Approximate Matching of Hierarchical Data Using *pq*-Grams

Nikolaus Augsten Michael Böhlen Johann Gamper

Free University of Bozen-Bolzano
Dominikanerplatz 3, Bozen
Italy
{augsten,boehlen,gamper}@inf.unibz.it

Abstract

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. As a running example we use residential address information. Addresses are hierarchical structures and are present in many databases. Often they are the best, if not only, relationship between autonomous data sources. Typically the matching has to be approximate since the representations in the sources differ.

We propose *pq*-grams to approximately match hierarchical information from autonomous sources. We define the *pq*-gram distance between ordered labeled trees as an effective and efficient approximation of the well-known tree edit distance. We analyze the properties of the *pq*-gram distance and compare it with the edit distance and alternative approximations. Experiments with synthetic and real world data confirm the analytic results and the scalability of our approach.

1 Introduction

When integrating data from autonomous sources, exact matches of data items representing the same real world object often fail due to missing global keys and different data representations. Approximate matching techniques must be applied instead. We focus on hierarchical data, where, in addition to data values, the data structure must also be considered.

As a running example we use an application from our local municipality. The GIS Office wants to relate data about apartments stored in different databases and display this information on a map. This requires a join on the address attributes. An equality join gives extremely poor results, mainly due to the different street names in various databases. Street names vary because different conventions are used to represent them. They may even be stored in different languages, which prevents the use of standard string comparison techniques. To overcome this problem we exploit the hierarchical organization of addresses. Instead of comparing street names we look for similarities in the hierarchical structure imposed by the addresses of a street.

Hierarchical data can be represented as ordered labeled trees. Data is then matched based on similarities of the corresponding trees. A well-known measure for comparing trees is the tree edit distance. It is computationally very expensive and leads to a prohibitively high run time. We propose the *pq*-gram distance as an effective and efficient approximation of the tree edit distance. The *pq*-grams of a tree are all its subtrees of a particular shape. Intuitively, two trees are close to each other if they have many *pq*-grams in common. For a pair of trees the *pq*-gram distance can be computed in $O(n \log n)$ time and $O(n)$ space, where n is the number of tree nodes.

In general, the *pq*-gram distance is a good approximation of the tree edit distance. In contrast to the tree edit distance, it places more emphasis on modifications to the structure of the tree. For example, deletions of nodes with a rich structure (many descendants) are more expensive than deletions of nodes with a poor structure (e.g., leaf nodes). We show that this property yields intuitive results.

At a technical level, our contribution is a new approximation for the tree edit distance with *pq*-grams.

Proceedings of the 31st VLDB Conference,
Trondheim, Norway, 2005

We present an algorithm to compute the *pq*-gram distance in $O(n \log n)$ time and $O(n)$ space, and we show its scalability to large trees stored in a relational database. A core feature of the *pq*-gram distance is its sensitivity to structural changes. This sets it apart from other approximations. Our analytical results are confirmed by experiments on both, synthetic and real data.

In the following section we describe the application scenario at our local municipality and give a problem definition. In Section 3 we discuss related work. We define the *pq*-gram distance in Section 4. In Section 5 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. We analyze properties of the *pq*-gram distance in Section 6. In Section 7 we evaluate the efficiency and effectiveness of our method on synthetic and real world data and compare it to other approximations. We draw conclusions in Section 8.

2 Problem Definition

As a running example we use an application and data from the Municipality of Bozen. The GIS office in the municipality maintains maps of the city area. It would like to enrich the maps with information retrieved from various databases of the municipality as well as external institutions. Residential addresses turn out to play a pivotal role in this process since they have to be used to access and link relevant information.

Whenever we join on address attributes, we have to know which streets correspond to each other in the joined tables. As an example consider the streets in the databases of the Registration Office (SRO) and the Land Register (SLR) shown in Figure 1. The exact join on the street names

$$\texttt{SRO} \bowtie [\texttt{SRO}.street = \texttt{SLR}.street]\ \texttt{SLR}$$

yields poor results since street names are different in different databases due to spelling mistakes, different naming conventions, and renamed streets which are not always updated in all databases. Moreover, in the bilingual region of Bozen two names for each street exist, and they are used interchangeably. A join on the street identifiers is not possible, as they are different in each system. In practice there is no central registry for residential addresses which maintains common keys for street names or addresses.

