The *pq*-Gram Distance between Ordered Labeled Trees

NIKOLAUS AUGSTEN, MICHAEL BÖHLEN, and JOHANN GAMPER
Free University of Bozen-Bolzano

When integrating data from autonomous sources, exact matches of data items that represent the same real world object often fail due to a lack of common keys. Yet in many cases structural information is available and can be used to match such data. Typically the matching must be approximate since the representations in the sources differ.

We propose *pq*-grams to approximately match hierarchical data from autonomous sources and define the *pq*-gram distance between ordered labeled trees as an effective and efficient approximation of the fanout weighted tree edit distance. We prove that the *pq*-gram distance is a lower bound of the fanout weighted tree edit distance and give a normalization of the *pq*-gram distance for which the triangle inequality holds. Experiments on synthetic and real world data (residential addresses and XML) confirm the scalability of our approach and show the effectiveness of *pq*-grams.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—relational databases, query processing; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—indexing methods; G.2.2 [**Discrete Mathematics**]: Graph Theory—trees

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: database algorithms, similarity search, approximate matching, hierarchical data, XML, tree edit distance, distance metric

1. INTRODUCTION

When integrating data from different sources exact matches of data items that represent the same real world object often fail due to missing global keys and different data representations. Approximate matching techniques must be applied instead.

Often the data items that need to be matched are part of a hierarchical structure that can be represented as ordered labeled trees. Thus, the approximate matching of data items can exploit the similarity of trees. A well-known but computationally expensive measure for comparing trees is the tree edit distance, which is defined as the minimum cost sequence of edit operations (node insertion, node deletion, and rename) that transforms one tree into another [Tai 1979]. Zhang and Shasha [Zhang and Shasha 1989] present an algorithm to compute the tree edit distance with a worst case complexity of $O(n^4)$, where n is the number of nodes. Later improve-

Authors' addresses: N. Augsten, M. Böhlen, J. Gamper, Faculty of Computer Science, Free University of Bozen-Bolzano, Italy; email: {augsten,boehlen,gamper}@inf.unibz.it

ments of their algorithm [Klein 1998; Demaine et al. 2007] still need at least $O(n^3)$ time and $O(n^2)$ space and do not scale to large trees.

In this paper we propose a new distance measure, the *pq-gram distance*, to approximately match ordered labeled trees. Intuitively, the *pq*-grams of a tree are all its subtrees of a specific shape. Two trees are similar if they have many *pq*-grams in common. By adjusting the two parameters p and q, which specify the shape of the *pq*-grams, the new distance measure allows to control the relevance of the tree structure.

For two trees of size n, the *pq*-gram distance can be computed in $O(n \log n)$ time and $O(n)$ space. We show that the new distance is an effective and efficient approximation of the fanout weighted tree edit distance, which assigns to each edit operation a cost proportional to the fanout of the modified nodes. Thus, an edit operation on a node with few children, e.g., a leaf, is cheaper than an edit operation on a node with many children. We prove that the *pq*-gram distance provides a lower bound for the fanout weighted tree edit distance.

To summarize, the main constributions of this paper are the following:

—We present the *pq*-gram distance as a new approximation of the fanout weighted tree edit distance and give a lower bound guarantee for the approximation. We show that our normalization of the *pq*-gram distance satisfies the triangle inequality.

—We present an algorithm and its relational implementation to compute the *pq*-gram distance in $O(n \log n)$ time and $O(n)$ space, where n is the number of tree nodes.

—We report extensive experimental studies with various real-world data sets (address data from our local municipality, DBLP, SwissProt, and Treebank), which confirm the analytical results: the *pq*-gram distance is competitive both in terms of quality and efficiency with other tree edit distance approximations, and the *pq*-gram distance is scalable to large data sets.

The rest of the paper is organized as follows. In Section 2 we describe an application scenario at our local municipality. In Section 3 we discuss related work. Section 4 gives preliminaries, and Section 5 defines the fanout weighted tree edit distance. We define the *pq*-gram distance in Section 6 and show that it provides a lower bound for the fanout weighted tree edit distance in Section 7. In Section 8 we give an algorithm for the computation of the *pq*-grams, analyze the complexity of this algorithm, and discuss its implementation in a relational database. In Section 9 we evaluate the efficiency and effectiveness of our method on synthetic and real world data and compare it to other approximations. We draw conclusions and point to future work in Section 10.

2. APPLICATION SCENARIO

Many public administrations own or have access to multiple autonomous databases with geographical data, i.e., data associated with address information. Common examples of geographical data include databases with residential information, databases with property information, and databases with tax information. An increasingly important need for public administrations is to automate the linking and merging of geographical data from autonomous databases. Below we describe two applications from the Municipality of Bozen-Bolzano. Both applications need to match residential addresses from different databases.

2.1 Street Matching

To link geographical data from autonomous databases we first establish a matching between streets. If a central registry for streets is being used this is simple. Without a central registry the street names can be compared lexicographically. For autonomous databases the comparison of street names yields poor results since street names differ due to the use of multiple languages, spelling mistakes, different naming conventions, and renamed streets that are not updated in all databases. Figure 1 illustrates the problem in terms of two example tables with street names and addresses from the municipality.

`Ownr`

*s*1	*n*1	*e*1	*a*1	*resident*
Cimitero	4	-	-	Rose
Cimitero	6	-	-	Lara
Friedensplatz	2	A	1	Igor
Friedensplatz	2	A	2	Sarah
Friedensplatz	3	-	-	Sue
Mariengasse	1	A	-	Linas
Rentscherstr	1	-	-	Pia
Rentscherstr	2	-	-	Tony
Trienterstr	1	A	1	Tom
Trienterstr	1	A	2	Tom
Trienterstr	1	A	3	Pam
Via Bivio	1	-	1	Peter
Via Bivio	1	-	3	John
Via Bivio	2	A	-	Marc
...				

`Rsdt`

*s*2	*n*2	*e*2	*a*2	*owner*
Friedhofplatz	4	-	-	Dario
Friedhofplatz	6	A	-	Luigi
Friedhofplatz	6	B	-	Marc
Kaiserau	1	-	1	Peter
Kaiserau	1	-	2	Lena
Kaiserau	1	-	3	Lena
Kaiserau	2	A	-	Anita
Mariengasse	1	-	-	Adam
Reschenstr	1	A	1	Tony
Triesterstr	1	-	-	Ron
Siegesplatz	2	A	1	Martin
Siegesplatz	3	-	1	Leo
Siegesplatz	3	-	2	Maria
Siegesplatz	3	-	3	Rosa
...				

Fig. 1. Tables with Owner (`Ownr`) and Resident (`Rsdt`) Information.

The `Ownr` table stores the owner and the `Rsdt` table the resident of each property. Clearly, matching streets from `Ownr` and `Rsdt` is non-trivial. For example, `Cimitero` and `Friedhofsplatz`, `Via Bivio` and `Kaiserau`, and `Friedensplatz` and `Siegesplatz` are matching pairs of streets. In all cases string comparison performs poorly. Besides streets that match there are also streets that are indeed different and may not be matched. For example, `Trienterstr` and `Triesterstr` are different streets as are `Rentscherstrasse` and `Reschenstrasse`. In both cases the street names are similar and a string comparison might wrongly match these street pairs.

To reliably match streets it is necessary to exploit the information about the structure of streets. Towards this end the addresses of single streets can be organized hierarchically and represented as *address trees* [Augsten et al. 2004]. Figure 2 shows the address trees for streets `Via Bivio` and `Kaiserau` in Figure 1. The root of the tree is the street name, the children of the street name are the house numbers, the children of house numbers are the entrance numbers, and the children of entrance numbers are the apartment numbers. We omit empty values ("-") in the leaves of address trees. A complete address is the path from the root to any leaf node. For example, the tuple (`Via Bivio`, `2`, `A`, `-`) of table `Ownr` represents the address `Via Bivio 2A` and corresponds to the shaded path in Figure 2.

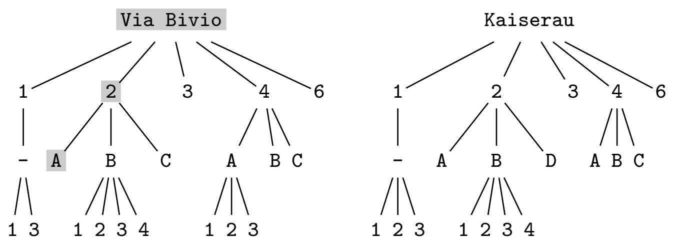


Fig. 2. Address Trees of Streets `Via Bivio` from `Ownr` and `Kaiserau` from `Rsdt`.

To match streets from different databases we compare the structure of their address trees. Intuitively, two streets match if they have (almost) the same address tree. We use this idea to replace the match on the street names by an approximate match on the structure of a street.

Below, we use the street matching to solve two concrete problems in the context of geographic databases. Our solution is expressed as a sequence of algebraic expressions. The aggregation operator $\Gamma[G, A]$ takes two arguments: a list of grouping attributes G and a list of aggregates A. $\mathbf{T}(s, n, e, a)$ is an aggregate that constructs an address tree with street name s as root, the street number n at level one, the entry number e at the second level, and the apartment number a at the third level (cf. Figure 2). $\text{dist}(\mathbf{T}_1, \mathbf{T}_2)$ is the distance between trees $\mathbf{T}_1$ and $\mathbf{T}_2$. τ is a distance threshold. We write $e \rightarrow a$ to rename attribute e to a.

2.2 Application 1: Matching Owners and Residents

As a first application, suppose we need to *match up owners and residents* to check the property taxes. Clearly, there exists a relationship between owners and residents. However, if no explicit information about this relationship is stored the only connection between owners and residents is the location: the owner is the person who owns the property with the address at which the resident lives.

In order to match owners and residents we first establish a matching between the streets. This is done by constructing the address trees for the streets in both databases followed by an approximate address tree join. The final step is to join owners and residents using the street matching. For this step we must take into consideration that owner and resident information may not be stored at the same granularity. For instance an owner might own an entire house but let the apartments in the house to different residents. The following algebraic expression uses the tables `Ownr` and `Rsdt` of Figure 1 and implements this approach:

$$
\begin{aligned}
O &= \Gamma[s1, \mathbf{T}(s1, n1, e1, a1) \rightarrow T1] Ownr \\
R &= \Gamma[s2, \mathbf{T}(s2, n2, e2, a2) \rightarrow T2] Rsdt \\
Y &= \pi[s1 \rightarrow u, s2 \rightarrow v](O \bowtie_{dist(T1,T2)<\tau} R) \\
X &= \sigma[n1 \sim n2 \wedge e1 \sim e2 \wedge a1 \sim a2](Ownr \bowtie_{s1=u} Y \bowtie_{v=s2} Rsdt)
\end{aligned}
$$

The first two statements construct the address trees by aggregating the addresses within a street. The third statement is the approximate tree join that pairs streets with a similar structure. The final statement uses the street matching to match addresses within streets. Two addresses match if house, entrance, and apartment numbers are pairwise identical. Numbers are also considered identical if one is empty. Thus, in SQL terminology $n1 \sim n2$ is expressed as `n1=n2 OR n1 IS NULL OR n2 IS NULL`.

2.3 Application 2: Master Address Table

As a second application, consider the generation of a *master address table* that contains all addresses. Producing a master address table is subtle and most public administrations are not able to provide a consolidated master address table. In order to produce a master address table they must merge addresses from different sources, recognize addresses that refer to the same location, and retain exactly one of these addresses.

In order to construct a master address table with all addresses two key problems must be solved. First, we must detect duplicates. We identify two streets as duplicates if their address trees are very similar. Second, we must decide which address tree to retain. We assume that we want the street with data at the most detailed level. Thus, if two streets are similar we keep the larger one, i.e., the one with more addresses. This solution is implemented by the following algebraic expressions:

$$\begin{aligned}
O &= \Gamma[s1 \to s, \mathbf{T}(s1, n1, e1, a1) \to T] Ownr \\
R &= \Gamma[s2 \to s, \mathbf{T}(s2, n2, e2, a2) \to T] Rsdt \\
Adr &= O \cup R \\
Y &= \pi[A.s \to u, B.s \to v, A.T \to C, B.T \to D](Adr \to A \bowtie_{dist(A.T,B.T)<\tau} Adr \to B) \\
Dupl &= \pi[u]\sigma[size(C) < size(D) \vee size(C) = size(D) \wedge u < v]Y \\
X &= \pi[u]Adr \setminus Dupl
\end{aligned}$$

The first two statements construct the address trees. The third statement collects all addresses. The fourth statement is a self join, which determines all pairs of streets with similar address trees. The fifth statement selects all streets that have a larger street that is very similar (streets of the same size are distinguished by their name). These are the duplicated streets we are not interested in. The last statement uses a set difference to return all streets except the duplicated ones. We assume a duplicate eliminating projection (`SELECT DISTINCT` in SQL terms).

2.4 Lessons Learned and Generalization

Besides the key aspects of our solution described above we summarize additional observations and lessons learned from deploying our system.

—In autonomous databases with hierarchical data different levels of detail can (and should!) exist. For instance an owner who owns a house might let apartments to many residents (e.g., the owner of `Cimitero 6` has let apartments to residents in `Friedhofsplatz 6A` and `Friedhofsplatz 6B`).

—The matching of addresses can be refined according to the semantics of the application. Permitting null values when comparing house, entry, and apartment numbers works well for a relational representation of addresses where different levels of details exist and null values are used for non-existing levels of detail.

—Structural information is used to encourage and discourage matching, respectively. For instance `Via Bivio` and `Kaiserau` should be matched although the street names are very different. On the other hand, `Trienterstr` and `Triesterstr` may not be matched although the street names are very similar.

—We assume that the street names in different databases are disjoint. This prevents that streets are merged when, e.g., doing a union of the addresses of residents and owners. If this is not the case the street name can be concatenated with a database identifier.

—The tables in Figure 1 are denormalized. With normalized tables the street name would be stored in a separate table and an additional join must be used to construct `Ownr` and `Rsdt` tables.

In this section we have discussed two applications of the Municipality of Bozen-Bolzano that are based on addresses. Note though that instead of addresses we can easily use article trees from DBLP, proteins from Swissprot, etc. All this data can be represented as trees. In many cases the tree representation is, in fact, the default one, e.g., the XML representation of DBLP. Apart from the discussion in this section our solutions and descriptions are kept general and apply to ordered labeled trees in general. No application specific assumptions are made. In the experimental evaluation we evaluate our solution on various hierarchical data sets: address data, DBLP, Swissprot and Treebank.

3. RELATED WORK

A well known distance function for trees is the *tree edit distance*, which is defined as the minimum cost sequence of edit operations (node insertion, node deletion, and rename) that transforms one tree into another [Tai 1979]. Zhang and Shasha [Zhang and Shasha 1989] present an algorithm to compute the tree edit distance in $O(n^2 \min^2(l,h))$ time and $O(n^2)$ space for trees with n nodes, l leaves, and height h. Since in a tree both l and h may be of size $O(n)$, the worst case complexity of this algorithm is $O(n^4)$. Later works have improved the worst case complexity of the tree edit distance algorithm [Klein 1998; Chen 2001; Demaine et al. 2007], but all of them use at least $O(n^3)$ time and $O(n^2)$ space and do not scale to large trees.

By imposing restrictions on the edit operations that can be applied to transform a tree, *suboptimal solutions* with better runtime complexities can be found: the alignment distance [Jiang et al. 1995], the constrained edit distance [Zhang 1995; Guha et al. 2002], and the top-down distance [Selkow 1977; Yang 1991] all need at least $O(n^2)$ time. The bottom-up distance [Valiente 2001] can be computed in $O(n)$ time. It tries to find the largest possible common subtrees of two trees, starting with the leaf nodes. This renders the distance sensitive to differences of leaf nodes: if the leaf nodes are different the inner nodes are never compared.

Chawathe et al. [Chawathe et al. 1996] use a variant of the tree edit distance for *change detection* in hierarchical data. Lee et al. [Lee et al. 2004] tune the algorithm presented by Chawathe et al. to XML documents that are represented as ordered labeled trees. Both algorithms first compute a match between the nodes of the trees, and based on the match the distance is computed in $O(ne)$ time, where e is the edit distance between the trees. In a change detection scenario typically trees with small differences are compared, but for approximate joins the distances between all pairs of trees must be computed. For trees that are very different the edit distance e is $O(n)$, which yields $O(n^2)$ runtime for both algorithms. The change detection algorithm by Cobéna et al. [Cobéna et al. 2002] takes advantage of existing element IDs, which can not be assumed for data from different sources.

