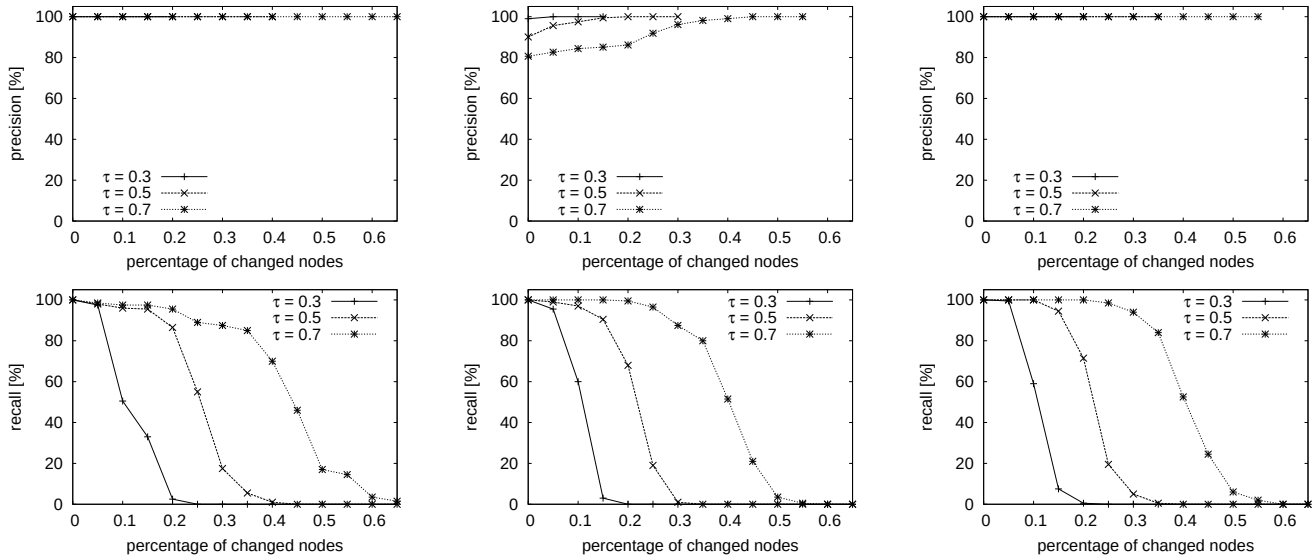
We test the effectiveness of windowed pq-grams on the Bolzano address trees[5]. This dataset consists of two sets of

[2] http://dblp.uni-trier.de
[3] http://us.expasy.org/sprot/
[4] http://www.cis.upenn.edu/~treebank/
[5] http://www.inf.unibz.it/~augsten/tods10/



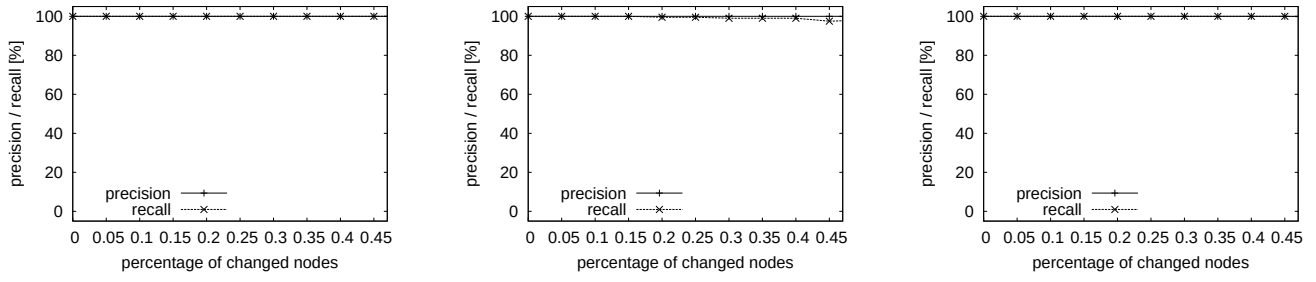
(a) DBLP (b) SwissProt (c) Treebank

Fig. 19 Distance between Matches and Non-Matches.