In order to improve the results we exploit the information about the streets that is stored in the address tables RO and LR (see Figure 2) that reference the streets in SRO and SLR, respectively. The addresses from a street are then organized into hierarchies and can be represented in a so-called address tree [2]. Figure 3 shows the address trees for the framed addresses in Figure 2. 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. A complete address is the path from the root to any leaf node. For example, the tuple $(30, 2, \texttt{A}, \texttt{-})$ of table RO represents the address 'Giuseppe-Cesare-Abba-Str. 2A' and corresponds to the shaded path in Figure 3. We omit unnecessary empty values ("-") in the address trees.

SRO

id	*street*
30	Giuseppe-Cesare-Abba-Str.
120	Sebastian-Altmann-Str.
5220	Bozner-Boden-Str.
3000	Hermann-von-Gilm-Str.
3030	Pater-Reginaldo-Giuliani-Str.
3540	Italienallee
4440	Musterplatzl
7180	Raffaello-Sernesi-Galerie
7590	Telsergalerie
7620	Friedensplatz
7650	Turiner Str.
7740	Trienter Str.
7860	Triester Str.
8580	Walther-v.-d.-Vogelweide-Pl.
3930	Giannantonio-Manci-Str.
...	

SLR

id	*street*
91	CESARE ABBA STRASSE
74	S. ALTMANN STRASSE
33	BOZNER BODENWEG
109	GILMWEG
185	P. R. GIULIANI STR.
115	ITALIENSTRASSE
165	MUSTERPLATZ
207	SERNESIDURCHGANG
259	TELSERDURCHGANG
139	SIEGESPLATZ
266	TURINER STRASSE
262	TRIENTER STRASSE
263	TRIESTER STRASSE
285	WALTHERPLATZ
86	MANCISTRASSE
...	

Figure 1: Street names in different departments.

RO

id	*num*	*entr*	*apt*	*resident*
30	1	-	1	Pichler
30	1	-	3	Rieder
30	2	A	-	Maier
30	2	B	1	Rossi
30	2	B	2	Woelk
30	2	B	3	Verdi
30	2	B	4	Verdi
30	2	C	-	Burger
30	3	-	-	Hofer
30	4	A	1	Tribus
30	4	A	2	Palermo
30	4	A	3	Palermo
30	4	B	-	Abel
30	4	C	-	Rossi
30	6	-	-	Spiro
120	3	A	1	Spiro
120	3	A	2	Barducci
120	3	A	3	Costanzi
120	3	A	4	Pichler
120	3	A	5	Spiro
120	3	A	6	Raifer
...				

LR

id	*num*	*entr*	*apt*	*owner*
91	1	-	1	Maier
91	1	-	2	Rossi
91	1	-	3	Sparber
91	2	A	-	Maier
91	2	B	1	Totti
91	2	B	2	Bracco
91	2	B	3	Mair
91	2	B	4	Lun
91	2	D	-	Tribus
91	3	-	-	Costanzi
91	4	A	-	Palermo
91	4	B	-	Abel
91	4	C	-	Rossi
91	6	-	-	Spiro
74	3	A	1	Spiro
74	3	A	2	Barducci
74	3	A	3	Costanzi
74	3	A	4	Spiro
74	3	A	6	Spiro
74	3	A	7	Hofer
74	4	-	-	Mueller
...				

Figure 2: Addresses stored in different departments.

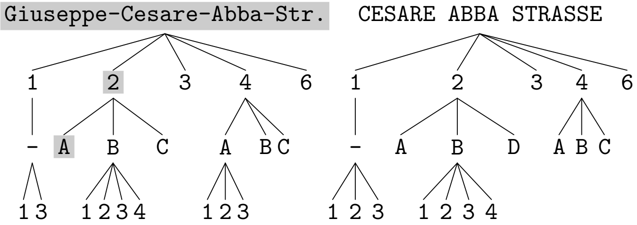


Figure 3: Address trees of streets 30 from RO and 91 from LR.