Guha et al. [Guha et al. 2002] present a framework for *approximate joins* between XML documents based on a distance metric, for example, the tree edit distance. They use reference sets to take advantage of the triangle inequality of a metric, thus reducing the actual number of distance computations in a join. They give upper and lower bounds for the tree edit distance that can be computed in $O(n^2)$ time. The *pq*-gram distance provides a lower bound for the fanout weighted tree edit distance and can be computed in $O(n \log n)$ time.

Garofalakis and Kumar [Garofalakis and Kumar 2005] investigate an algorithm for *embedding the tree edit distance* with subtree move as an additional edit operation into a numeric vector space equipped with the standard L_1 distance norm. The algorithm runs in $O(n \times log^* n)$ time and $O(n)$ space and computes an approximation of the tree edit distance with subtree move to within an $O(log^2 n \times log^* n)$ factor.[1] No lower bound guarantee is given. We implement this approximation and empirically compare it to the *pq*-gram distance. The tree embedding distance assumes a unit cost model for edit operations and typically gives less weight to structural changes than the fanout weighted tree edit distance [Augsten et al. 2005].

The *binary branch distance* [Yang et al. 2005] was introduced as a lower bound for the tree edit distance with a unit cost model. The trees are split into small subtrees, the binary branches. Two trees are similar if they have many binary branches in common. Each tree node produces exactly one binary branch that consist of the node itself, its right sibling, and its first child. The edges to the other children are not stored. These edges are essential to distinguish trees with different structure that have similar sibling sequences. An example are the address trees introduced in Section 2, where the sibling sequence $(1, 2, 3, 4)$ frequently appears. Depending on the parent node the sequence represents either house numbers or apartment numbers of different houses. *pq*-Grams explicitly store both parent-child edges and sibling relationships. We implement the binary branch distance and evaluate its effectiveness for approximate joins.

pq-Grams were introduced by Augsten et al. [Augsten et al. 2005] to compute the distance between ordered labeled trees. Their empirical results suggest that *pq*-grams are a good approximation of the tree edit distance. We extend this work with real world experiments on XML data and we prove that the *pq*-gram distance provides a lower bound for the fanout weighted tree edit distance. Moreover, the normalization used by Augsten et al. is not a metric, in particular, it does not

[1] $\log^* n$ denotes the number of log applications required to reduce n to a quantity that is ≤ 1.

satisfy the triangle inequality. We provide a normalization that satisfies the triangle inequality. Windowed *pq*-grams [Augsten et al. 2008] approximate the tree edit distance between unordered trees where the sibling order is not relevant. We assume ordered trees where the sibling order matters. Ribeiro and Härder [Ribeiro and Härder 2008] introduce extended *pq*-grams. Extended *pq*-grams consider in addition to the structural similarity between trees also the string similarity between leaf nodes.

Ohkura et al. [Ohkura et al. 2005] propose *q*-grams for ordered *unlabeled* trees. A *q*-gram is a subtree with q nodes that either is a vertical node chain or has a root node with fanout two and two leaves. There are $q-1$ different *q*-gram shapes. A tree is represented by a *q*-gram vector with $q-1$ dimensions, where each dimension counts the *q*-grams of a specific shape. The *q*-gram distance between two trees is the L_1 distance between their *q*-gram vectors. The authors propose a linear time algorithm to compute the *q*-grams. No experimental results are given. We propose *pq*-grams for ordered *labeled* trees and evaluate *pq*-grams on real world datasets.

Weis and Naumann [Weis and Naumann 2005] propose an XML similarity measure for a *duplicate detection* framework. In the worst case, all pairs of elements must be compared. Puhlmann et al. [Puhlmann et al. 2006] 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. Sanz et al. [Sanz et al. 2008] 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. Their solution is designed for highly heterogeneous XML collections where both hierarchy and sibling order may differ between pattern and matched region.

A core operation in XML query processing is to find all occurrences of a *twig pattern* [Bruno et al. 2002; Jiang et al. 2003]. The goal of our work is not to find occurrences of a pattern to answer queries. We split the tree into subtrees in order to calculate the distance between trees. Polyzotis et al. [Polyzotis et al. 2004] build *synopsis* of an XML tree optimized for approximate query answering. They introduce the element simulation distance to capture the difference between the original tree and the synopsis with respect to twig queries. This distance is tailored to measure the quality of a synopsis and is not suitable as an approximation for the tree edit distance. Several papers deal with the related, but different problem of detecting the *structural similarity* between XML documents [Nierman and Jagadish 2002; Flesca et al. 2005; Dalamagas et al. 2006; Helmer 2007]. Two documents are considered structurally similar if they are valid for a similar DTD. The text content of the elements and the values of the attributes are ignored.

Navarro [Navarro 2001] gives a good overview of the edit distance for *strings* and its variants. Ukkonen [Ukkonen 1992] introduces the *q*-gram distance as a lower bound for the string edit distance. The *q*-gram distance between two strings is based on the number of common substrings of length q. Gravano et al. [Gravano et al. 2001] present algorithms for approximate string joins based on the edit distance and use *q*-grams as a filtering algorithm. Approximate string matching techniques are successful if the distance between corresponding strings is smaller than that of other strings in the join set. This is typically the case for spelling mistakes, where only a few characters change. The distance between corresponding street names, however, is often larger than the length of the shorter string. If streets are renamed or the street names are spelled in different languages, string matching fails completely.

4. PRELIMINARIES

A *tree* $\mathbf{T}$ is a directed, acyclic, connected, non-empty graph with *nodes* $N(\mathbf{T})$ and edges $E(\mathbf{T})$. An *edge* is an ordered pair (p, c), where $\mathsf{p}, \mathsf{c} \in N(\mathbf{T})$ are nodes, and p is the *parent* of c. A node can have at most one parent, and nodes with the same parent are *siblings*. A total order $<$ is defined on each group of sibling nodes. Two siblings $\mathsf{s}_1, \mathsf{s}_2$ are *contiguous* iff $\mathsf{s}_1 < \mathsf{s}_2$ and they have no sibling x such that $\mathsf{s}_1 < \mathsf{x} < \mathsf{s}_2$. A node c is the i-th *child* of p iff $i = |\{\mathsf{x} \in N(\mathbf{T}) | (\mathsf{p}, \mathsf{x}) \in E(\mathbf{T}), \mathsf{x} \leq \mathsf{c}\}|$. The number of children of p is its *fanout* f_{p}. The node with no parent is the *root* node, $\mathrm{root}(\mathbf{T})$, and a node without children is a *leaf*. Each node v has a *label*, $\lambda(\mathsf{v}) \in \mathbf{\Sigma}$, where $\mathbf{\Sigma}$ is a finite alphabet. In the sequel, such trees are called *ordered labeled* trees.

The *empty node*, ϵ, does not appear in trees and its label is different from the labels of all other nodes. $N_\epsilon(\mathbf{T}) = N(\mathbf{T}) \cup \{\epsilon\}$ denotes the set of all nodes of $\mathbf{T}$ including the empty node. A node o with the label $\lambda(\mathsf{o}) = *$ is a *dummy node*.

Each node a in the path from the root node to a node v is called an *ancestor* of v. If there is a path of length $k > 0$ from a to v, then a is the ancestor of v at distance k. The parent of a node is its ancestor at distance 1. Node d is a *descendant* of v if v is an ancestor of d. The *level* of a node is the length of the path from the root to v. The *height* of a tree is the length of the longest path from the root to any one of the leaves.

A *subtree* $\mathbf{S}$ of $\mathbf{T}$ is a tree with $N(\mathbf{S}) \subseteq N(\mathbf{T})$ and $E(\mathbf{S}) \subseteq E(\mathbf{T})$, retaining the sibling order. A *preorder traversal* of a tree visits a node before its children (if any) and after its left siblings and their descendants (if any), and each node is visited exactly once. Node v is the i-th node of $\mathbf{T}$ in preorder if v is visited as the i-th node in the preorder traversal.

Each node has a unique identifier, and the identifiers of different trees are disjoint. We use v to refer to both the node as well as its identifier. In our graphical representation of trees we represent nodes as (identifier, label)-pairs, the edges are lines between the nodes, and siblings are ordered from left to right. Whenever possible we omit the identifiers of nodes to avoid clutter (e.g., in Figure 2).

EXAMPLE 4.1. *Figure 3 shows two example trees,* $\mathbf{T}_1$ *and* $\mathbf{T}_2$*. The nodes of* $\mathbf{T}_1$ *are* $N(\mathbf{T}_1) = \{\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4, \mathsf{v}_5, \mathsf{v}_6\}$*, the edges are* $E(\mathbf{T}_1) = \{(\mathsf{v}_1, \mathsf{v}_2), (\mathsf{v}_1, \mathsf{v}_5), (\mathsf{v}_1, \mathsf{v}_6), (\mathsf{v}_2, \mathsf{v}_3), (\mathsf{v}_2, \mathsf{v}_4)\}$*, and the sibling orders are* $\mathsf{v}_2 < \mathsf{v}_5 < \mathsf{v}_6$ *and* $\mathsf{v}_3 < \mathsf{v}_4$*. Node* v_1 *has three children:* v_2 *is the first,* v_5 *the second, and* v_6 *the third child. The root node is* $\mathrm{root}(\mathbf{T}) = \mathsf{v}_1$*, and it is the ancestor of all other nodes.* v_3*,* v_4*,* v_5 *and* v_6 *are leaf nodes. The node labels in* $\mathbf{T}_1$ *are* $\lambda(\mathsf{v}_1) = \mathsf{a}$*,* $\lambda(\mathsf{v}_2) = \mathsf{a}$*,* $\lambda(\mathsf{v}_3) = \mathsf{e}$*,* $\lambda(\mathsf{v}_4) = \mathsf{b}$*,* $\lambda(\mathsf{v}_5) = \mathsf{b}$*, and* $\lambda(\mathsf{v}_6) = \mathsf{c}$*. Tree* $\mathbf{S}_1$ *with nodes* $N(\mathbf{S}_1) = \{\mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4\}$*, edges* $E(\mathbf{S}_1) = \{(\mathsf{v}_2, \mathsf{v}_3), (\mathsf{v}_2, \mathsf{v}_4)\}$ *and sibling order* $\mathsf{v}_3 < \mathsf{v}_4$ *is a subtree of* $\mathbf{T}_1$*. The preorder traversal of* $\mathbf{T}_1$ *visits the nodes in the following order:* $\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4, \mathsf{v}_5, \mathsf{v}_6$.

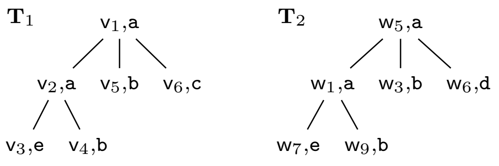


Fig. 3. Two Example Trees $\mathbf{T}_1$ and $\mathbf{T}_2$.

5. THE TREE EDIT DISTANCE

The tree edit distance has emerged as the most widely used distance measure to capture the difference between two ordered labeled trees. Together with a cost model, it sums up the cost of the cheapest sequence of edit operations that transform one tree into the other. In this section we provide a formal definition of the tree edit distance together with a cost model that is sensitive to structural changes.

An edit operation transforms a tree $\mathbf{T}$ into a tree $\mathbf{T}'$. We use the following three standard edit operations on trees: *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. Following previous work on the tree edit distance [Tai 1979; Zhang and Shasha 1989] we formally define the edit operations in terms of edit mappings.

Definition 5.1 *Edit Mapping and Node Alignment.* Let $\mathbf{T}$ and $\mathbf{T}'$ be ordered labeled trees. $M \subseteq N_\epsilon(\mathbf{T}) \times N_\epsilon(\mathbf{T}')$ is an *edit mapping between* $\mathbf{T}$ *and* $\mathbf{T}'$ iff

(1) each node is mapped:
  (a) $\forall \mathsf{v}[\mathsf{v} \in N(\mathbf{T}) \Leftrightarrow \exists \mathsf{v}'((\mathsf{v},\mathsf{v}') \in M)]$
  (b) $\forall \mathsf{v}'[\mathsf{v}' \in N(\mathbf{T}') \Leftrightarrow \exists \mathsf{v}((\mathsf{v},\mathsf{v}') \in M)]$
  (c) $(\epsilon,\epsilon) \notin M$
(2) all pairs of non-empty nodes $(\mathsf{v},\mathsf{v}'), (\mathsf{w},\mathsf{w}') \in M$ satisfy the following conditions:
  (a) $\mathsf{v} = \mathsf{w} \Leftrightarrow \mathsf{v}' = \mathsf{w}'$ (one-to-one condition)
  (b) v is an ancestor of $\mathsf{w} \Leftrightarrow \mathsf{v}'$ is an ancestor of w' (ancestor condition)
  (c) v is to the left[2] of $\mathsf{w} \Leftrightarrow \mathsf{v}'$ is to the left of w' (order condition)

A pair $(\mathsf{v},\mathsf{v}') \in M$ is called a *node alignment.*

Non-empty nodes that are mapped to other non-empty nodes are either renamed or not modified when $\mathbf{T}$ is transformed into $\mathbf{T}'$. Nodes of $\mathbf{T}$ that are mapped to the empty node are deleted from $\mathbf{T}$, and nodes of $\mathbf{T}'$ that are mapped to the empty node are inserted into $\mathbf{T}'$.

Definition 5.2 *Deletions, Insertions, and Renames defined by an Edit Mapping.* Let $M \subseteq N_\epsilon(\mathbf{T}) \times N_\epsilon(\mathbf{T}')$ be an edit mapping between trees $\mathbf{T}$ and $\mathbf{T}'$. Then M defines the following sets of alignments:

—$D(M) = \{(\mathsf{v},\epsilon) \mid (\mathsf{v},\epsilon) \in M\}$ are the *deletions*,
—$I(M) = \{(\epsilon,\mathsf{v}') \mid (\epsilon,\mathsf{v}') \in M\}$ are the *insertions*,
—$R(M) = \{(\mathsf{v},\mathsf{v}') \mid (\mathsf{v},\mathsf{v}') \in M \wedge \lambda(\mathsf{v}) \neq \lambda(\mathsf{v}') \wedge v \neq \epsilon \wedge v' \neq \epsilon\}$ are the *renames.*

[2]A node v is to the left of a node w iff v precedes w in the preorder traversal of the tree and v is not an ancestor of w.

Definition 5.3 *Edit Operation.* $e(\mathbf{T}, \mathsf{v}, \mathsf{v}') = \mathbf{T}'$ is an edit operation that transforms tree $\mathbf{T}$ into tree $\mathbf{T}'$ by either renaming v to v', deleting v, or inserting v' iff there exists an edit mapping M between $\mathbf{T}$ and $\mathbf{T}'$ that satisfies the following condition:

$$\begin{aligned} \textit{Rename: } e(\mathbf{T}, \mathsf{v}, \mathsf{v}') = \mathbf{T}' &\Leftrightarrow \quad R(M) = \{(\mathsf{v}, \mathsf{v}')\}, D(M) = I(M) = \{\} \\ \textit{Delete: } e(\mathbf{T}, \mathsf{v}, \epsilon) = \mathbf{T}' &\Leftrightarrow \quad D(M) = \{(\mathsf{v}, \epsilon)\}, I(M) = R(M) = \{\} \\ \textit{Insert: } e_{\mathsf{p},i,m}(\mathbf{T}, \epsilon, \mathsf{v}') = \mathbf{T}' &\Leftrightarrow \quad I(M) = \{(\epsilon, \mathsf{v}')\}, D(M) = R(M) = \{\} \end{aligned}$$

The insert operation inserts v' as the i-th child of p, and children i to $i + m - 1$ of p become children of v'. Formally, $\mathsf{p} \neq \epsilon \wedge i + m - 1 \leq \mathsf{f}_\mathsf{p} \Leftrightarrow \mathsf{f}_{\mathsf{v}'} = m \wedge \exists \mathsf{p}'[(\mathsf{p}, \mathsf{p}') \in M \wedge \mathsf{v}'$ is i-th child of $\mathsf{p}']$, and $\mathsf{p} = \epsilon \Leftrightarrow \mathsf{v}' = \text{root}(\mathbf{T}')$.