(a) DBLP (b) SwissProt (c) Treebank

Fig. 20 Matching with Different Thresholds.



(a) DBLP (b) SwissProt (c) Treebank

Fig. 21 Precision and Recall for 1:1 Matches (Nearest Neighbor).

trees, L and R, and each tree (called address tree) represents a street of the city of Bolzano with all its addresses. Each root-leaf path in the tree represents a residential address. The root node is the street name, the second level stores the house numbers, the third level the entrance numbers, and the fourth level the apartment numbers.

The trees are generated from two databases of the city administration that both store residential addresses. Although both databases cover the entire city, the residential addresses in the two databases differ due to spelling mistakes, missing addresses, street names that are spelled differently, and addresses stored at different levels of detail (e.g., with/without apartment numbers). Thus, address trees that

should match are different. The tree set L consists of 299 trees with 52,509 nodes in total, reflecting 44,427 addresses; the set R consists of 302 trees with 52,509 nodes and 43,187 addresses.

Street matching is challenging since there is a large overlap between the labels of the trees. All labels (except the street name) are either numbers (house or apartment numbers) or single characters (entrance names, e.g., 'A', 'B'). A tree decomposition may produce identical snippets at different positions in a tree (for example, a snippet `123` may be produced from three consecutive house numbers *or* apartment numbers), and all trees produce similar snippets.

The matching is done as follows. For each distance function D^x we compute a mapping $M^x \subseteq F_1 \times F_2$. Two trees $T_i \in F_1$ and $T_j \in F_2$ are paired, i.e., $(T_i, T_j) \in M^x$, iff T_i has only one nearest neighbor in F_2, namely T_j, and vice versa. We sort the trees and compute a mapping for our windowed pq-gram distance, the ordered tree edit distance [43] (see Section 4), the pq-gram distance [4, 6], the tree embedding distance [18], the binary branch distance [40], single path shingles [8], embedded pivots [34], and the node intersection distance. The node intersection distance is a simple algorithm that completely ignores the structure of the tree. It is computed in the same way as the windowed pq-gram distance, the only difference being that the index of a tree consists of the bag of all its node labels. The correct mapping, M^c, contains all pairs of trees that represent the same street in the real world and is computed by hand. There are three streets in R that do not exist in L, thus $|M^c| = 299$ for the computation of precision and recall.

The results are shown in Table 2. In terms of overall effectiveness (F-measure) the windowed pq-gram distance outperforms all other distances. The closest competitor is the ordered tree edit distance with sorting (see Section 4). Windowed pq-grams are more effective than pq-grams. By increasing the window size, windowed pq-grams preserve more sibling information (which is relevant to distinguish similar trees) without increasing the children error. For pq-grams the base size q is a tradeoff between preserving more sibling information and increasing the children error (see Section 7). Table 2 shows the effectiveness of pq-grams for the best parameter setting that we found.

The other distance algorithms perform only slightly better or even worse than the simple node intersection distance. The tree-embedding produces many snippets that are single nodes and thus gives less weight to the tree structure than windowed pq-grams [4]. Similarly, the binary branch snippets do not store the edges between a parent and its children (except the edge to the first child), leading to poor performance when many nodes in the trees have identical labels. Embedded pivots give a disproportionate weight to the root label: a quadratic number of snippets is produced from the root, while only a linear number is produced for each leaf [34]. The root label is the street name, thus address trees with different street names are unlikely to be paired. This effect is even more pronounced for path shingles, where all snippets contain the root node.

We also tested variants of embedded pivots which store the level difference between the nodes in the snippet or select a subset of all pivots for which the level difference is within a threshold (1, 2, or 3) [34]. For single path shingles we tested shingles of size larger than one. In Table 2 we only show the results for the winning variants.