With address trees in place, we are able to compare entire address trees so as to match street names of different databases. Intuitively, two streets are identical if they have (almost) the same address tree. We use this to formulate the original join as an *approximate tree join*

$$\texttt{SRO} \bowtie [\mathrm{dist}(\mathbf{T}(\texttt{SRO}.id), \mathbf{T}(\texttt{SLR}.id)) \leq \tau] \ \texttt{SLR}.$$

Here $\mathbf{T}(id)$ are the address trees of the streets, $\mathrm{dist}(\mathbf{T}_1, \mathbf{T}_2)$ is the distance between trees $\mathbf{T}_1$ and $\mathbf{T}_2$, and τ is a distance threshold. The equality match between street names has been replaced by an approximate matching of the corresponding address trees.

Our goal is to find an effective approximation for the tree edit distance that can be efficiently computed and is scalable to large trees.

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 label change) that transforms one tree into another [19]. Zhang and Shasha [24] present an algorithm to compute the tree edit distance in $O(n^2 \min^2(l, d))$ time and $O(n^2)$ space for trees with n nodes, l leaves, and depth d. Other algorithms were presented in more recent works [7, 14]. All of them have more than $O(n^2)$ runtime complexity 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: Alignment distance [13], isolated subtree distance [20], and top-down distance [18, 22] have runtime at least $O(n^2)$, bottom-up distance can be computed in $O(n)$ time. Bottom-up distance tries to find the largest possible common subtrees of two trees, starting with the leaf nodes. It is very sensitive to differences between the leaf nodes. If the leaves are different, the inner nodes are never compared. This makes the bottom-up distance applicable in only very specific domains.

Guha et al. [11] present a framework for approximate XML joins based on tree edit distance, where XML documents are represented as ordered labeled trees. They give upper and lower bounds for the tree edit distance that can be computed in $O(n^2)$ time and use reference sets to take advantage of the fact that the tree edit distance is a metric, thus reducing the actual number of distances to compute in a join. The success of this method depends heavily on a good choice of the reference set. We do not try to limit the number of distance calculations with the expensive tree edit distance, rather we substitute it with an efficient approximation.

Chawathe et al. [6] use a variant of the tree edit distance for change detection. Lee et al. [15] tune the algorithm presented by Chawathe et al. to XML documents. Both algorithms first compute a match between the nodes of the trees, and based on this the distance is computed in $O(ne)$ time, where e is the edit distance between the trees. Whereas in a change detection scenario typically trees with small differences are compared, for joins the distances between all pairs of trees have to be computed. For trees that are very different the edit distance e is $O(n)$, which yields $O(n^2)$ runtime for both algorithms.

A core operation in XML query processing is to find all occurrences of a twig pattern [3, 12]. 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. [17] 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.

Garofalakis and Kumar [9] 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 computes an approximation of the tree edit distance with subtree move (to within a $O(log^2 n \times log^* n)$ factor) in $O(n \times log^* n)$ time and $O(n)$ space[1]. We implement this approximation and empirically compare it to the *pq*-gram distance. The tree embedding distance gives less weight to structural changes than the tree edit distance. The sensitivity of the *pq*-gram distance to structural changes is controlled by the parameters p and q. The *pq*-gram distance typically weights them more than the edit distance.

Navarro [16] gives a good overview of the edit distance for *strings* and its variants. Ukkonen [21] 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. [10] present algorithms for approximate string joins based on 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, string matching fails completely.

4 THE *pq*-GRAM DISTANCE

Hierarchical data can be represented as rooted, ordered, labeled trees, where the single data values are represented as labels of the tree nodes. In this section we first give a definition of trees and then define the *pq*-gram distance of trees.

[1] $\log^* n$ denotes the number of log applications required to reduce n to a quantity that is ≤ 1, cf. [9].