EXAMPLE 5.1. *Consider Figure 4, which illustrates two consecutive edit operations. Renaming node* v_6 *transforms* $\mathbf{T}_1$ *into* $\mathbf{T}_2$. *The edit mapping between* $\mathbf{T}_1$ *and* $\mathbf{T}_2$ *is* $\{(\mathsf{v}_1, \mathsf{w}_5), (\mathsf{v}_2, \mathsf{w}_1), (\mathsf{v}_5, \mathsf{w}_3), (\mathsf{v}_6, \mathsf{w}_6), (\mathsf{v}_3, \mathsf{w}_7), (\mathsf{v}_4, \mathsf{w}_9)\}$. $\lambda(\mathsf{v}_6) = \mathsf{c}$ *and* $\lambda(\mathsf{w}_6) = \mathsf{d}$. *Deleting node* w_1 *in* $\mathbf{T}_2$ *yields tree* $\mathbf{T}_3$. *The edit mapping between* $\mathbf{T}_2$ *and* $\mathbf{T}_3$ *is given as* $\{(\mathsf{w}_5, \mathsf{x}_1), (\mathsf{w}_7, \mathsf{x}_2), (\mathsf{w}_9, \mathsf{x}_3), (\mathsf{w}_3, \mathsf{x}_4), (\mathsf{w}_6, \mathsf{x}_5), (\mathsf{w}_1, \epsilon)\}$. $\mathbf{T}_3$ *is transformed to* $\mathbf{T}_2$ *by an insert,* $\mathbf{T}_2$ *is transformed to* $\mathbf{T}_1$ *by a rename.*

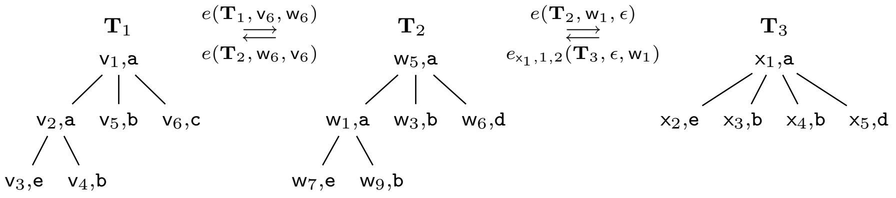


Fig. 4. Two Edit Operations that Transform $\mathbf{T}_1$ into $\mathbf{T}_3$.

In order to determine the distance between trees a cost model must be defined. We assign a cost to each node alignment of an edit mapping. This cost is proportional to the fanout of the aligned nodes.

Definition 5.4 *Cost of a Node Alignment.* Let $\mathbf{T}$ and $\mathbf{T}'$ be ordered labeled trees, let $c > 0$ be a constant, $\mathsf{v} \in N_\epsilon(\mathbf{T})$, $\mathsf{v}' \in N_\epsilon(\mathbf{T}')$. The *cost of a node alignment,* $\gamma(\mathsf{v}, \mathsf{v}')$, is defined as:

$$\gamma(\mathsf{v}, \mathsf{v}') = \begin{cases} \mathsf{f}_\mathsf{v} + c & \text{if } \mathsf{v} \neq \epsilon \wedge \mathsf{v}' = \epsilon \text{ (delete)} \\ \mathsf{f}_{\mathsf{v}'} + c & \text{if } \mathsf{v} = \epsilon \wedge \mathsf{v}' \neq \epsilon \text{ (insert)} \\ \frac{\mathsf{f}_\mathsf{v} + \mathsf{f}_{\mathsf{v}'}}{2} + c & \text{if } \mathsf{v} \neq \epsilon \wedge \mathsf{v}' \neq \epsilon \wedge \lambda(\mathsf{v}) \neq \lambda(\mathsf{v}') \text{ (rename)} \end{cases} \tag{1}$$

Thus, the cost of aligning leaf nodes is a constant c for all edit operations. For the other nodes the cost of deletion and insertion is proportional to the node's fanout. The choice of the rename cost should depend on the costs of deletion and insertion. The maximum meaningful cost of rename is below the cost of deletion plus insertion. If the rename cost is larger, then rename can always be substituted by deletion and insertion at lower cost, and rename is never used in the cheapest edit sequence. If the rename cost is small compared to the costs of insertion and deletion, then the cheapest edit sequence optimizes on aligning the structure and ignores the labels. We choose the cost of rename to be the average between the costs of deletion and insertion.

Definition 5.5 *Cost of Edit Mapping.* Let $\mathbf{T}$ and $\mathbf{T}'$ be two ordered labeled trees, $M \subseteq N_\epsilon(\mathbf{T}) \times N_\epsilon(\mathbf{T}')$ be an edit mapping between $\mathbf{T}$ and $\mathbf{T}'$, and $\gamma(\mathsf{v}, \mathsf{v}')$ be the cost of a node alignment. The *cost of the edit mapping M* is defined as

$$\gamma^*(M) = \sum_{(\mathsf{v},\mathsf{v}') \in M, \lambda(\mathsf{v}) \neq \lambda(\mathsf{v}')} \gamma(\mathsf{v}, \mathsf{v}').$$

The cost of an edit mapping is the sum of the cost of all node alignments in the mapping. The node alignments $(\mathsf{v}, \mathsf{v}') \in M$ with $\lambda(\mathsf{v}) = \lambda(\mathsf{v}')$ represent the nodes that have not been modified and thus are not considered in the cost formula. The cost of an edit operation, $e(\mathbf{T}, \mathsf{v}, \mathsf{v}')$, on a tree $\mathbf{T}$ is the cost of aligning v and v'.

The tree edit distance between two trees $\mathbf{T}$ and $\mathbf{T}'$ is the cost of the cheapest sequence of edit operations that transforms $\mathbf{T}$ into $\mathbf{T}'$ [Tai 1979]. Zhang and Shasha [Zhang and Shasha 1989] show that the cost of the cheapest edit sequence is equivalent to the cost of the cheapest edit mapping. We adopt the cost model of Definition 5.4 and define the *fanout weighted tree edit distance.*

Definition 5.6 *Fanout Weighted Tree Edit Distance.* Let $\mathbf{T}$ and $\mathbf{T}'$ be two ordered labeled trees. The *fanout weighted tree edit distance*, $\text{dist}_{ed}(\mathbf{T}, \mathbf{T}')$, between $\mathbf{T}$ and $\mathbf{T}'$ is the cost of the cheapest edit mapping, $M \subseteq N_\epsilon(\mathbf{T}) \times N_\epsilon(\mathbf{T}')$, between the two trees, i.e.,

$$\text{dist}_{ed}(\mathbf{T}, \mathbf{T}') = \min\{\gamma^*(M) \mid M \subseteq N_\epsilon(\mathbf{T}) \times N_\epsilon(\mathbf{T}') \text{ is an edit mapping}\}.$$

Previous work often adopts a unit cost model, which defines the cost of all node alignments to be 1. This reduces the tree edit distance to the minimum number of edit operations that transforms one tree into the other. The unit cost model does not distinguish between nodes with small and large fanout. This behavior leads to non-intuitive results, as illustrated in Figure 5. Tree $\mathbf{T}'$ is the result of deleting the leaves with labels `g` and `k` from $\mathbf{T}$, whereas $\mathbf{T}''$ is obtained from $\mathbf{T}$ by deleting the nodes labeled `c` and `e`. Intuitively, $\mathbf{T}'$ and $\mathbf{T}$ are much more similar (in structure) than $\mathbf{T}''$ and $\mathbf{T}$, but the unit cost tree edit distance dist_{unit} is 2 in both cases. The fanout weighted tree edit distance weights leaf and non-leaf changes differently.

The computation of the tree edit distance is expensive. The fastest algorithms use a dynamic programming approach that recursively decomposes the trees into subforests and computes the distance between all pairs of subforests. The decomposition of a single tree can result in $O(n^2)$ subforests, but not all of them are required to compute the tree edit distance. The optimal decomposition strategy results in an algorithm that uses $O(n^3)$ time and $O(n^2)$ space [Demaine et al. 2007]. The base algorithm by Zhang and Shasha [Zhang and Shasha 1989] uses $O(n^2 \log^2 n)$ time for trees with height $O(\log n)$, and $O(n^2)$ space.

6. THE PQ-GRAM DISTANCE

In this section we define *pq*-grams and a tree distance based on *pq*-grams. We show that, different from the tree edit distance, the *pq*-gram distance can be com-

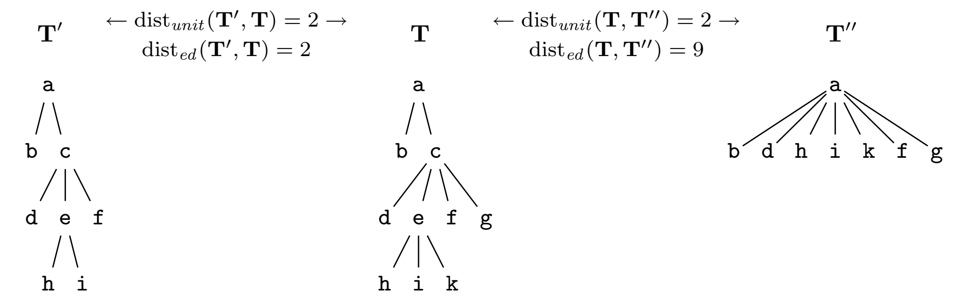


Fig. 5. Tree Edit Distance with Unit Cost (dist_{unit}) and Fanout Weighted (dist_{ed}) Cost Model.

puted efficiently in $O(n \log n)$ time and linear space, and that the *pq*-gram distance provides a lower bound for the fanout weighted tree edit distance.

Intuitively, the *pq*-grams of a tree are all subtrees of a specific shape. To ensure that each node of the tree appears in at least one *pq*-gram, we extend the tree with dummy nodes. The *pq*-grams are defined as subtrees of the extended tree.

Definition 6.1 *pq-Extended Tree.* Let $\mathbf{T}$ be a tree, and $p > 0$ and $q > 0$ be two integers. The *pq-extended tree*, $\mathbf{T}^{p,q}$, is constructed from $\mathbf{T}$ by adding $p-1$ ancestors to the root node, inserting $q-1$ children before the first and after the last child of each non-leaf node, and adding q children to each leaf of $\mathbf{T}$. All newly inserted nodes are dummy nodes that do not occur in $\mathbf{T}$.

Example 6.1. *Figure 6 shows the graphical representation of* $\mathbf{T}_1^{2,3}$*, the* 2,3-*extended tree of our example tree* $\mathbf{T}_1$.

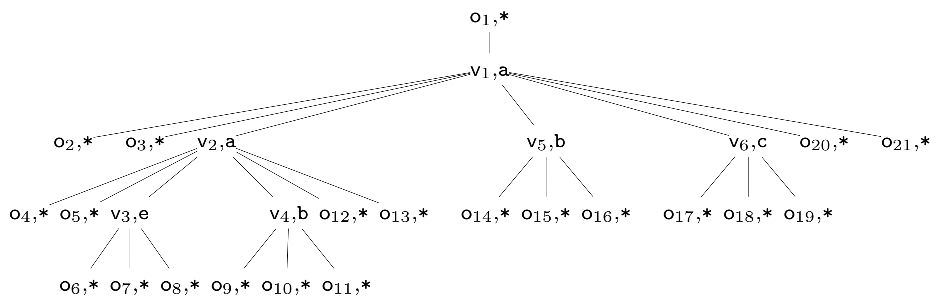

o1,*
v1,a
o2,* o3,* v2,a v5,b v6,c o20,* o21,*
o4,* o5,* v3,e v4,b o12,* o13,* o14,* o15,* o16,* o17,* o18,* o19,*
o6,* o7,* o8,* o9,* o10,* o11,*

Fig. 6. The Extended Tree $\mathbf{T}_1^{2,3}$.

Definition 6.2 *pq-Gram.* Let $\mathbf{T}$ be a tree, $\mathbf{T}^{p,q}$ the respective extended tree, $p > 0$, $q > 0$. A subtree of $\mathbf{T}^{p,q}$ is a *pq-gram* $\mathbf{G}$ of $\mathbf{T}$ iff

(1) $\mathbf{G}$ has q leaf nodes and p non-leaf nodes,

(2) all leaf nodes of $\mathbf{G}$ are children of a single node $\mathsf{a} \in N(\mathbf{G})$ with fanout q, called the *anchor node*,

(3) the leaf nodes of $\mathbf{G}$ are consecutive siblings in $\mathbf{T}^{p,q}$.

EXAMPLE 6.2. *Figure 7 shows some of the* $2, 3$*-grams of the example tree* $\mathbf{T}_1$.

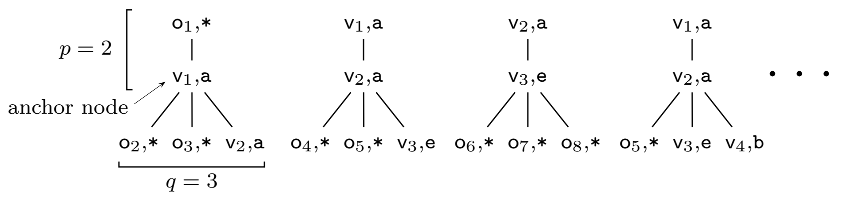


Fig. 7. Some of the $2, 3$-Grams of $\mathbf{T}_1$.

Definition 6.3 *Label Tuple.* Let $\mathbf{G}$ be a *pq*-gram with the nodes $N(\mathbf{G}) = \{\mathsf{v}_1, \ldots, \mathsf{v}_p,\ \mathsf{v}_{p+1}, \ldots,\ \mathsf{v}_{p+q}\}$, where v_i is the i-th node in preorder. The tuple $\lambda^*(\mathbf{G}) = (\lambda(\mathsf{v}_1), \ldots, \lambda(\mathsf{v}_p),\ \lambda(\mathsf{v}_{p+1}), \ldots, \lambda(\mathsf{v}_{p+q}))$ is called the *label tuple* of $\mathbf{G}$.

Subsequently, if the distinction is clear from the context, we use the term *pq-gram* for both, the *pq*-gram itself and its representation as a label tuple.

Definition 6.4 *pq-Gram Index.* Let $\mathbf{P}$ be the set of all *pq*-grams of a tree $\mathbf{T}$, $p > 0$, $q > 0$. The *pq-gram index*, $\mathcal{I}^{p,q}(\mathbf{T})$, of $\mathbf{T}$ is defined as the bag of label tuples of all *pq*-grams of $\mathbf{T}$, i.e., $\mathcal{I}^{p,q}(\mathbf{T}) = \biguplus_{\mathbf{G} \in \mathbf{P}} \lambda^*(\mathbf{G})$.

The tables in Figure 8 show the $2, 3$-gram index of $\mathbf{T}_1$ and $\mathbf{T}_2$, respectively. Note that *pq*-grams might appear more than once in a *pq*-gram index, e.g., $(\mathsf{a}, \mathsf{b}, *, *, *)$ appears twice in the index of $\mathbf{T}_1$.

$\mathcal{I}^{2,3}(\mathbf{T}_1)$ *labels*	$\mathcal{I}^{2,3}(\mathbf{T}_2)$ *labels*
(*, a, *, *, a)	(*, a, *, *, a)
(a, a, *, *, e)	(a, a, *, *, e)
(a, e, *, *, *)	(a, e, *, *, *)
(a, a, *, e, b)	(a, a, *, e, b)
(a, b, *, *, *)	(a, b, *, *, *)
(a, a, e, b, *)	(a, a, e, b, *)
(a, a, b, *, *)	(a, a, b, *, *)
(*, a, *, a, b)	(*, a, *, a, b)
(a, b, *, *, *)	(a, b, *, *, *)
(*, a, a, b, c)	(*, a, a, b, d)
(a, c, *, *, *)	(a, d, *, *, *)
(*, a, b, c, *)	(*, a, b, d, *)
(*, a, c, *, *)	(*, a, d, *, *)

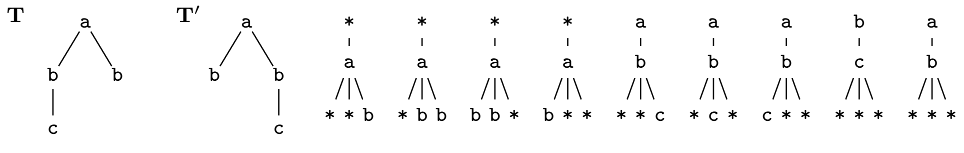
Fig. 8. $2, 3$-Gram Indexes of $\mathbf{T}_1$ and $\mathbf{T}_2$.

The *pq*-gram distance is a measure for the similarity of two trees. It is based on the number of *pq*-grams that differ between the indexes of two trees.