9.4 Windowed pq-Grams vs. Sorted Tree Edit Distance

The final experiment compares the approximation quality of the ordered tree edit distance with sorting and the windowed pq-gram distance. Both distances approximate the unordered tree edit distance. We apply edit operations on permuted trees to simulate the unordered tree edit distance as follows.

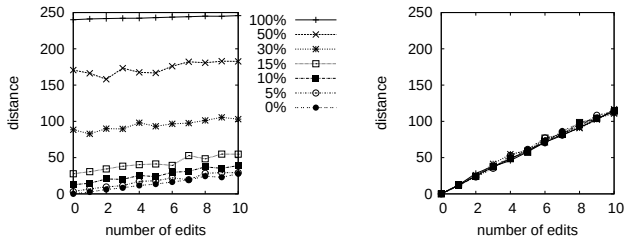
1. A complete tree T with a depth of 5 and a fanout from 2 to 5 is synthetically generated.
2. n edit operations (insert, delete, or rename) are applied to random nodes of T, yielding T'. Every node in the tree has an equal probability of being edited, and after each edit operation the tree is permuted randomly.
3. T and T' are sorted.
4. The ordered tree edit distance and the windowed pq-grams distance between the sorted trees T and T' are computed.
5. This process is repeated 1000 times for different trees T. Each data point is the average over all runs.

In the experiment, we vary the number of edits (from 0 to 9) and the percentage of siblings that have the same label (0%, 5%, 10%, 15%, 30%, 50%, and 100%), where 0% means every sibling has a unique label, and 100% means that all siblings have the same label. The percentage of duplicate sibling labels is relevant in XML. For instance, all `track` nodes in the music albums in Figure 1 have identical labels. A good approximation of the unordered tree edit distance should be linear in the number of edits and invariant to the percentage of identical labels. The distance between identical trees (zero edits) should be zero.

Figure 22 shows the side-by-side results for (a) the sorted tree algorithm, and (b) the windowed pq-grams algorithm ($p = 1$, $q = 2$, $w = 3$). The approximation quality of the sorted tree edit distance drastically decreases if siblings have the same label. This is evident from the distance values between identical trees (zero edits): already for 5% identical labels the average distance is 2.3, for 10% it is 13.8, for 50% it grows to 167. For higher percentages of identical labels the sorted tree edit distance is almost constant and independent of the number of edits. By contrast, the windowed pq-gram

Distance	Correct	Recall	Precision	F-Measure	Runtime
windowed pq-grams ($w=8, p=2, q=2$)	248	82.9%	98.4%	0.900	24.9 s
windowed pq-grams ($w=5, p=1, q=2$)	245	81.9%	98.8%	0.896	12.7 s
windowed pq-grams ($w=3, p=2, q=2$)	240	80.3%	98.8%	0.886	7.4 s
ordered tree edit distance	247	82.6%	96.5%	0.890	669.0s
pq-grams ($p=3, q=2$)	237	79.3%	99.2%	0.881	4.4s
tree-embedding	206	68.9%	96.3%	0.803	6.9s
node intersection	197	65.9%	93.8%	0.774	2.2s
binary branch	193	64.5%	93.2%	0.763	7.1s
embedded pivots	165	55.1%	98.2%	0.707	344.4s
path shingles	73	24.4%	98.6%	0.391	7.8s

Table 2 Effectiveness of Different Tree Distances for Matching Address Trees.



(a) Sorted Tree Edit Distance (b) Windowed pq-Gram Distance

Fig. 22 Sorted Tree Edit Distance vs. Windowed pq-Gram Distance.

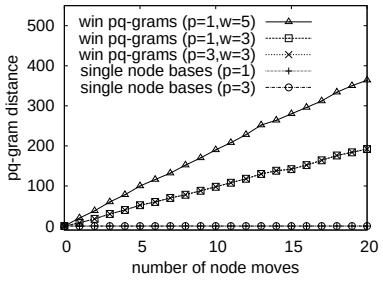


Fig. 23 Single Leaf Bases vs. Windowed pq-Gram Bases.

distance is nearly linear in the number of edits, and the distance between identical trees is always zero. The windowed pq-gram distance is only marginally affected by the percentage of identical siblings.