4.1 Preliminaries

Let $G = (V, E)$ be a graph with *nodes* $V(G) = V$ and edges $E(G) = E$. A *tree* $\mathbf{T}$ is a directed, acyclic, connected, non-empty graph. An *edge* is an ordered pair (p, c), where $\mathsf{p}, \mathsf{c} \in V(\mathbf{T})$ are nodes, and p is the *parent* of c. Nodes with the same parent are *siblings*. An order $\leq$ is defined on the nodes, and this order is total among siblings. The siblings $\mathsf{s}_1 \leq \mathsf{s}_2$ ($\mathsf{s}_1 \neq \mathsf{s}_2$) are *contiguous* if s_1 and s_2 have no sibling x ($\mathsf{s}_1 \neq \mathsf{x} \neq \mathsf{s}_2$) with $\mathsf{s}_1 \leq \mathsf{x} \leq \mathsf{s}_2$. Node c is the i-th *child* of p with $i = |\{\mathsf{x} \in V(\mathbf{T}) | (\mathsf{p}, \mathsf{x}) \in E(\mathbf{T}), \mathsf{x} \leq \mathsf{c}\}|$. The number of p's children is its *fanout* f_{p}. The node with no parent is the *root* node $\mathsf{r} = \mathrm{root}(\mathbf{T})$, and a node without children is a *leaf*.

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. d is a *descendant* of v if v is an ancestor of d. The *level* of a node $\mathrm{level}(\mathsf{v})$ is the length of the path from the root to v, the *depth* of a tree $\mathrm{depth}(\mathbf{T})$ is the length of the longest path from the root to any one of the leaves.

A *label* is a symbol $\sigma \in \mathbf{\Sigma}$, where $\mathbf{\Sigma}$ is a finite alphabet. Each node $\mathsf{v} \in V(\mathbf{T})$ has assigned a label $\mathrm{l}(\mathsf{v})$. A node o with the special label $\mathrm{l}(\mathsf{o}) = *$ is a *null node*.

In our graphical representation of trees we represent nodes as an (identifier, label)-pair, the edges are lines between the nodes, and siblings are ordered from left to right. Whenever possible we omit the identifiers of the nodes to avoid clutter (e.g., in Figure 3).

Example 4.1 *Figure 4 shows a tree* $\mathbf{T}_1 = (V, E)$ *with* $V = \{\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4, \mathsf{v}_5, \mathsf{v}_6\}$, $E = \{(\mathsf{v}_1, \mathsf{v}_2), (\mathsf{v}_1, \mathsf{v}_5), (\mathsf{v}_1, \mathsf{v}_6), (\mathsf{v}_2, \mathsf{v}_3), (\mathsf{v}_2, \mathsf{v}_4)\}$, *and the order* $\mathsf{v}_2 \leq \mathsf{v}_5 \leq \mathsf{v}_6$, $\mathsf{v}_3 \leq \mathsf{v}_4$. v_1 *has 3 children, where* v_2 *is the first,* v_5 *the second, and* v_6 *the third child. The root node* $\mathrm{root}(\mathbf{T}) = \mathsf{v}_1$. v_1 *is the ancestor of all other nodes.* v_3, v_4, v_5 *and* v_6 *are leaf nodes. The node labels of our example tree are* $\mathrm{l}(\mathsf{v}_1) = \mathsf{a}$, $\mathrm{l}(\mathsf{v}_2) = \mathsf{a}$, $\mathrm{l}(\mathsf{v}_3) = \mathsf{e}$, $\mathrm{l}(\mathsf{v}_4) = \mathsf{b}$, $\mathrm{l}(\mathsf{v}_5) = \mathsf{b}$, *and* $\mathrm{l}(\mathsf{v}_6) = \mathsf{c}$.

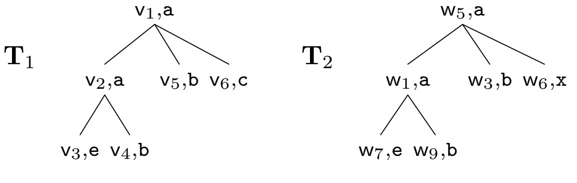


Figure 4: Graphical representation of trees.