Definition 6.5 *pq-Gram Distance.* Let $\mathbf{T}_1$ and $\mathbf{T}_2$ be trees, $\mathbf{I}_1 = \mathcal{I}^{p,q}(\mathbf{T}_1)$, $\mathbf{I}_2 = \mathcal{I}^{p,q}(\mathbf{T}_2)$, $p > 0$, $q > 0$. The *pq-gram distance*, $\text{dist}^{p,q}(\mathbf{T}_1, \mathbf{T}_2)$, between the trees $\mathbf{T}_1$ and $\mathbf{T}_2$ is defined as the symmetric difference between the respective indexes:

$$\begin{aligned}\text{dist}^{p,q}(\mathbf{T}_1, \mathbf{T}_2) &= |\mathbf{I}_1 \uplus \mathbf{I}_2| - 2|\mathbf{I}_1 \Cap \mathbf{I}_2| \\ &= |\mathbf{I}_1 \Delta \mathbf{I}_2|\end{aligned} \tag{2}$$

The *pq*-gram distance is a pseudo-metric, i.e., it is non-negative, it is zero for identical trees, it is symmetric, and the triangle inequality holds. The pseudo-metric properties are essential for many similarity search algorithms since they allow to efficiently prune the search space [Zezula et al. 2006]. Different from a metric, in a pseudo-metric non-identical trees may be at distance zero. An example of two different trees with the same *pq*-gram index is shown in Figure 9(a). The children of the root nodes are swapped. The *pq*-grams responsible for detecting the swapped children are those anchored in the root nodes of $\mathbf{T}$ and $\mathbf{T}'$ (the four leftmost *pq*-grams in Figure 9(b)). However, as the swapped children have the same label, these *pq*-grams are not affected by the swap and the *pq*-gram distance is zero.



(a) The Trees $\mathbf{T}$ and $\mathbf{T}'$. (b) The 2, 3-Grams of $\mathbf{T}$ and $\mathbf{T}'$.

Fig. 9. Different Trees with the Same *pq*-Gram Index.

The *pq*-gram distance is the number of *pq*-grams that differ between two trees. The same number of different *pq*-grams may be considered a small change if the two trees are large, but a big change if the trees are small. We therefore define the *normalized pq-gram distance* that accounts for the tree size. The normalized *pq*-gram distance is 1 if two trees share no *pq*-grams and 0 for identical trees.

Definition 6.6 *Normalized pq-Gram Distance.* Let $\mathbf{T}_1$ and $\mathbf{T}_2$ be trees, $\mathbf{I}_1 = \mathcal{I}^{p,q}(\mathbf{T}_1)$, $\mathbf{I}_2 = \mathcal{I}^{p,q}(\mathbf{T}_2)$, $p > 0$, $q > 0$. The *normalized pq-gram distance*, $\text{dist}^{p,q}_{norm}(\mathbf{T}_1, \mathbf{T}_2)$, between the trees $\mathbf{T}_1$ and $\mathbf{T}_2$ is defined as follows:

$$\text{dist}^{p,q}_{norm}(\mathbf{T}_1, \mathbf{T}_2) = \frac{\text{dist}^{p,q}(\mathbf{T}_1, \mathbf{T}_2)}{|\mathbf{I}_1 \uplus \mathbf{I}_2| - |\mathbf{I}_1 \Cap \mathbf{I}_2|} \tag{3}$$

Example 6.3. *Consider the normalized* 2, 3*-gram distance between the trees* $\mathbf{T}_1$ *and* $\mathbf{T}_2$*. The corresponding* 2, 3*-gram indexes are shown in Figure 8. The bag-intersection of the two indexes is* $\{(\texttt{*},\texttt{a},\texttt{*},\texttt{*},\texttt{a})$, $(\texttt{a},\texttt{a},\texttt{*},\texttt{*},\texttt{e})$, $(\texttt{a},\texttt{e},\texttt{*},\texttt{*},\texttt{*})$, $(\texttt{a},\texttt{a},\texttt{*},\texttt{e},\texttt{b})$, $(\texttt{a},\texttt{b},\texttt{*},\texttt{*},\texttt{*})$, $(\texttt{a},\texttt{a},\texttt{e},\texttt{b},\texttt{*})$, $(\texttt{a},\texttt{a},\texttt{b},\texttt{*},\texttt{*})$, $(\texttt{*},\texttt{a},\texttt{*},\texttt{a},\texttt{b})$, $(\texttt{a},\texttt{b},\texttt{*},\texttt{*},\texttt{*})\}$, *which yields* $|\mathcal{I}^{2,3}(\mathbf{T}_1) \Cap \mathcal{I}^{2,3}(\mathbf{T}_2)| = 9$*. For the cardinality of the bag-union we get* $|\mathcal{I}^{2,3}(\mathbf{T}_1) \uplus \mathcal{I}^{2,3}(\mathbf{T}_2)| = |\mathcal{I}^{2,3}(\mathbf{T}_1)| + |\mathcal{I}^{2,3}(\mathbf{T}_2)| = 26$*. The normalized pq-gram distance is*

$$\text{dist}^{2,3}_{norm}(\mathbf{T}_1, \mathbf{T}_2) = \frac{26 - 2 \times 9}{26 - 9} = 0.47.$$

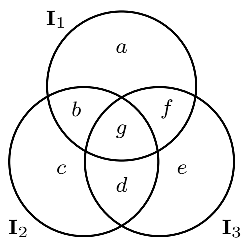


Fig. 10. Disjoint Subsets of $\mathbf{I}_1$, $\mathbf{I}_2$, and $\mathbf{I}_3$.

It is not obvious that the normalization maintains the pseudo-metric properties of the *pq*-gram distance. For example, if we normalize by $|\mathbf{I}_1 \uplus \mathbf{I}_2|$ instead of $|\mathbf{I}_1 \uplus \mathbf{I}_2| - |\mathbf{I}_1 \Cap \mathbf{I}_2|$, the triangle inequality is not satisfied. We show that our normalization of the *pq*-gram distance is a pseudo-metric.

Theorem 6.7 Pseudo-Metric. *The normalized pq-gram distance is a pseudo-metric, i.e., for any trees* $\mathbf{T}_1$, $\mathbf{T}_2$, *and* $\mathbf{T}_3$, *the following holds:*

(*1*) non-negativity*:* $\text{dist}^{p,q}_{norm}(\mathbf{T}_1, \mathbf{T}_2) \geq 0$
(*2*) reflexivity*:* $\mathbf{T}_1 = \mathbf{T}_2 \Rightarrow \text{dist}^{p,q}_{norm}(\mathbf{T}_1, \mathbf{T}_2) = 0$
(*3*) symmetry*:* $\text{dist}^{p,q}_{norm}(\mathbf{T}_1, \mathbf{T}_2) = \text{dist}^{p,q}_{norm}(\mathbf{T}_2, \mathbf{T}_1)$
(*4*) triangle inequality*:* $\text{dist}^{p,q}_{norm}(\mathbf{T}_1, \mathbf{T}_2) + \text{dist}^{p,q}_{norm}(\mathbf{T}_2, \mathbf{T}_3) \geq \text{dist}^{p,q}_{norm}(\mathbf{T}_1, \mathbf{T}_3)$

Proof. We follow a proof by Yianilos [Yianilos 2002], who introduces a metric normalization for the symmetric difference between sets. The *pq*-gram distance is the symmetric difference between *pq*-gram indexes which are bags.
Let $\mathbf{I}_1$, $\mathbf{I}_2$, $\mathbf{I}_3$ be the *pq*-gram indexes of $\mathbf{T}_1$, $\mathbf{T}_2$, and $\mathbf{T}_3$, respectively. The *non-negativity* of the normalized *pq*-gram distance follows from $|\mathbf{I}_1 \uplus \mathbf{I}_2| \geq 2|\mathbf{I}_1 \Cap \mathbf{I}_2|$, the *reflexivity* follows from $\mathbf{T}_1 = \mathbf{T}_2 \Rightarrow \mathbf{I}_1 = \mathbf{I}_2$, the *symmetry* follows from the symmetry of bag union and bag intersection. *Triangle inequality*: We use

$$\frac{|\mathbf{I}_1 \uplus \mathbf{I}_2| - 2|\mathbf{I}_1 \Cap \mathbf{I}_2|}{|\mathbf{I}_1 \uplus \mathbf{I}_2| - |\mathbf{I}_1 \Cap \mathbf{I}_2|} = 1 - \frac{|\mathbf{I}_1 \Cap \mathbf{I}_2|}{|\mathbf{I}_1| + |\mathbf{I}_2| - |\mathbf{I}_1 \Cap \mathbf{I}_1|}$$

to rewrite the triangle inequality as

$$1 - \frac{|\mathbf{I}_1 \Cap \mathbf{I}_2|}{|\mathbf{I}_1| + |\mathbf{I}_2| - |\mathbf{I}_1 \Cap \mathbf{I}_2|} + 1 - \frac{|\mathbf{I}_2 \Cap \mathbf{I}_3|}{|\mathbf{I}_2| + |\mathbf{I}_3| - |\mathbf{I}_2 \Cap \mathbf{I}_3|} \geq 1 - \frac{|\mathbf{I}_1 \Cap \mathbf{I}_3|}{|\mathbf{I}_1| + |\mathbf{I}_3| - |\mathbf{I}_1 \Cap \mathbf{I}_3|}. \quad (4)$$

We partition the indexes into disjoint subsets as shown in Figure 10. The lowercase letters in the figure are the cardinalities of the respective subsets. We further define $h := b + d + f + g$, substitute the cardinalities in (4) and simplify:

$$1 - \frac{b+g}{a+c+h} + 1 - \frac{d+g}{c+e+h} \geq 1 - \frac{f+g}{a+e+h}$$

$$\frac{b+g}{a+c+h} + \frac{d+g}{c+e+h} \leq \frac{f+g}{a+e+h} + 1. \quad (5)$$

Removing c from the denominator of the left-hand side can not decrease the left-hand side, thus (5) holds if the following is true:

$$\frac{b+g}{a+h} + \frac{d+g}{e+h} \leq \frac{f+g}{a+e+h} + 1.$$

We substitute 1 by h/h and add the fractions on each side:

$$\frac{(e+h)(b+g)+(a+h)(d+g)}{(a+h)(e+h)} \leq \frac{h(f+g)+h(a+e+h)}{h(a+e+h)}. \tag{6}$$

As $(a+h)(e+h) = ae+h(a+e+h) \geq h(a+e+h)$, we can remove the denominators, and (6) holds if the resulting inequality is true:

$$(e+h)(b+g)+(a+h)(d+g) \leq h(f+g)+h(a+e+h) \tag{7}$$

As $h = b+d+f+g$, the following inequalities hold: (α) $h \geq b+g$, (β) $h \geq d+g$, and (γ) $h^2+hg \geq h(b+d+2g)$. We show (7):

$$\begin{aligned}
(e+h)(b+g)+(a+h)(d+g) &\overset{(\alpha),(\beta)}{\leq} eh+h(b+g)+ah+h(d+g)\\
&= eh+h(b+d+2g)+ah\\
&\overset{(\gamma)}{\leq} eh+h^2+hg+ah\\
&= hg+h(a+e+h)\\
&\leq h(f+g)+h(a+e+h) \quad \square
\end{aligned}$$

Below we state and prove important properties of pq-grams. Theorem 6.8 states that the size of the pq-gram index is linear in the number of tree nodes. Lemma 6.9 gives a bound for the number of pq-grams in which a node appears.

THEOREM 6.8 LINEAR SIZE OF THE PQ-GRAM INDEX. *Let* $\mathbf{T}$ *be a tree with* $n = |N(\mathbf{T})|$ *nodes,* $p > 0$, $q > 0$. *The size of the pq-gram index of* $\mathbf{T}$ *is linear in the number of tree nodes:* $|\mathcal{I}^{p,q}(\mathbf{T})| = O(n)$.

PROOF. Let l and i be the number of leaf and non-leaf nodes of $\mathbf{T}$, respectively. We show $|\mathcal{I}^{p,q}(\mathbf{T})| = 2l+qi-1$ and conclude $|\mathcal{I}^{p,q}(\mathbf{T})| = O(n)$. First we count all pq-grams whose leftmost leaf is a dummy node: Each leaf is the anchor node of exactly one pq-gram whose leftmost leaf is a dummy node, giving l pq-grams. Each non-leaf node is the anchor of $q-1$ pq-grams whose leftmost leaf is a dummy node, giving $i(q-1)$ pq-grams. Second we count all pq-grams whose leftmost leaf is not a dummy node: Each node of the tree except the root is the leftmost leaf of exactly one pq-gram, giving $l+i-1$ pq-grams. The overall number of pq-grams is $l+i(q-1)+(l+i-1) = 2l+qi-1$. $\square$

The next lemma is used in the lower bound proof in Section 7 and determines the number of pq-grams in which a node v appears. This number is small for leaves and increases with the fanout of a node.

LEMMA 6.9. *Let* $\mathbf{T}$ *be a balanced tree with all leaf nodes at the same distance* h *from the root node and a fixed fanout* $f > 1$ *for all non-leaf nodes. Further, let* $\mathsf{v} \in N(\mathbf{T})$ *be a node at level* l, $0 \leq l \leq h$. *The number of pq-grams* ($p > 0, q > 0$) *that contain node* v *is as follows:*

$$\mathrm{cnt}_{pq}(\mathbf{T},\mathsf{v}) = \begin{cases} \frac{f^h-1}{f-1}(f+q-1)+f^h & l=0, h<p \quad (1)\\ \frac{f^p-1}{f-1}(f+q-1) & l=0, h\geq p \quad (2)\\ q+\frac{f^{h-l}-1}{f-1}(f+q-1)+f^{h-l} & l>0, h<l+p \quad (3)\\ q+\frac{f^p-1}{f-1}(f+q-1) & l>0, h\geq l+p \quad (4)\end{cases}$$

Proof. For the proof we systematically consider all positions in a *pq*-gram where node v can appear. We begin with the most general case (4). In this case node v is located in the "middle" of the tree such that v may appear in all positions of a *pq*-gram without touching the top or bottom of the tree. First, v appears as leaf in exactly q *pq*-grams (with v's parent being the anchor node). Next, v appears in the p non-leaf nodes of a *pq*-gram. When v is the anchor node, we get exactly $(f+q-1)$ *pq*-grams. Next, each of the f children of v becomes the anchor node, yielding $f(f+q-1)$ *pq*-grams. Repeating this step until v appears as root in the *pq*-grams, which is the case after $p-1$ steps, yields a total of $(f+q-1)\sum_{i=0}^{p-1} f^i = \frac{f^p-1}{f-1}(f+q+1)$ *pq*-grams, where v appears as non-leaf.

The other three cases cover special cases, where v, due to its level l in the tree and the definition of *pq*-grams, cannot appear in all positions of a *pq*-gram. In case (3), v is non-root but close enough to the leaf level such that each of the f^{h-l} leaf nodes that are descendants of v appear as anchor node in exactly one *pq*-gram. Note that, if $p > h - l + 1$, v does not appear in all p non-leaf positions of the *pq*-grams. Cases (1) and (2) are analogous to cases (3) and (4), except that v is the root of the tree. Therefore, by definition of the *pq*-grams, v does not appear in any of the q leaf positions of a *pq*-gram. □

Lemma 6.9 assumes a completely balanced tree with a fixed fanout. If f is the *maximum* fanout of v and its descendants within distance $p-1$, then $\text{cnt}_{pq}(\mathbf{T}, \mathsf{v})$ is an upper bound for the number of *pq*-grams that contain v. The cost for changing a leaf node ($h = l$) is $q+1$, thus depends only on q. For non-leaf nodes the impact of p is prevalent, and we can control the sensitivity of *pq*-grams to structural changes by choosing the value for p. The difference between non-leaf and leaf nodes is relevant for hierarchical data where values higher up in the hierarchy are more significant.

7. LOWER BOUND FOR THE TREE EDIT DISTANCE

In this section we show that the *pq*-gram distance, which can be computed in $O(n \log n)$ time and $O(n)$ space, provides a lower bound approximation for the fanout weighted tree edit distance. The lower bound guarantee allows to safely use *pq*-grams for pruning, for example, in a distance join. With the lower bound guarantee the pruning produces no false negatives, i.e., the pruning increases the efficiency without changing the join result.