9.5 Effectiveness of pq-Gram Bases

Single Leaf Bases We compare the effectiveness of bases with a single leaf node to windowed pq-gram bases for the task of detecting node moves (cf. Section 7.3). The XML tree used in this experiment is the DBLP entry for the proceedings of a conference with 142 articles and a total of 527 authors (including duplicates). The authors are children of articles, which are children of the root node. We introduce errors by randomly moving author nodes between articles and compute the distance between the original and the modified version of the proceedings. The results for up to 20 authors that are assigned to the wrong article are shown in Figure 23. The distance for pq-grams with a single leaf node ($q = 1$) is zero, independent of the number of moved author nodes, i.e., the node moves are not detected. Windowed pq-grams detect the node moves and the distance increases with each moved node. The results are independent of the stem size since the ancestors of the author nodes before and after the move have the same labels.

Ordered pq-Gram Bases We compare the effectiveness of ordered (cf. Section 5) and windowed pq-grams on a pair of XML document trees, T and T', with recursively nested sections and paragraphs. Document T has three top-level sections, each section has a title node and three recursively nested section nodes; the children of the sections at the deepest nesting level (depth 4) are paragraphs instead of sections. The node labels are (tag,value)-pairs. The value of a section element is the text that appears before the subsections, the value of a paragraph element is the text of that paragraph. Document T' differs from T only in the tag name of the section title, which in T' is `heading` instead of `title`.

In our experiment, we randomly change the text values of different section nodes. Figure 24 shows ordered and windowed pq-gram distance between the sorted trees T and T' for up to 20 edits. A horizontal line means that the distance does not increase after the change, i.e., the change is not detected. Ordered pq-grams miss all changes. They can not detect changes in the section text since all pq-grams that contain the changed node also contain a title/heading node, which does not match between T and T'. Windowed pq-grams miss only two changes. The good performance of windowed pq-grams is explained by the use of the window to form the bases: although all windows contains a title/heading node, only a subset of the bases ($q = 2$) formed from a window contain these nodes.

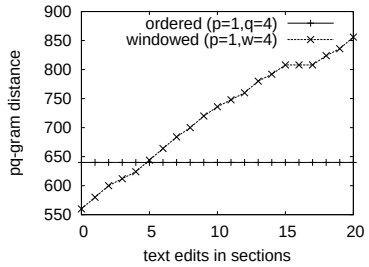


Fig. 24 Ordered vs. Windowed pq-Gram Bases.

10 Conclusion

When XML data from different sources is integrated in a single data collection, data items that represent the same real world object must be recognized. Exact matches, however, often fail in such applications (elements may be missing in one database, content values may not match due to different coding conventions and spelling mistakes, and the data may be arranged in a different structure) and approximate matching techniques must be applied.

Previous research developed approximate join operations based on ordered tree matching, but for data-centric XML applications the order of siblings is irrelevant, and unordered tree matching techniques must be applied. In this paper we propose an approximate join technique for data-centric XML based on windowed pq-grams. Windowed pq-grams consist of a stem and a base and are generated in a three-step process: sort the tree, extend the sorted tree with dummy nodes, and decomposing the extended tree into windowed pq-grams. While the stems are invariant to order, the main challenge is to compute the bases such that the sibling order is ignored. pq-Grams (and the bases thereof) enjoy the following important properties: each edge appears in the same number of pq-grams, the Jaccard distance between children is preserved, and the combined bases of two nodes are different if a child from one node is moved to become a child of the other node.

We show that the permutation of a constant number of siblings changes only a constant number of windowed pq-grams. This core feature allows us to use windowed pq-grams together with tree sorting to approximate the unordered tree edit distance.