A *subtree* $\mathbf{S} \subseteq \mathbf{T}$ is a tree with $V(\mathbf{S}) \subseteq V(\mathbf{T})$ and $E(\mathbf{S}) \subseteq E(\mathbf{T})$, retaining the node order. A *preorder traversal* of a tree visits the root node first, and then recursively traverses all the subtrees rooted in its children in preorder, preserving the children's order. We call a node v the i-th node of $\mathbf{T}$ in preorder if v is visited as the i-th node in a preorder traversal.

Two trees $\mathbf{T}$ and $\mathbf{T}'$ are *isomorphic* if there is a bijective mapping m between the nodes $V(\mathbf{T})$ and $V(\mathbf{T}')$ such that the following holds true: (v, w) is an edge of $\mathbf{T}$ and w is the i-th child of v if and only if $(m(\mathsf{v}), m(\mathsf{w}))$ is an edge of $\mathbf{T}'$ and $m(\mathsf{w})$ is the i-th child of $m(\mathsf{v})$.

Example 4.2 *Consider Figure 4. The tree* $\mathbf{S}_1 = (\{\mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4\}, \{(\mathsf{v}_2, \mathsf{v}_3), (\mathsf{v}_2, \mathsf{v}_4)\}), \mathsf{v}_3 \leq \mathsf{v}_4$ *is a subtree of* $\mathbf{T}_1$. *The preorder traversal of* $\mathbf{T}_1$ *visits the nodes in the following order:* $\mathsf{v}_1, \mathsf{v}_2, \mathsf{v}_3, \mathsf{v}_4, \mathsf{v}_5, \mathsf{v}_6$. *Tree* $\mathbf{T}_2$ *is isomorphic to* $\mathbf{T}_1$ *with* $m = \{(\mathsf{v}_1, \mathsf{w}_5), (\mathsf{v}_2, \mathsf{w}_1), (\mathsf{v}_3, \mathsf{w}_7), (\mathsf{v}_4, \mathsf{w}_9), (\mathsf{v}_5, \mathsf{w}_3), (\mathsf{v}_6, \mathsf{w}_6)\}$.

4.2 The pq-Gram Distance

In the following paragraphs we define the notion of *pq*-grams and a distance measure based on *pq*-grams. 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 of the *pq*-grams, we extend the tree with *null nodes*. The *pq*-grams are then defined as subtrees of the extended tree.

Definition 4.1 (*pq*-Extended Tree) *Let* $\mathbf{T}$ *be a tree, and* $p > 0$ *and* $q > 0$ *be two integers. The* *pq*-extended tree, $\mathbf{T}^{pq}$, *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 null nodes that do not occur in* $\mathbf{T}$.

Example 4.3 *Figure 5 shows the graphical representation of* $\mathbf{T}_1^{2,3}$, *the* $2,3$*-extended tree of our example tree* $\mathbf{T}_1$.

Definition 4.2 (*pq*-Gram Pattern) *For* $p > 0$ *and* $q > 0$, *the* *pq*-gram pattern *is a tree that consists of an anchor node with* $p - 1$ *ancestors and* q *children.*

Example 4.4 *An example of a* $2,3$*-gram pattern is the tree* $(\{\mathsf{p}_1, \mathsf{p}_2, \mathsf{p}_3, \mathsf{p}_4, \mathsf{p}_5\}, \{(\mathsf{p}_1, \mathsf{p}_2), (\mathsf{p}_2, \mathsf{p}_3), (\mathsf{p}_2, \mathsf{p}_4), (\mathsf{p}_2, \mathsf{p}_5)\}), \mathsf{p}_3 \leq \mathsf{p}_4 \leq \mathsf{p}_5$. p_2 *is the anchor node, and it has* 1 *ancestor* (p_1) *and* 3 *children* (p_3, p_4, *and* p_5).

Definition 4.3 (*pq*-Gram) *For* $p > 0$ *and* $q > 0$, *a* *pq*-gram $\mathbf{G}$ *of a tree* $\mathbf{T}$ *is defined as a subtree of the extended tree* $\mathbf{T}^{pq}$ *with the following properties:* $\mathbf{G}$ *is isomorphic to the pq-gram pattern, and contiguous siblings in* $\mathbf{G}$ *are contiguous siblings in* $\mathbf{T}^{pq}$.