The lower bound proof proceeds in three steps:

(1) *Unchanged pq-Grams* : Assume trees, $\mathbf{T}$ and $\mathbf{T}'$, with indexes $\mathbf{I}$ and $\mathbf{I}'$, respectively, an edit mapping, $M \subseteq N_\epsilon(\mathbf{T}) \times N_\epsilon(\mathbf{T}')$, and the inverse edit mapping, M^{-1}, of M. We define the unchanged *pq*-grams of $\mathbf{T}$, $U(\mathbf{T}, M, \mathbf{T}')$, and show

$$|\mathbf{I} \Cap \mathbf{I}'| \geq |U(\mathbf{T}, M, \mathbf{T}')| = |U(\mathbf{T}', M^{-1}, \mathbf{T})|.$$

(2) *Upper Bound for the pq-Gram Distance:* Using the above result, we prove the following upper bound for the *pq*-gram distance:

$$\begin{aligned} \text{dist}^{p,q}(\mathbf{T}, \mathbf{T}') &= |\mathbf{I}| + |\mathbf{I}'| - 2|\mathbf{I} \Cap \mathbf{I}'| \\ &\leq \textstyle\sum_{(\mathsf{v},\mathsf{v}') \in D(M)} \max(2q+2, 2\,\mathrm{f}_{\mathsf{v}} + 4q - 2) + \\ &\quad \textstyle\sum_{(\mathsf{v},\mathsf{v}') \in I(M)} \max(2q+2, 2\,\mathrm{f}_{\mathsf{v}'} + 4q - 2) + \\ &\quad \textstyle\sum_{(\mathsf{v},\mathsf{v}') \in R(M)} \max(2q+2, \mathrm{f}_{\mathsf{v}} + \mathrm{f}_{\mathsf{v}'} + 4q - 2) \end{aligned}$$

(3) *Lower Bound for the Tree Edit Distance:* Finally we show that the upper bound for the pq-gram distance (divided by 2) is a lower bound for the fanout weighted tree edit distance and conclude:

$$\frac{\text{dist}^{p,q}(\mathbf{T},\mathbf{T}')}{2} \le \text{dist}_{ed}(\mathbf{T},\mathbf{T}')$$

The lower bound holds for the minimal pq-gram pattern ($p = 1$). For larger patterns ($p > 1$) the pq-gram distance is more sensitive to structure changes than the fanout weighted tree edit distance and can grow beyond the fanout weighted tree edit distance.

The pq-gram distance also provides a lower bound for normalized tree edit distances: $\text{dist}^{p,q}(\mathbf{T},\mathbf{T}')/2x \le \text{dist}_{ed}(\mathbf{T},\mathbf{T}')/x$. Note though that not all normalizations yield a pseudo-metric. We proved the pseudo-metric properties for the normalization of Definition 6.6, which normalizes the pq-gram distance to values between 0 and 1.

7.1 Step 1: Unchanged pq-Grams

Definition 7.1 *Unchanged pq-Grams.* Let $\mathbf{T}$ and $\mathbf{T}'$ be two trees, $M \subseteq N_\epsilon(\mathbf{T}) \times N_\epsilon(\mathbf{T}')$ be an edit mapping, and $\mathbf{P}'$ be the set of all pq-grams of $\mathbf{T}'$. A pq-gram $\mathbf{G}$ of $\mathbf{T}$ is *unchanged* iff there is a pq-gram $\mathbf{G}'$ of $\mathbf{T}'$ such that M maps all non-dummy nodes of $\mathbf{G}$ to nodes of $\mathbf{G}'$ with the same label, and vice versa. The set of all unchanged pq-grams of $\mathbf{T}$ is denoted as $U(\mathbf{T},M,\mathbf{T}')$:

$$\begin{aligned}\mathbf{G} \in U(\mathbf{T},M,\mathbf{T}') \Leftrightarrow \exists\, \mathbf{G}' \in \mathbf{P}' :& \\ &\forall \mathsf{v} \in N(\mathbf{G}) \cap N(\mathbf{T})\, \exists \mathsf{v}' \in N(\mathbf{G}')[(\mathsf{v},\mathsf{v}') \in M \wedge \lambda(\mathsf{v}) = \lambda(\mathsf{v}')] \wedge \\ &\forall \mathsf{v}' \in N(\mathbf{G}') \cap N(\mathbf{T}')\, \exists \mathsf{v} \in N(\mathbf{G})[(\mathsf{v},\mathsf{v}') \in M \wedge \lambda(\mathsf{v}) = \lambda(\mathsf{v}')]\end{aligned}$$

For each unchanged pq-gram of one tree there is at least one pq-gram in the other tree with the same label tuple. Therefore the cardinality of the index intersection of two trees is at least the cardinality of the unchanged pq-grams.

LEMMA 7.2. *Given an edit mapping M between the nodes of two trees, $\mathbf{T}$ and $\mathbf{T}'$, the respective pq-gram indexes, $\mathbf{I}$ and $\mathbf{I}'$, and the inverse mapping $M^{-1} = \{(\mathsf{v}',\mathsf{v}) \mid (\mathsf{v},\mathsf{v}') \in M\}$, then*

$$|\mathbf{I} \Cap \mathbf{I}'| \ge |U(\mathbf{T},M,\mathbf{T}')| = |U(\mathbf{T}',M^{-1},\mathbf{T})|. \tag{8}$$

PROOF. $|U(\mathbf{T},M,\mathbf{T}')| = |U(\mathbf{T}',M^{-1},\mathbf{T})|$ follows from the symmetry of Definition 7.1. We show that each unchanged pq-gram, $\mathbf{G} \in U(\mathbf{T},M,\mathbf{T}')$, adds a label tuple to the index intersection: According to Definition 7.1 there is exactly one pq-gram $\mathbf{G}'$ of $\mathbf{T}'$ such that all non-dummy nodes of $\mathbf{G}$ are mapped to nodes of $\mathbf{G}'$ with the same label, and vice versa. Thus $\mathbf{G}$ adds the label tuple $\lambda^*(\mathbf{G}) = \lambda^*(\mathbf{G}')$ to the intersection. □

7.2 Step 2: Upper Bound for the pq-Gram Distance

In the following we prove an upper bound for the pq-gram distance that is a function of the deletions, insertions, and renames defined by M. Intuitively, the proof counts the non-matching pq-grams between two trees $\mathbf{T}$ and $\mathbf{T}'$ to derive an upper bound for their pq-gram distance. Part A of the proof counts the pq-grams that exist in $\mathbf{T}$ but not in $\mathbf{T}'$. This can be either due to a deletion or a rename (set V in the proof) or due to an insertion (set W in the proof). Part B of the proof equivalently counts the *pq*-grams that are present in $\mathbf{T}'$ but not in $\mathbf{T}$.

LEMMA 7.3. *Let M be an edit mapping between two trees, $\mathbf{T}$ and $\mathbf{T}'$, $p = 1$, $q > 0$, then*

$$\begin{aligned}\text{dist}^{p,q}(\mathbf{T},\mathbf{T}') \le\ & \textstyle\sum_{(\mathsf{v},\mathsf{v}')\in D(M)} \max(2q+2, 2\,\mathrm{f}_\mathsf{v} + 4q - 2) + \\ & \textstyle\sum_{(\mathsf{v},\mathsf{v}')\in I(M)} \max(2q+2, 2\,\mathrm{f}_{\mathsf{v}'} + 4q - 2) + \\ & \textstyle\sum_{(\mathsf{v},\mathsf{v}')\in R(M)} \max(2q+2, \mathrm{f}_\mathsf{v} + \mathrm{f}_{\mathsf{v}'} + 4q - 2).\end{aligned} \tag{9}$$

PROOF. Given $\text{dist}^{p,q}(\mathbf{T},\mathbf{T}') \overset{Def.\ 6.5}{=} |\mathbf{I}| + |\mathbf{I}'| - 2|\mathbf{I} \Cap \mathbf{I}'| \overset{(8)}{\le} |\mathbf{I}| - |U(\mathbf{T}, M, \mathbf{T}')| + |\mathbf{I}'| - |U(\mathbf{T}', M^{-1}, \mathbf{T})|$, we first derive an upper bound for $|\mathbf{I}| - |U(\mathbf{T}, M, \mathbf{T}')|$ (Part A), then we derive an upper bound for $|\mathbf{I}'| - |U(\mathbf{T}', M^{-1}, \mathbf{T})|$ (Part B). The sum of the two upper bounds is an upper bound for the *pq*-gram distance, $\text{dist}^{p,q}(\mathbf{T},\mathbf{T}')$.

A) Upper bound for $|\mathbf{I}| - |U(\mathbf{T}, M, \mathbf{T}')|$: We denote the set of all *pq*-grams of $\mathbf{T}$ with $\mathbf{P}$ and the set of all *pq*-grams of $\mathbf{T}'$ with $\mathbf{P}'$. As $|\mathbf{I}| = |\mathbf{P}|$ and $U(\mathbf{T}, M, \mathbf{T}') \subseteq \mathbf{P}$, $|\mathbf{I}| - |U(\mathbf{T}, M, \mathbf{T}')| = |\mathbf{P} \setminus U(\mathbf{T}, M, \mathbf{T}')|$. We derive an upper bound for the number of *pq*-grams that change, $|\mathbf{P} \setminus U(\mathbf{T}, M, \mathbf{T}')|$, by partitioning $\mathbf{P} \setminus U(\mathbf{T}, M, \mathbf{T}')$ into two disjoint sets, V and W, and by adding the upper bounds for $|V|$ and $|W|$. $\mathbf{P} \setminus U(\mathbf{T}, M, \mathbf{T}')$ is partitioned as follows:

(1) $\mathbf{G} \in V$ iff $\mathbf{G}$ contains a deleted or a renamed node, i.e., $\exists(\mathsf{v},\mathsf{v}') : \mathsf{v} \in N(\mathbf{G}) \wedge (\mathsf{v},\mathsf{v}') \in D(M) \cup R(M)$,

(2) $\mathbf{G} \in W$ otherwise.

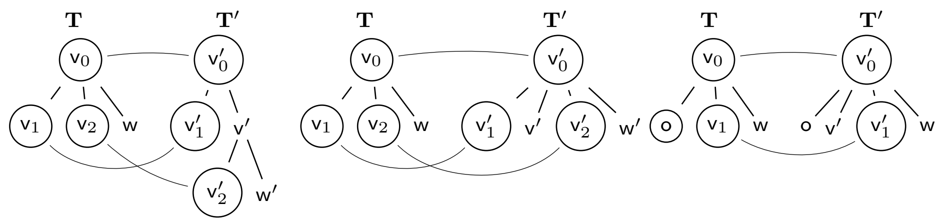
Upper bound for $|V|$: V is the set of all *pq*-grams, $\mathbf{G} \in \mathbf{P}$, that contain a renamed or a deleted node. The number of *pq*-grams that contain a node, $\mathsf{v} \in N(\mathbf{T})$, depends on the position of node v in the tree (root vs. non-root, leaf vs. non-leaf) and is shown in Table I. We get

$$\begin{aligned}|V| \le & \sum_{(\mathsf{v},\mathsf{v}')\in D(M),\mathrm{f}_\mathsf{v}=0} (q+1) + \sum_{(\mathsf{v},\mathsf{v}')\in D(M),\mathrm{f}_\mathsf{v}>0} (\mathrm{f}_\mathsf{v} + 2q - 1) + \\ & \sum_{(\mathsf{v},\mathsf{v}')\in R(M),\mathrm{f}_\mathsf{v}=0} (q+1) + \sum_{(\mathsf{v},\mathsf{v}')\in R(M),\mathrm{f}_\mathsf{v}>0} (\mathrm{f}_\mathsf{v} + 2q - 1).\end{aligned} \tag{10}$$

Position of v	$\lvert\{G \in \mathbf{P} \mid \mathsf{v} \in N(\mathbf{G})\}\rvert$
v is a root and leaf	1
v is root and non-leaf	$\mathrm{f}_\mathsf{v} + q - 1$
v is non-root and leaf	$q + 1$
v is non-root and non-leaf	$\mathrm{f}_\mathsf{v} + 2q - 1$

Table I. Number of *pq*-Grams ($p = 1$) that Contain a Node v.

Upper bound for $|W|$: Let $\mathbf{G} \in W$ be a *pq*-gram, $(\mathsf{v}_0, \mathsf{v}_1, \ldots, \mathsf{v}_k)$, $k \le q$, be the sequences of non-dummy nodes of $\mathbf{G}$ in preorder, and $\{\mathsf{v}'_0, \mathsf{v}'_1, \ldots, \mathsf{v}'_k\} \subseteq N(\mathbf{T}')$ be the nodes of $N(\mathbf{T}')$ such that $(\mathsf{v}_i, \mathsf{v}'_i) \in M$ for $0 \le i \le k$. As $\mathbf{G}$ does not contain renamed or deleted nodes ($\mathbf{G} \notin V$), all non-dummy nodes of $\mathbf{G}$ are mapped to a node of $N(\mathbf{T}')$ with the same label. Further, as $\mathbf{G} \notin U(\mathbf{T}, M, \mathbf{T}')$, there is no



(a) v' Is a Descendant of v'_0 and an Ancestor of v'_2. (b) v' Is a Sibling of v'_1 and v'_2, $\mathsf{v}'_1 < \mathsf{v}' < \mathsf{v}'_2$. (c) v' Is the Immediate Left Sibling of v'_1.

Fig. 11. Illustration for Proof 7.3: The Nodes of the $1,2$-Gram, $\mathbf{G}$, and their Mappings in $\mathbf{T}'$ are Circled. Mapped Nodes are Connected with a Line, the Lines for Deleted and Inserted Nodes are Not Shown.

pq-gram, $\mathbf{G}' \in \mathbf{P}'$, such that there is a one-to-one mapping $M' \subseteq M$ between the non-dummy nodes of $\mathbf{G}$ and $\mathbf{G}'$. Since M is an edit mapping, ancestor/descendant relation and node order can not change between the nodes mapped by M' (see Definition 5.1). This implies that there is a node $\mathsf{v}' \in N(\mathbf{T}') \setminus \{\mathsf{v}'_0, \mathsf{v}'_1, \dots, \mathsf{v}'_k\}$ that appears "between" the mapped nodes $\{\mathsf{v}'_0, \mathsf{v}'_1, \dots, \mathsf{v}'_k\}$ in $\mathbf{T}'$. There are two distinct cases:

(1) v' is a descendant of v'_0 and an ancestor of some nodes $\{\mathsf{v}'_1, \dots, \mathsf{v}'_k\}$ (see Figure 11(a)),

(2) v' is a sibling of some nodes $\{\mathsf{v}'_1, \dots, \mathsf{v}'_k\}$, v' has no descendant in $\{\mathsf{v}'_1, \dots, \mathsf{v}'_k\}$, and v' appears either between two nodes $\mathsf{v}'_1 \leq \mathsf{v}'_i < \mathsf{v}'_j \leq \mathsf{v}'_k$, (see Figure 11(b)) or is the immediate left (right) sibling of v'_1 (v'_k) (see Figure 11(c)).

We count the number of *pq*-grams that may be affected by a single node v'. Case (1): We need to count the *pq*-grams $\mathbf{G} \in \mathbf{P}$ with a node $\{\mathsf{v}_1, \dots, \mathsf{v}_k\} \in N(\mathbf{G})$ that is mapped to a child of v'. For a node v' with fanout $\mathrm{f}_{\mathsf{v}'}$ this can be at most $\mathrm{f}_{\mathsf{v}'} + q - 1$ *pq*-grams. Case (2): There are at most $k + 1 \leq q + 1$ positions between/before/after the nodes $\{\mathsf{v}'_1, \dots, \mathsf{v}'_k\}$ where v' can be inserted, thus the number of *pq*-grams that are affected by v' is at most $q + 1$.

We identify the nodes $\mathsf{v}' \in N(\mathbf{T}')$ that qualify for one of the above cases. Case (1): v' is a non-leaf (by definition). Further v' is an inserted node, i.e., $(\epsilon, \mathsf{v}') \in I(M)$, otherwise there would be a node $\mathsf{v} \in N(\mathbf{T}) \setminus N(\mathbf{G})$ such that $(\mathsf{v}, \mathsf{v}') \in M$, which violates the ancestor condition of Definition 5.1. Case (2): v' is an inserted node, i.e., $(\epsilon, \mathsf{v}') \in I(M)$, otherwise there would be a node $\mathsf{v} \in N(\mathbf{T}) \setminus N(\mathbf{G})$ such that $(\mathsf{v}, \mathsf{v}') \in M$, which violates the order condition of Definition 5.1. For the same reason, also all descendants of v' are inserted nodes. None of the descendants of v' qualifies for Case (2) (v' has, by definition, no descendants in $\{\mathsf{v}'_1, \dots, \mathsf{v}'_k\}$, thus no descendant of v' can be a sibling of such a node). v' is either a leaf or a non-leaf; if v' is a non-leaf, then it has at least one leaf descendant that does not qualify for Case (2), and we consider its leaf descendants instead of v' itself. By adding for each node v' the *pq*-grams $\mathbf{G} \in \mathbf{P}$ that v' may affect in the Cases (1) and (2) we get

$$|W| \leq \sum_{(\mathsf{v},\mathsf{v}') \in I(M), \mathrm{f}_{\mathsf{v}'} = 0} (q + 1) + \sum_{(\mathsf{v},\mathsf{v}') \in I(M), \mathrm{f}_{\mathsf{v}'} > 0} (\mathrm{f}_{\mathsf{v}'} + q - 1) \tag{11}$$

The upper bound of $|\mathbf{I}| - |U(\mathbf{T}, M, \mathbf{T}')| = |V| + |W|$ is the sum of the upper bounds for $|V|$ (10) and $|W|$ (11).