We provide an efficient algorithm for the approximate join of data-centric XML. In data-centric XML, the distance algorithm can not take advantage of a predefined document order. The join is implemented as an equality join on windowed pq-grams and permits standard join optimization techniques. Extensive experiments on both synthetic and real world data confirm the analytic results and show that our technique is both useful and scalable.

Future work includes the investigation of persistent, updatable index structures for the windowed pq-gram join. As windowed pq-grams store local information, a document modification (e.g., an altered text value) affects only a limited number of windowed pq-grams. The index should be updated incrementally by substituting the affected pq-grams only, thus avoiding the recomputation of all windowed pq-grams from scratch. A similar approach has been taken by Augsten et al. [5] for ordered pq-grams.

Further we plan to combine our approximate join on data-centric XML with approximate string matching techniques. The string values of some elements or attributes may be particularly important to identify a data item, for example, the title of an article is very significant in a XML database about publications. We would like to include both the similarity of the XML structure and the similarity of selected string values into our approximate join.

Acknowledgments

The work has been done in the framework of the project *eBZ – Digital City*, which is funded by the Municipality of Bolzano-Bozen.

References

1. Sihem Amer-Yahia, Laks V. S. Lakshmanan, and Shashank Pandit. FleXPath: Flexible structure and full-text querying for XML. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 83–94, 2004.
2. Kiyoko F. Aoki, Atsuko Yamaguchi, Nobuhisa Ueda, Tatsuya Akutsu, Hiroshi Mamitsuka, Susumu Goto, and Minoru Kanehisa. KCaM (KEGG carbohydrate matcher): a software tool for analyzing the structures of carbohydrate sugar chains. *Nucleic Acids Research*, 32:267–272, July 2004.
3. Nikolaus Augsten, Michael Böhlen, Curtis Dyreson, and Johann Gamper. Approximate joins for data-centric XML. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 814–823, Cancún, Mexico, April 2008.
4. Nikolaus Augsten, Michael Böhlen, and Johann Gamper. Approximate matching of hierarchical data using pq-grams. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 301–312, Trondheim, Norway, September 2005.
5. Nikolaus Augsten, Michael Böhlen, and Johann Gamper. An incrementally maintainable index for approximate lookups in hierarchical data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 247–258, Seoul, Korea, September 2006.
6. Nikolaus Augsten, Michael Böhlen, and Johann Gamper. The pq-gram distance between ordered labeled trees. *ACM Transactions on Database Systems (TODS)*, 35(1), 2010.
7. Nicolas Bruno, Nick Koudas, and Divesh Srivastava. Holistic twig joins: Optimal XML pattern matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 310–321, Madison, Wisconsin, June 2002.
8. David Buttler. A short survey of document structure similarity algorithms. In *Proceedings of the International Conference on Internet Computing*, pages 3–9, Las Vegas, Nevada, USA, June 2004.