B) Upper bound for $|\mathbf{I}'| - |U(\mathbf{T}', M^{-1}, \mathbf{T})|$: The upper bound for $|\mathbf{I}'| - |U(\mathbf{T}', M^{-1}, \mathbf{T})| = V' + W'$ is derived by symmetric considerations,

$$|V'| \leq \sum_{(\mathsf{v},\mathsf{v}')\in I(M),\mathrm{f}_{\mathsf{v}'}=0} (q+1) + \sum_{(\mathsf{v},\mathsf{v}')\in I(M),\mathrm{f}_{\mathsf{v}'}>0} (\mathrm{f}_{\mathsf{v}'} + 2q - 1) +$$
$$\sum_{(\mathsf{v},\mathsf{v}')\in R(M),\mathrm{f}_{\mathsf{v}'}=0} (q+1) + \sum_{(\mathsf{v},\mathsf{v}')\in R(M),\mathrm{f}_{\mathsf{v}'}>0} (\mathrm{f}_{\mathsf{v}'} + 2q - 1) \tag{12}$$

$$|W'| \leq \sum_{(\mathsf{v},\mathsf{v}')\in D(M),\mathrm{f}_{\mathsf{v}}=0} (q+1) + \sum_{(\mathsf{v},\mathsf{v}')\in D(M),\mathrm{f}_{\mathsf{v}}>0} (\mathrm{f}_{\mathsf{v}} + q - 1) \tag{13}$$

We get the upper bound for $\mathrm{dist}^{p,q}(\mathbf{T}, \mathbf{T}') = |\mathbf{I}_1 \Delta \mathbf{I}_2| = V + W + V' + W'$ by adding the upper bounds for V (10), W (11), V' (12), and W' (13). For the sets $D(M)$ and $I(M)$ the different sums for leaves and non-leaves are expressed in a single sum that is equal or larger, for example:

$$\sum_{(\mathsf{v},\mathsf{v}')\in D(M),\mathrm{f}_{\mathsf{v}}=0} 2(q+1) + \sum_{(\mathsf{v},\mathsf{v}')\in D(M),\mathrm{f}_{\mathsf{v}}>0} (\mathrm{f}_{\mathsf{v}} + 2q - 1) + (\mathrm{f}_{\mathsf{v}} + q - 1) \leq$$
$$\sum_{(\mathsf{v},\mathsf{v}')\in D(M)} \max(2q+2, 2\,\mathrm{f}_{\mathsf{v}} + 4q - 2)$$

For the set $R(M)$ we need to consider the fanout of both v and v'. Considering all possible cases ($\mathrm{f}_{\mathsf{v}} = 0 \wedge \mathrm{f}_{\mathsf{v}'} = 0$, $\mathrm{f}_{\mathsf{v}} > 0 \wedge \mathrm{f}_{\mathsf{v}'} = 0$, $\mathrm{f}_{\mathsf{v}} = 0 \wedge \mathrm{f}_{\mathsf{v}'} > 0$, $\mathrm{f}_{\mathsf{v}} > 0 \wedge \mathrm{f}_{\mathsf{v}'} > 0$) we find that a single pair $(\mathsf{v}, \mathsf{v}') \in R(M)$ can contribute at most with $\max(2q+2, \mathrm{f}_{\mathsf{v}} + \mathrm{f}_{\mathsf{v}'} + 4q - 2)$ to the overall sum. □

7.3 Step 3: Lower Bound for the Tree Edit Distance

Theorem 7.4 Lower Bound. *Let $p = 1$ and $c \geq \max(2q-1, 2)$ be the cost of aligning leaf nodes. The pq-gram distance provides a lower bound for the fanout weighted tree edit distance, i.e., for any two trees, $\mathbf{T}$ and $\mathbf{T}'$,*

$$\frac{\mathrm{dist}^{p,q}(\mathbf{T}, \mathbf{T}')}{2} \leq \mathrm{dist}_{ed}(\mathbf{T}, \mathbf{T}').$$

Proof. Let M be a minimum cost edit mapping between the trees $\mathbf{T}$ and $\mathbf{T}'$. We define $a(\mathsf{x}, \mathsf{y}) := \max(2q+2, \mathrm{f}_{\mathsf{x}} + \mathrm{f}_{\mathsf{y}} + 4q - 2)$, $b(\mathsf{x}, \mathsf{y}) := \frac{\mathrm{f}_{\mathsf{x}} + \mathrm{f}_{\mathsf{y}}}{2} + c$,

$$A := \textstyle\sum_{(\mathsf{v},\mathsf{v}')\in D(M)} a(\mathsf{v},\mathsf{v}) + \sum_{(\mathsf{v},\mathsf{v}')\in I(M)} a(\mathsf{v}',\mathsf{v}') + \sum_{(\mathsf{v},\mathsf{v}')\in R(M)} a(\mathsf{v},\mathsf{v}')$$
$$B := \textstyle\sum_{(\mathsf{v},\mathsf{v}')\in D(M)} b(\mathsf{v},\mathsf{v}) + \sum_{(\mathsf{v},\mathsf{v}')\in I(M)} b(\mathsf{v}',\mathsf{v}') + \sum_{(\mathsf{v},\mathsf{v}')\in R(M)} b(\mathsf{v},\mathsf{v}').$$

As $\mathrm{dist}^{p,q}(\mathbf{T}, \mathbf{T}') \overset{\text{Def. 6.5}}{=} |\mathbf{I}\Delta\mathbf{I}'| \overset{(9)}{\leq} A$ and $B \overset{\text{Def. 5.6}}{=} \mathrm{dist}_{ed}(\mathbf{T}, \mathbf{T}')$, it is sufficient to show $A \leq 2B$. We show:

(1) $\sum_{(\mathsf{v},\mathsf{v}')\in D(M)} a(\mathsf{v},\mathsf{v}) \leq 2\sum_{(\mathsf{v},\mathsf{v}')\in D(M)} b(\mathsf{v},\mathsf{v})$: follows from $\forall(\mathsf{v},\mathsf{v}') \in D(M)$: $a(\mathsf{v},\mathsf{v}) \leq 2b(\mathsf{v},\mathsf{v})$. If $q = 1$ and $\mathrm{f}_{\mathsf{v}} = 0$ then $a(\mathsf{v},\mathsf{v}) = 2q+2$ and $a(\mathsf{v},\mathsf{v}) \leq 2b(\mathsf{v},\mathsf{v}) \Leftrightarrow 2q+2 \leq 2\,\mathrm{f}_{\mathsf{v}} + 2c \Leftrightarrow 2 \leq c$; otherwise $a(\mathsf{v},\mathsf{v}) = 2\,\mathrm{f}_{\mathsf{v}} + 4q - 2$ and $a(\mathsf{v},\mathsf{v}) \leq 2b(\mathsf{v},\mathsf{v}) \Leftrightarrow 2\,\mathrm{f}_{\mathsf{v}} + 4q - 2 \leq 2\,\mathrm{f}_{\mathsf{v}} + 2c \Leftrightarrow 2q - 1 \leq c$.

(2) $\sum_{(\mathsf{v},\mathsf{v}')\in I(M)} a(\mathsf{v}',\mathsf{v}') \leq 2\sum_{(\mathsf{v},\mathsf{v}')\in I(M)} b(\mathsf{v}',\mathsf{v}')$: analog rationale.

(3) $\sum_{(\mathsf{v},\mathsf{v}')\in R(M)} a(\mathsf{v},\mathsf{v}') \leq 2\sum_{(\mathsf{v},\mathsf{v}')\in R(M)} b(\mathsf{v},\mathsf{v}')$: follows from $\forall(\mathsf{v},\mathsf{v}') \in R(M)$: $a(\mathsf{v},\mathsf{v}') \leq 2b(\mathsf{v},\mathsf{v}')$. If $q = 1$ and the pair $(\mathrm{f}_{\mathsf{v}}, \mathrm{f}_{\mathsf{v}'}) \in \{(0,0),(0,1),(1,0)\}$ then $a(\mathsf{v},\mathsf{v}') = 2q+2$ and $a(\mathsf{v},\mathsf{v}') \leq 2b(\mathsf{v},\mathsf{v}') \Leftrightarrow 2q+2 \leq \mathrm{f}_{\mathsf{v}} + \mathrm{f}_{\mathsf{v}'} + 2c$ which holds for $c \geq 2$; otherwise $a(\mathsf{v}',\mathsf{v}') = \mathrm{f}_{\mathsf{v}} + \mathrm{f}_{\mathsf{v}'} + 4q - 2$ and $a(\mathsf{v}',\mathsf{v}') \leq 2b(\mathsf{v}',\mathsf{v}') \Leftrightarrow \mathrm{f}_{\mathsf{v}} + \mathrm{f}'_{\mathsf{v}} + 4q - 2 \leq \mathrm{f}_{\mathsf{v}} + \mathrm{f}_{\mathsf{v}'} + 2c \Leftrightarrow 2q - 1 \leq c$. □

8. ALGORITHMS

The core of the pq-gram distance computation is the computation of the indexes. In this section we present an algorithm for computing the pq-gram index and we show its linear complexity. A feature of this algorithm is that it can be efficiently implemented for trees, for example, XML data, stored in a relational database. We present an implementation that requires only a single scan over the relation that stores the trees.

8.1 An Algorithm for the pq-Gram-Index

The basic idea of the pq-`Gram-Index` algorithm (see Algorithms 8.1 and 8.2) is to move a pq-gram pattern vertically and horizontally over the tree as illustrated in Figure 12. After each move the nodes covered by the pattern form a pq-gram.

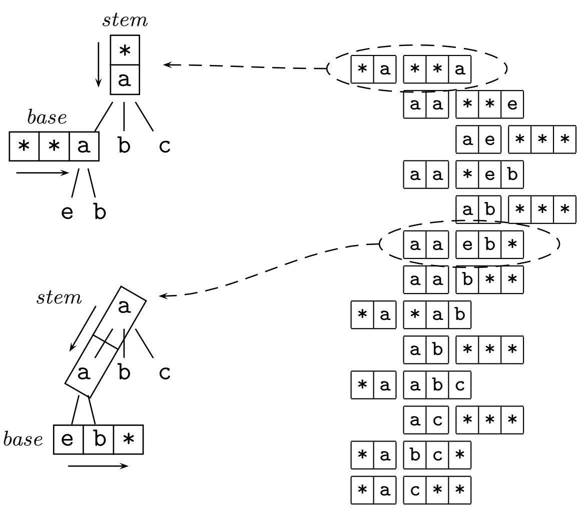
Algorithm 8.1: pq-`Gram-Index`$(\mathbf{T}, p, q)$

```
1 I : empty relation with schema (labels);
2 stem: shift register of size $p$ (filled with *);
3 I ← index($\mathbf{T}, p, q$, I, root($\mathbf{T}$), stem);
4 return I;
```

Algorithm 8.2: `index`$(\mathbf{T}, p, q, \mathtt{I}, \mathsf{a}, stem)$

```
5  base: shift register of size $q$ (filled with *);
6  stem ← shift(stem, $\lambda$(a));
7  if a is a leaf then
8      I ← I ∪ {stem ∘ base};
9  else
10     foreach child c (from left to right) of a do
11         base ← shift(base, $\lambda$(c));
12         I ← I ∪ {stem ∘ base};
13         I ← index($\mathbf{T}, p, q$, I, c, stem);
14     for $k$ ← 1 to $q - 1$ do
15         base ← shift(base, *);
16         I ← I ∪ {stem ∘ base};
17 return I;
```

We use two shift registers, *stem* of size p and *base* of size q to represent the labels of a pq-gram; *stem* stores the labels of the anchor node and its ancestors, *base* the



(a) Moving the *pq*-Gram Pattern in the Tree.

(b) Shift Registers *stem* and *base*.

I

labels
(*, a, *, *, a)
(a, a, *, *, e)
(a, e, *, *, *)
(a, a, *, e, b)
(a, b, *, *, *)
(a, a, e, b, *)
(a, a, b, *, *)
(*, a, *, a, b)
(a, b, *, *, *)
(*, a, a, b, c)
(a, c, *, *, *)
(*, a, b, c, *)
(*, a, c, *, *)

(c) Resulting Index I.

Fig. 12. Illustration of the *pq*-Gram Index Calculation.

labels of the children of the anchor node. A shift register *reg* supports a single operation shift(*reg*, *el*), which returns *reg* with the oldest element dequeued and *el* enqueued. For example, shift((a, b, c), x) returns (b, c, x). The concatenation of the two registers, *stem* ∘ *base*, is a label tuple in the *pq*-gram index, i.e., for $stem = (l_1, \dots, l_p)$ and $base = (l_{p+1}, \dots, l_{p+q})$ the label tuple in the *pq*-gram index is $(l_1, \dots, l_p, l_{p+1}, \dots, l_{p+q})$.

pq-`Gram-Index` takes as input a tree **T**, *p*, and *q*, and returns a relation that contains the *pq*-gram index of **T**. After the initialization, `index` calculates the *pq*-grams recursively, starting from the root node of **T**. Intuitively, `index` shifts a *pq*-gram shaped pattern vertically and horizontally over the tree, and the nodes covered by the pattern form a *pq*-gram. First, the label of the anchor node a is shifted into register *stem*, which corresponds to moving the *pq*-gram pattern one step down. Now *stem* contains the labels of a and its $p - 1$ ancestors. The loop at line 10 moves the register *base* from left to right over the children of a in order to produce all the *pq*-grams with anchor node a and calls `index` recursively for each child of a. Overall, `index` adds $f_a + q - 1$ label tuples to I if a is a non-leaf, otherwise 1 label tuple is added. The *pq*-extended tree is calculated on the fly by an adequate initialization of the shift registers (lines 2, 5, 14–16).

Example 8.1. *Assume $p = 2$, $q = 3$, and the tree $\mathbf{T}_1$ from Figure 3. The main data structures of the* `index` *algorithm are visualized in Figure 12. After the initialization,* `index`$(\mathbf{T}_1, 2, 3, \{\}, v_1, (*, *))$ *is called. Line 5 initializes base = (*, *, *), and line 6 shifts the label of v_1 into the register stem, yielding stem = (*, a). Since v_1 is not a leaf, we enter the loop at line 10 and process all children of v_1. The label of the first child, v_2, is shifted into register base, yielding base = (*, *, a),* *and the first label tuple* $(\mathtt{*}, \mathsf{a}, \mathtt{*}, \mathtt{*}, \mathsf{a})$ *is added to the result set* `I`*. Figure 12(b) shows the values of stem and base each time a label tuple is added to* `I`*. The indentation illustrates the recursion. The table in Figure 12(c) shows the result relation* `I` *with the label tuples in the order in which they are produced by the algorithm.*

pq-`Gram-Index` has runtime complexity $O(n)$ for a tree $\mathbf{T}$, where $n = |N(\mathbf{T})|$: Each recursive call of `index` processes one node, and each node is processed exactly once. For the distance computation between two trees the index intersection is computed in $O(n \log n)$ time and $O(n)$ space using a standard sort-merge approach. Thus, the overall complexity of computing the *pq*-gram distance between two trees $\mathbf{T}$ and $\mathbf{T}'$ is $O(n \log n)$ time and $O(n)$ space, where $n = \max(|N(\mathbf{T})|, |N(\mathbf{T}')|)$ is the number of nodes of the larger tree.

8.2 Relational Implementation

The algorithms given above are not optimized for a particular encoding of trees. In this section we present a scalable *pq*-gram index algorithm for tree sets that are stored in a relational database. The algorithm works for all encodings of ordered labeled trees that support efficient implementations of the following functions: (a) sort the tree nodes in preorder, (b) decide whether a node is a leaf, and (c) decide the ancestor-descendant relationship between nodes. Examples of encodings that satisfy these criteria are the interval encoding [Celko 1994; 2004] and the Dewey encoding [Tatarinov et al. 2002; O'Neil et al. 2004]. The presentation of our algorithms assumes the interval encoding.

The interval encoding has been used to store and query XML data [Zhang et al. 2001; Grust 2002; Al-Khalifa et al. 2002; DeHaan et al. 2003]. It stores the structure information of a node as a pair of integers (interval). We associate a unique index number with each tree in the set. A node of a tree is represented as a quadruple of tree index, node label, and left and right endpoint of the node's interval.

Definition 8.1 *Interval Encoding.* An *interval encoding* of a tree $\mathbf{T}$ is a relation R that for each node $\mathsf{v} \in \mathbf{T}$ contains a tuple $(id(\mathbf{T}), \lambda(\mathsf{v}), lft, rgt)$; $id(\mathbf{T})$ is a unique identifier of the tree $\mathbf{T}$, $\lambda(\mathsf{v})$ is the label of v, *lft* and *rgt* are the endpoints of the interval representing the node. *lft* and *rgt* are constrained as follows:

—$lft < rgt$ for all $(id, lbl, lft, rgt) \in R$,

—$lft_{\mathsf{a}} < lft_{\mathsf{d}}$ and $rgt_{\mathsf{a}} > rgt_{\mathsf{d}}$ if node a is an ancestor of node d, and $(id(\mathbf{T}), \lambda(\mathsf{a}), lft_{\mathsf{a}}, rgt_{\mathsf{a}}) \in R$, and $(id(\mathbf{T}), \lambda(\mathsf{d}), lft_{\mathsf{d}}, rgt_{\mathsf{d}}) \in R$,

—$rgt_{\mathsf{v}} < lft_{\mathsf{w}}$ if node v is a left sibling of node w, and $(id(\mathbf{T}), \lambda(\mathsf{v}), lft_{\mathsf{v}}, rgt_{\mathsf{v}}) \in R$, and $(id(\mathbf{T}), \lambda(\mathsf{w}), lft_{\mathsf{w}}, rgt_{\mathsf{w}}) \in R$,

—$rgt = lft + 1$ if node v is a leaf node, and $(id(\mathbf{T}), \lambda(\mathsf{v}), lft, rgt) \in R$.

We get an interval encoding for a tree by traversing the tree in preorder, using an incremental counter that assigns the left interval value *lft* to each node when it is visited first, and the right value *rgt* when it is visited last. Figure 13 shows an address tree of our application, where each node is annotated with the endpoints of the interval.

The interval encoding of a tree allows a scalable implementation of the algorithm *pq*-`Gram-Index` for a set of trees stored in a relation `F` with schema $(treeID, label, lft, rgt)$. We define the following cursor:

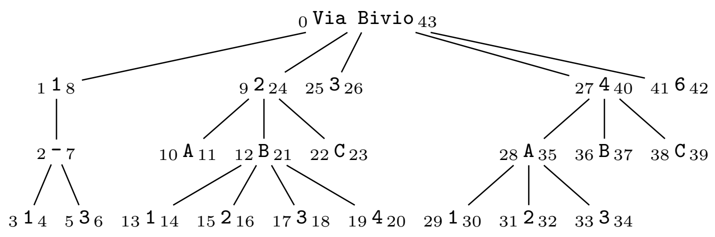


Fig. 13. Address Tree with Interval Encoding.

```
cur = SELECT * FROM F ORDER BY treeID,lft
```

Then with a single scan all trees can be processed, and each tree is processed node-by-node in preorder. Our experiments in Section 9.1 confirm the scalability of this approach to large trees.

The Algorithms 8.3 and 8.4 are adapted for the interval encoding and the changes are highlighted. Instead of a tree, *pq*-`Gram-Index` gets a cursor as an argument. Algorithm `index` processes all nodes of the tree in preorder, and when it terminates the cursor points to the root node of the next tree in the set.

Algorithm 8.3: pq-`Gram-Index`(cur, p, q)

```
I : empty relation with schema ($labels$);
$stem$: shift register of size $p$ (filled with *);
I $\leftarrow$ index($cur, p, q$, I, fetch($cur$), $stem$);
return I;
```

Algorithm 8.4: `index`(cur, p, q, I, a, $stem$)

```
$base$: shift register of size $q$ (filled with *);
$stem \leftarrow$ shift($stem, \lambda$(a));
$cur \leftarrow$ next($cur$);
if isLeaf(a) then
    I $\leftarrow$ I $\cup$ {$stem \circ base$};
else
    c $\leftarrow$ fetch($cur$);
    while isDescendant(c, a) do
        $base \leftarrow$ shift($base, \lambda$(c));
        I $\leftarrow$ I $\cup$ {$stem \circ base$};
        I $\leftarrow$ index($cur, p, q$, I, c, $stem$);
        c $\leftarrow$ fetch($cur$);
    for $k \leftarrow 1$ to $q - 1$ do
        $base \leftarrow$ shift($base$, *);
        I $\leftarrow$ I $\cup$ {$stem \circ base$};
return I;
```

Algorithm `index` calls the following two functions:

—`isLeaf`(`v`): Returns true iff `v` is a leaf node, i.e., $\text{lft}(\mathtt{v}) + 1 = \text{rgt}(\mathtt{v})$.

—`isDescendant`(`d`, `a`): Returns true iff `d` is a descendant of `a`, i.e., $\text{lft}(\mathtt{a}) < \text{lft}(\mathtt{d})$ **and** $\text{rgt}(\mathtt{a}) > \text{rgt}(\mathtt{d})$ **and** $\mathit{treeID}(\mathtt{a}) = \mathit{treeID}(\mathtt{d})$ **and** $\mathtt{d} \neq \mathtt{null}$.

Checking the ancestor-descendant relationship between nodes is a constant time operation for the interval encoding, while checking the parent-child relationship is more complex. In our algorithm this amounts to the same thing: When the loop on line 12 is entered the first time, `c` is the next node after `a` in preorder (or `null`). Thus, if `c` is a descendant of `a`, it must be a child. The recursive call in line 15 will process `c` and all its descendants, and set the cursor on the next node after the processed nodes. Again, if this is a descendant of `a`, then it is a child. Thus, the while-loop of Algorithm 8.4 is equivalent to the for-loop of Algorithm 8.2.

9. EXPERIMENTS

We evaluate the efficiency and the effectiveness of the pq-gram distance. The pq-gram distance can be computed efficiently even between very large trees for which the tree edit distance is not feasible. The sensitivity of pq-grams is controlled by the parameters p and q; we test deletions for leaf and non-leaf nodes with different pairs of parameters and confirm our analytic results. The effectiveness of pq-grams is evaluated in two experiments on real world data. In the first experiment pq-grams are tested for the street matching task introduced in Section 2 and their effectiveness is compared to other approximations in the literature that we have implemented. In the second experiment we approximately join XML documents that contain spelling mistakes and missing elements. Both experiments show that pq-grams are a good approximation of the tree edit distance. The other approximations are clearly outperformed in terms of precision and recall.

All algorithms were implemented in Java 1.6 and run on a 2.6GHz processor. Unless otherwise mentioned, the main memory was limited to 1GB.

9.1 Scalability

We evaluate the scalability of our algorithm by comparing it with the tree edit distance by Zhang and Shasha [Zhang and Shasha 1989] and the tree embedding distance [Garofalakis and Kumar 2005], and we investigate the influence of the parameters p and q on the scalability of the pq-gram distance.

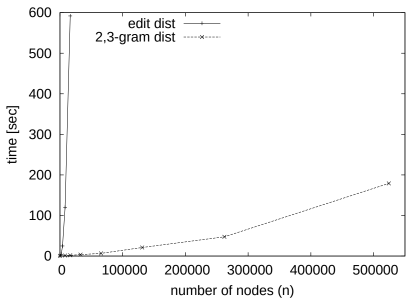
As a test set we produce pairs of trees $(\mathbf{T}_1, \mathbf{T}_2)$ of size $|N(\mathbf{T}_1)| = |N(\mathbf{T}_2)| = n$ with up to 5×10^5 nodes. The height of the trees is logarithmic, and the labels for each tree are randomly chosen from a set of n different labels.

Figure 14(a) shows the runtimes for the tree edit distance and the $2,3$-gram distance computation for different tree sizes. For the tree edit distance we use our own implementation which for large trees runs almost three orders of magnitude faster than the implementation provided by Zhang and Shasha[3]. For the pq-gram distance we use the relational implementation presented in Section 8.2. For large trees the computation time of the tree edit distance grows very fast. The largest

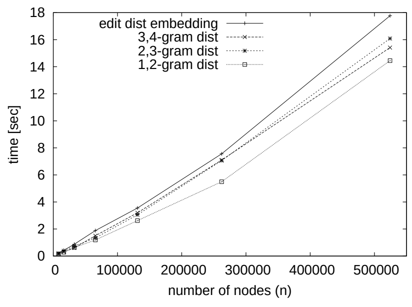
[3]`http://www.cs.nyu.edu/cs/faculty/shasha/papers/tree.html`

tree in our test set that we were able to compute has 16383 nodes. For larger trees the tree edit distance runs out of main memory, even if we allow the maximum available memory of 12GB. The runtime of the *pq*-gram distance is almost linear in the tree size.

Figure 14(b) compares the *pq*-gram distance for varying parameters with the tree embedding distance. The tree embedding distance was implemented according to the algorithm of Garofalakis and Kumar [Garofalakis and Kumar 2005]. For the comparison both algorithms run in main memory. The *pq*-gram distance algorithm is slightly faster, and varying values for p and q have little impact on the scalability of the *pq*-gram distance computation.



(a) Tree Edit Distance.



(b) Tree Embedding Distance.

Fig. 14. Scalability Results.

9.2 Sensitivity to Structural Changes

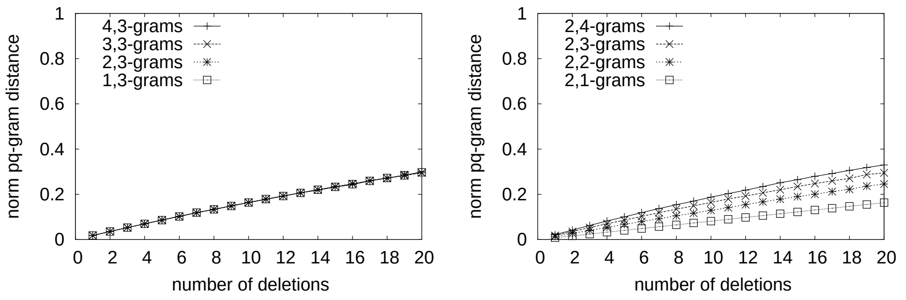
In Section 6 we point out that the *pq*-gram distance weights deletions of non-leaf nodes more than deletions of leaves, and the sensitivity to structural changes is controlled by the parameters p and q. We show this property in an experiment, where only non-leaf nodes or only leaf nodes are deleted for varying parameters, and calculate the *pq*-gram distance for both cases.

We create an artificial tree $\mathbf{T}$ with 144 nodes, 102 leaves, and height 6. The fanout of the non-leaf nodes is between 2 and 5. We randomly delete leaf or non-leaf nodes from $\mathbf{T}$. Figure 15 shows the normalized *pq*-gram distance for different numbers of deletions. Each value in Figure 15 is an average over 100 runs.

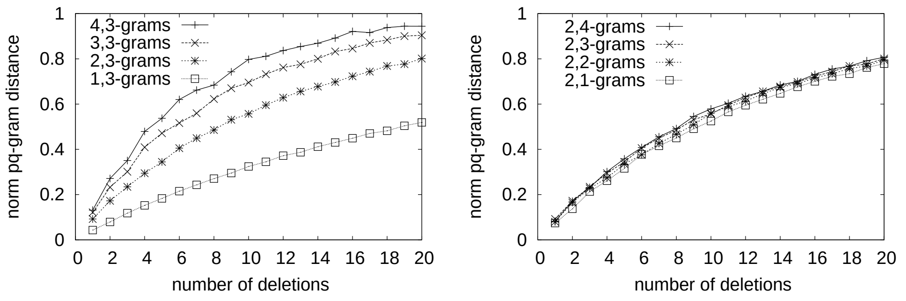
For leaf node deletions only q has an influence (see Figure 15(a)). For the deletion of non-leaf nodes q has a small impact compared to p (see Figure 15(b)). This confirms our analytical results. The sensitivity to leaf changes depends only on q, structural sensitivity is emphasized with higher values of p. For non-leaf deletions the *pq*-gram distance is larger than for leaf deletions.

9.3 Real World Data: Street Matching

9.3.1 *Evaluation of a Matching Algorithm.* In a tree matching scenario two sets of trees, F and F', are given and a mapping $M_x \subseteq \mathsf{F} \times \mathsf{F}'$ is computed. A *match* is a pair of trees in the mapping. The quality of the computed mapping, M_x, is evaluated with respect to a correct mapping $M_c \subseteq \mathsf{F} \times \mathsf{F}'$ that contains all pairs of



(a) Deletion of Leaf Nodes.



(b) Deletion of Non-Leaf Nodes.

Fig. 15. Properties of the pq-Gram Distance.

trees that *should* match. We measure *precision*, $p = \frac{|M_x \cap M_c|}{|M_x|}$, (correctly computed matches to total number of computed matches) and *recall*, $r = \frac{|M_x \cap M_c|}{|M_c|}$, (correctly computed matches to total number of correct matches). The precision is high if the returned matches are correct, the recall is high if the algorithm does not miss correct matches. The F-measure, $F = \frac{2pr}{p+r}$, is a well-known performance measure [van Rijsbergen 1979] that considers both recall and precision.

9.3.2 *Street Matching.* We test the effectiveness of pq-grams for the street matching task introduced in Section 2. We build the address trees for all streets of two address tables, `Ownr` and `Rsdt`. Each tree $\mathbf{T}$ represents a street with all the addresses in that street. The tree set produced from table `Ownr` consists of 299 trees with 52,509 nodes in total, reflecting 44,427 addresses, the set produced from `Rsdt` consists of 302 trees with 52,509 nodes and 43,187 addresses.

The matching is done as follows. For each distance function dist_x we compute a mapping $M_x \subseteq \mathsf{F} \times \mathsf{F}'$. Two trees $\mathbf{T} \in \mathsf{F}$ and $\mathbf{T}' \in \mathsf{F}'$ *match*, i.e., $(\mathbf{T}, \mathbf{T}') \in M_x$, iff $\mathbf{T}$ has only one nearest neighbor in F', namely $\mathbf{T}'$, and vice versa. We compute a mapping for the fanout weighted and the unit cost tree edit distance (see Section 5), the pq-gram distance (see Section 6), the tree embedding distance [Garofalakis and Kumar 2005], the binary branch distance [Yang et al. 2005], the bottom-up distance [Valiente 2001], 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 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 `Rsdt` that do not exist in `Ownr`, thus $|M_c| = 299$ for the computation of precision and recall.

The results for the address tables `Rsdt` and `Ownr` are shown in Table II. In terms of overall effectiveness (F-measure) the fanout weighted edit distance outperforms both the unit cost edit distances and the approximations. The efficiency of the approximations is clearly greater than that of the tree edit distances: All of them can be computed within seconds, whereas the tree edit distance takes more than half an hour.

Distance	Correct	Recall	Precision	F-Measure	Runtime
fanout weighted edit distance	259	86.6%	98.5%	0.922	ca. 19 min
unit cost edit distance	247	82.6%	96.5%	0.890	ca. 14 min
node intersection	197	65.9%	93.8%	0.774	4.3s
pq-grams ($p = 3, q = 3$)	236	78.9%	98.7%	0.877	8.1s
pq-grams ($p = 1, q = 2$)	234	78.3%	97.9%	0.870	9.7s
pq-grams ($p = 2, q = 3$)	231	77.3%	98.3%	0.865	9.6s
tree-embedding	206	68.9%	96.3%	0.803	7.1s
binary branch	193	64.5%	93.2%	0.763	7.4s
bottom-up	148	49.5%	92.5%	0.645	67.0s

Table II. Effectiveness of the Tree Edit Distance and its Approximations.

The precision of the pq-gram distance is higher than the precision of the unit cost tree edit distance. For some values of p and q it is even higher than the precision of the fanout weighted tree edit distance. The pq-gram distance outperforms the other approximations with respect to both recall and precision for all tested parameters.

9.4 Real World Data: Matching XML Data

In this experiment we study the effectiveness of the normalized pq-gram distance for XML data. We use real world XML data sets, add noise (spelling mistakes and missing elements), and we approximately join the original and the noisy set.