9. Sudarshan S. Chawathe and Hector Garcia-Molina. Meaningful change detection in structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 26–37, Tucson, Arizona, United States, May 1997.
10. Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 493–504, Montreal, Canada, June 1996.
11. Weimin Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40(2):135–158, August 2001.
12. Grégory Cobéna, Serge Abiteboul, and Amélie Marian. Detecting changes in XML documents. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 41–52, San Jose, California, 2002.
13. Theodore Dalamagas, Tao Cheng, Klaas-Jan Winkel, and Timos Sellis. A methodology for clustering XML documents by structure. *Information Systems*, 31(3):187–228, May 2006.
14. Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)*, volume 4596 of *LNCS*, pages 146–157, Wroclaw, Poland, July 2007.
15. Slawomir Duszynski, Jens Knodel, Matthias Naab, Dirk Hein, and Clemens Schitter. Variant comparison—a technique for visualizing software variants. In *Working Conference on Reverse Engineering*, pages 229–233, Antwerp, Belgium, 2008.
16. Sergio Flesca, Giuseppe Manco, Elio Masciari, Luigi Pontieri, and Andrea Pugliese. Fast detection of XML structural similarity. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 17(2):160–175, February 2005.
17. Minos Garofalakis and Amit Kumar. Correlating XML data streams using tree-edit distance embeddings. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2003)*, pages 143–154, San Diego, California, June 2003.
18. Minos Garofalakis and Amit Kumar. XML stream processing using tree-edit distance embeddings. *ACM Transactions on Database Systems (TODS)*, 30(1):279–332, 2005.
19. Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. Approximate XML joins. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 287–298, Madison, Wisconsin, 2002.
20. Sven Helmer. Measuring the structural similarity of semistructured documents using entropy. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 1022–1032, Vienna, Austria, September 2007.
21. Yair Horesh, Ramit Mehr, and Ron Unger. Designing an A* algorithm for calculating edit distance between rooted-unordered trees. *Journal of Computational Biology*, 13(6):1165–1176, 2006.
22. Haifeng Jiang, Wei Wang, Hongjun Lu, and Jeffrey Xu Yu. Holistic twig joins on indexed XML documents. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 273–284, Berlin, Germany, September 2003.
23. Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, March 1987.
24. Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th European Symposium on Algorithms*, volume 1461 of *LNCS*, pages 91–102, Venice, Italy, 1998.
25. Hans-Peter Kriegel and Stefan Schönauer. Similarity search in structured data. In *Data Warehousing and Knowledge Discovery (DaWaK)*, pages 309–319, 2003.
26. Kyong-Ho Lee, Yoon-Chul Choy, and Sung-Bae Cho. An efficient algorithm to compute differences between structured documents. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(8):965–979, August 2004.
27. Andrew Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)*, pages 61–66, Madison, Wisconsin, USA, June 2002.
28. Sven Puhlmann, Melanie Weis, and Felix Naumann. XML duplicate detection using sorted neighborhoods. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, volume 3896 of *LNCS*, pages 773–791, Munich, Germany, March 2006.
29. Leonardo Ribeiro and Theo Härder. Evaluating performance and quality of XML-based similarity joins. In *Advances in Databases and Information Systems (ADBIS)*, volume 5207 of *LNCS*, pages 246–261, Pori, Finland, 2008.
30. Leonardo A. Ribeiro, Theo Härder, and Fernanda S. Pimenta. A cluster-based approach to XML similarity joins. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS)*, pages 182–193, Cetraro, Calabria, Italy, 2009.
31. Ismael Sanz, Marco Mesiti, Giovanna Guerrini, and Rafael Berlanga. Fragment-based approximate retrieval in highly heterogeneous XML collections. *Data & Knowledge Engineering*, 64(1):266–293, January 2008.
32. Sunita Sarawagi and Alok Kirpal. Efficient set joins on similarity predicates. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 743–754, 2004.
33. Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the ACM (JACM)*, 26(3):422–433, July 1979.
34. Shirish Tatikonda and Srinivasan Parthasarathy. Hashing tree-structured data: Methods and applications. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 429–440, Long Beach, CA, USA, March 2010.
35. Joe Tekli, Richard Chbeir, and Kokou Yétongnon. An overview on XML similarity: Background, current trends and future directions. *Computer Science Review*, 3(3):151–173, 2009.
36. Esko Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, January 1992.
37. C. J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2nd edition, March 1979.
38. Yuan Wang, David J. DeWitt, and Jin-yi Cai. X-Diff: An effective change detection algorithm for XML documents. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 519–530, Bangalore, India, March 2003.
39. Melanie Weis and Felix Naumann. DogmatiX tracks down duplicates in XML. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 431–442, Baltimore, Maryland, USA, June 2005.
40. Rui Yang, Panos Kalnis, and Anthony K. H. Tung. Similarity evaluation on tree-structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 754–765, Baltimore, Maryland, USA, June 2005.
41. Peter N. Yianilos. Normalized forms for two common metrics. Technical report, NEC Research Institute, 1991, 2002.
42. Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search—The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
43. Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.
44. Kaizhong Zhang, Richard Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.