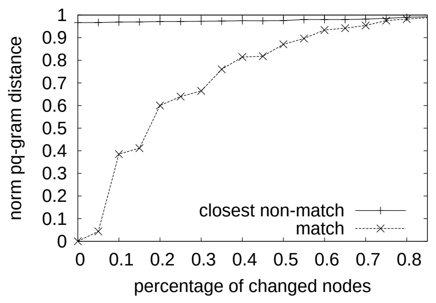
The Data Sets. We use the DBLP[4] (bibliography), the SwissProt[5] (protein sequence database), and the Treebank[6] (parts of speech tagged English sentences) XML databases. We split each database into a set of (sub)documents by deleting the root node, and we randomly choose 200 of the resulting documents for our experiments (requiring a minimum document size of 15 nodes). We represent the XML documents as ordered, labeled trees. A node label is a pair (tag, val), where tag is the name of an element/attribute and val is its text content. If an element contains only sub-elements and no content, then val is the empty string.

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

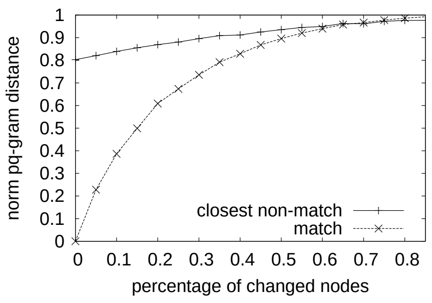
[4]`http://dblp.uni-trier.de`
[5]`http://us.expasy.org/sprot/`
[6]`http://www.cis.upenn.edu/~treebank/`

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.

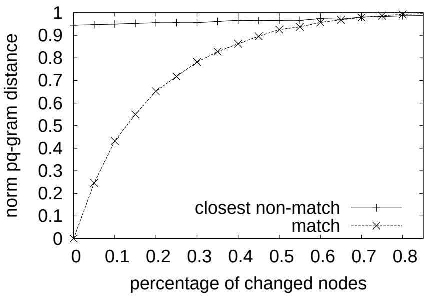
Adding Noise. We modify the *original documents* by deleting and renaming random nodes. Node deletions simulate missing elements or attributes and modify the document structure. Renamed nodes represent different tag names or spelling mistakes in the text values. The resulting noisy document is the *match* of the original document, all other noisy documents are *non-matches.* In our figures we show the noise as the percentage of changed nodes.



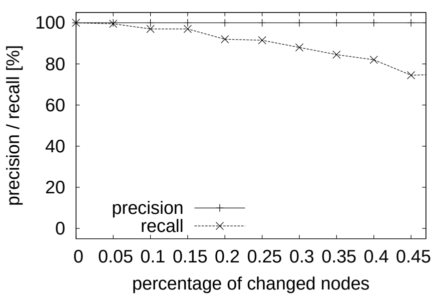
(a) DBLP



(b) SwissProt



(c) Treebank

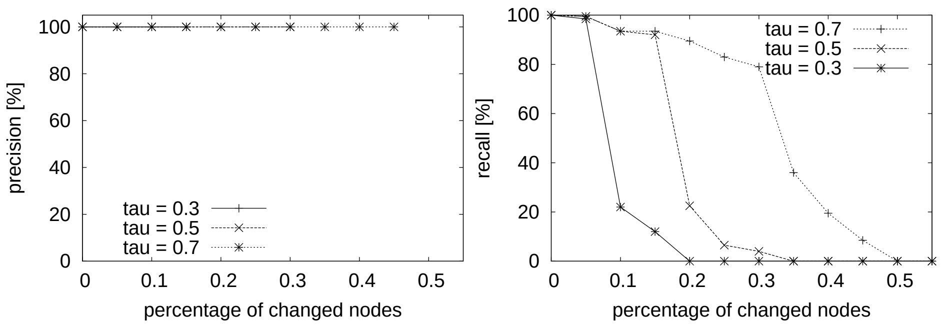


(d) 1:1 Matches for SwissProt

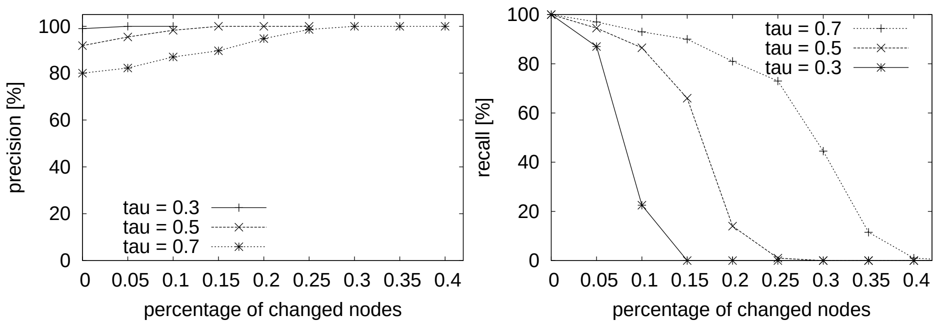
Fig. 16. (a-c) Distance between Matches and Non-Matches. (d) 1:1 Matches for SwissProt.

Distance between Matches and Non-Matches. Each original document has exactly one match. Figures 16(a)–16(c) show the average normalized *pq*-gram distance ($p = 2, q = 3$) of the original documents to their match and to the closest non-match. The noise is increased in steps of 5%. The SwissProt documents are more similar to each other than the DBLP and Treebank documents. The normalized *pq*-gram distance to the matches is almost linear to the percentage of modified nodes. It effectively approximates the tree edit distance. All documents are modified, thus also the distance to the non-matches increases with the number of changed nodes.

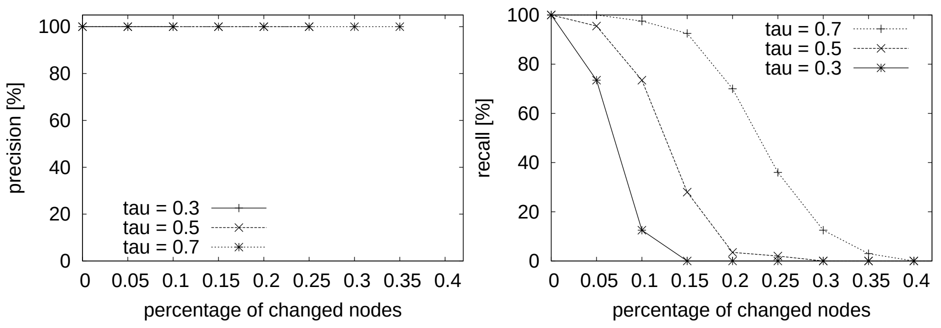
Effectiveness. Figure 17 shows precision and recall for different distance thresholds τ. Moving up the threshold decreases the precision and increases the recall.



(a) DBLP



(b) SwissProt



(c) Treebank

Fig. 17. Matching with Different Thresholds.

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 90% for 15% noise.

For SwissProt the precision drops as we increase the threshold. The SwissProt documents are clustered into groups of similar documents (protein variants). The clustering of the data is evident from the precision values in Figure 17(b) for 0% noise (approximate self join): Already for $\tau = 0.3$ some documents match other documents than themselves. We improve the result for SwissProt using a variable threshold. Each document is matched to its nearest neighbor. If a document has more than one nearest neighbor, no match is returned. Figure 16(d) shows the results for the SwissProt database. The algorithm returns precise matches, and even for 25% noise we miss only about 10% of the matches.

10. CONCLUSION

We presented a new distance measure, the *pq*-gram distance, for ordered labeled trees as an effective and efficient approximation for the fanout weighted tree edit distance. The *pq*-gram distance provides a lower bound for the fanout weighted tree edit distance. We proposed a normalization that preserves the triangle inequality and makes the *pq*-gram distance a pseudo-metric, and we provided an algorithm for the computation of *pq*-grams in $O(n)$ time, where n is the number of tree nodes. The *pq*-gram distance can be computed in $O(n \log n)$ time. We discussed a scalable implementation using an interval representation of trees in a relational database.

Detailed experiments on real and synthetic data confirmed that the *pq*-gram distance is orders of magnitude faster than the tree edit distance for large trees. The accuracy of the *pq*-gram distance for real world data was clearly better than other approximations of the tree edit distance.

In the future it would be interesting to extend the *pq*-gram technique to non-hierarchical data, such as graphs that represent social networks. It would also be interesting to apply approximate matching to labels as well and combine approximate structure and label matching.

ACKNOWLEDGMENTS

The work has been done in the framework of the project *eBZ – Digital City*, which is funded by the Municipality of Bozen-Bolzano. We wish to thank our colleagues at the municipality, in particular Franco Barducci, Walter Costanzi, Roberto Loperfido, and Danila Sartori.

REFERENCES

Al-Khalifa, S., Jagadish, H. V., Patel, J. M., Wu, Y., Koudas, N., and Srivastava, D. 2002. Structural joins: A primitive for efficient XML query pattern matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE Computer Science Press, San Jose, California, 141–152.

Augsten, N., Böhlen, M., Dyreson, C., and Gamper, J. 2008. Approximate joins for data-centric XML. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Cancún, Mexico, 814–823.

Augsten, N., Böhlen, M., and Gamper, J. 2004. Reducing the integration of public administration databases to approximate tree matching. In *Electronic Government – Third International Conference*, R. Traunmüller, Ed. Lecture Notes in Computer Science 3183. Springer, Zaragoza, Spain, 102–107.

Augsten, N., Böhlen, M., and Gamper, J. 2005. Approximate matching of hierarchical data using *pq*-grams. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. ACM Press, Trondheim, Norway, 301–312.

Bruno, N., Koudas, N., and Srivastava, D. 2002. Holistic twig joins: Optimal XML pattern matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, Madison, Wisconsin, 310–321.

Celko, J. 1994. Trees, databases and SQL. *Database Programming and Design 7,* 10 (Sept.), 48–57.

Celko, J. 2004. *Trees and Hierarchies in SQL for Smarties.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Chawathe, S. S., Rajaraman, A., Garcia-Molina, H., and Widom, J. 1996. Change detection in hierarchically structured information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* ACM Press, Montreal, Canada, 493–504.

Chen, W. 2001. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms 40,* 2 (Aug.), 135–158.

Cobéna, G., Abiteboul, S., and Marian, A. 2002. Detecting changes in XML documents. In *Proceedings of the International Conference on Data Engineering (ICDE).* IEEE Computer Science Press, San Jose, California, 41–52.

Dalamagas, T., Cheng, T., Winkel, K.-J., and Sellis, T. 2006. A methodology for clustering XML documents by structure. *Information Systems 31,* 3 (May), 187–228.

DeHaan, D., Toman, D., Consens, M. P., and Özsu, M. T. 2003. A comprehensive XQuery to SQL translation using dynamic interval encoding. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* ACM, San Diego, California, 623–634.

Demaine, E. D., Mozes, S., Rossman, B., and Weimann, O. 2007. An optimal decomposition algorithm for tree edit distance. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP 2007).* Lecture Notes in Computer Science, vol. 4596. Springer, Wroclaw, Poland, 146–157.

Flesca, S., Manco, G., Masciari, E., Pontieri, L., and Pugliese, A. 2005. Fast detection of XML structural similarity. *IEEE Transactions on Knowledge and Data Engineering (TKDE) 17,* 2 (Feb.), 160–175.

Garofalakis, M. and Kumar, A. 2005. XML stream processing using tree-edit distance embeddings. *ACM Transactions on Database Systems 30,* 1, 279–332.

Gravano, L., Ipeirotis, P. G., Jagadish, H. V., Koudas, N., Muthukrishnan, S., and Srivastava, D. 2001. Approximate string joins in a database (almost) for free. In *Proceedings of the International Conference on Very Large Databases (VLDB).* Morgan Kaufmann Publishers Inc., Roma, Italy, 491–500.

Grust, T. 2002. Accelerating XPath location steps. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* ACM, Madison, Wisconsin, 109–120.

Guha, S., Jagadish, H. V., Koudas, N., Srivastava, D., and Yu, T. 2002. Approximate XML joins. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* ACM Press, Madison, Wisconsin, 287–298.

Helmer, S. 2007. Measuring the structural similarity of semistructured documents using entropy. In *Proceedings of the International Conference on Very Large Databases (VLDB).* ACM Press, Vienna, Austria, 1022–1032.

Jiang, H., Wang, W., Lu, H., and Yu, J. X. 2003. Holistic twig joins on indexed XML documents. In *Proceedings of the International Conference on Very Large Databases (VLDB).* Morgan Kaufmann Publishers Inc., Berlin, Germany, 273–284.

Jiang, T., Wang, L., and Zhang, K. 1995. Alignment of trees—an alternative to tree edit. *Theoretical Computer Science 143,* 1 (July), 137–148.

Klein, P. N. 1998. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th European Symposium on Algorithms.* Lecture Notes in Computer Science, vol. 1461. Springer, Venice, Italy, 91–102.

Lee, K.-H., Choy, Y.-C., and Cho, S.-B. 2004. An efficient algorithm to compute differences between structured documents. *IEEE Transactions on Knowledge and Data Engineering (TKDE) 16,* 8 (Aug.), 965–979.

Navarro, G. 2001. A guided tour to approximate string matching. *ACM Computing Surveys 33,* 1, 31–88.

Nierman, A. and Jagadish, H. V. 2002. Evaluating structural similarity in XML documents. In *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB).* Madison, Wisconsin, USA.

Ohkura, N., Hirata, K., Kuboyama, T., and Harao, M. 2005. The q-gram distance for ordered unlabeled trees. In *Proceedings of the International Conference on Discovery Science (DS),*

A. G. Hoffmann, H. Motoda, and T. Scheffer, Eds. Lecture Notes in Computer Science, vol. 3735. Springer, 189–202.

O'Neil, P., O'Neil, E., Pal, S., Cseri, I., Schaller, G., and Westbury, N. 2004. ORDPATHs: Insert-friendly XML node labels. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, Paris, France, 903–908.

Polyzotis, N., Garofalakis, M., and Ioannidis, Y. 2004. Approximate XML query answers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, Paris, France, 263–274.

Puhlmann, S., Weis, M., and Naumann, F. 2006. XML duplicate detection using sorted neighborhoods. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. Lecture Notes in Computer Science, vol. 3896. Springer, Munich, Germany, 773–791.

Ribeiro, L. and Härder, T. 2008. Evaluating performance and quality of XML-based similarity joins. In *Advances in Databases and Information Systems (ADBIS)*. Lecture Notes in Computer Science, vol. 5207. Springer, Pori, Finland, 246–261.

Sanz, I., Mesiti, M., Guerrini, G., and Berlanga, R. 2008. Fragment-based approximate retrieval in highly heterogeneous XML collections. *Data & Knowledge Engineering 64,* 1 (Jan.), 266–293.

Selkow, S. M. 1977. The tree-to-tree editing problem. *Information Processing Letters 6,* 6 (Dec.), 184–186.

Tai, K.-C. 1979. The tree-to-tree correction problem. *Journal of the ACM 26,* 3 (July), 422–433.

Tatarinov, I., Viglas, S., Beyer, K. S., Shanmugasundaram, J., Shekita, E. J., and Zhang, C. 2002. Storing and querying ordered XML using a relational database system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, Madison, Wisconsin, 204–215.

Ukkonen, E. 1992. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science 92,* 1 (Jan.), 191–211.

Valiente, G. 2001. An efficient bottom-up distance between trees. In *Proceedings of the 8th Symposium on String Processing and Information Retrieval*. IEEE Computer Science Press, Laguna de San Rafael, Chile, 212–219.

van Rijsbergen, C. J. 1979. *Information Retrieval*, 2nd ed. Butterworth-Heinemann, UK.

Weis, M. and Naumann, F. 2005. DogmatiX tracks down duplicates in XML. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, Baltimore, Maryland, USA, 431–442.

Yang, R., Kalnis, P., and Tung, A. K. H. 2005. Similarity evaluation on tree-structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, Baltimore, Maryland, USA, 754–765.

Yang, W. 1991. Identifying syntactic differences between two programs. *Software—Practice & Experience 21,* 7 (July), 739–755.

Yianilos, P. N. 1991,2002. Normalized forms for two common metrics. Tech. rep., NEC Research Institute.

Zezula, P., Amato, G., Dohnal, V., and Batko, M. 2006. *Similarity Search—The Metric Space Approach*. Advances in Database Systems, vol. 32. Springer, Secaucus, NJ, USA.

Zhang, C., Naughton, J. F., DeWitt, D. J., Luo, Q., and Lohman, G. M. 2001. On supporting containment queries in relational database management systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM Press, Santa Barabara, California, 425–436.

Zhang, K. 1995. Algorithms for the constrained editing distance between ordered labeled trees and related problems. *Pattern Recognition 28,* 3, 463–474.

Zhang, K. and Shasha, D. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing 18,* 6, 1245–1262.