Definition 4.4 (Label-tuple) *Let* $\mathbf{G}$ *be a pq-gram with the nodes* $V(\mathbf{G}) = \{\mathsf{v}_1, \ldots, \mathsf{v}_p, \mathsf{v}_{p+1}, \ldots, \mathsf{v}_{p+q}\}$, *where* v_i *is the i-th node in preorder. The tuple* $\mathrm{l}(\mathbf{G}) = (\mathrm{l}(\mathsf{v}_1), \ldots, \mathrm{l}(\mathsf{v}_p), \mathrm{l}(\mathsf{v}_{p+1}), \ldots, \mathrm{l}(\mathsf{v}_{p+q}))$ *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.

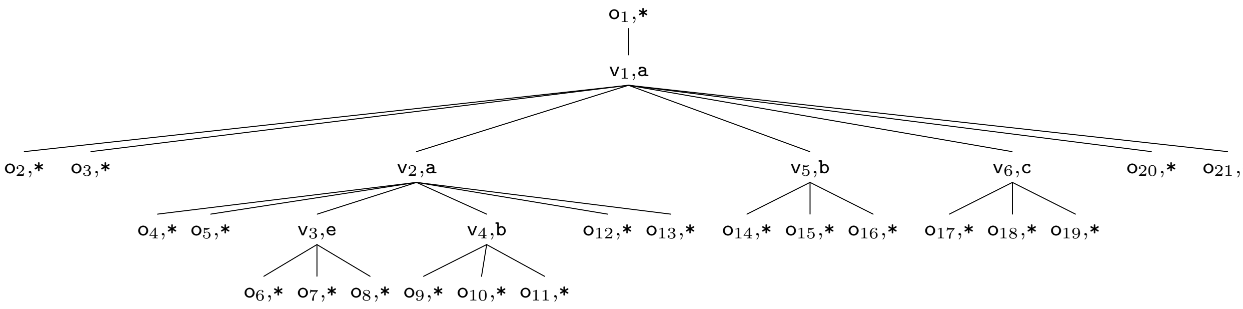


Figure 5: Graphical representation of the extended tree $\mathbf{T}_1^{2,3}$.

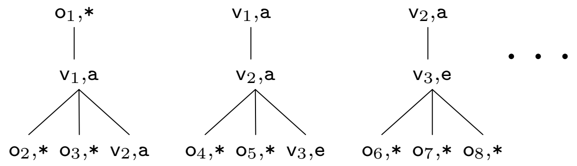


Figure 6: Some of the 2, 3-grams of $\mathbf{T}_1$.

Example 4.5 *Figure 6 shows some of the* 2, 3*-grams of the example tree* $\mathbf{T}_1$*. They are constructed by moving the* 2, 3*-gram pattern over the extended tree* $\mathbf{T}_1^{2,3}$ *(see Figure 5). We start at the top of the tree. For the first pq-gram the anchor node of the pattern is mapped to* v_1*, and the children of the anchor are mapped to two null nodes and* v_2*. The corresponding label-tuple is* $(*, \mathsf{a}, *, *, \mathsf{a})$.

Definition 4.5 (*pq*-Gram Profile) *For* $p > 0$ *and* $q > 0$*, the* pq-gram profile, $\mathrm{P}^{p,q}(\mathbf{T})$*, of a tree* $\mathbf{T}$ *is defined as the bag of label-tuples* $\mathrm{l}(\mathbf{G}_i)$ *of all pq-grams* $\mathbf{G}_i$ *of* $\mathbf{T}$.

The tables in Figure 7 show the 2, 3-gram profile of $\mathbf{T}_1$ and $\mathbf{T}_2$, respectively. Note that *pq*-grams might appear more than once in a *pq*-gram profile, e.g., $(\mathsf{a}, \mathsf{b}, *, *, *)$ appears twice in the profile of $\mathbf{T}_1$.

$\mathrm{P}^{2,3}(\mathbf{T}_1)$ *labels*	$\mathrm{P}^{2,3}(\mathbf{T}_2)$ *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, x)
(a, c, *, *, *)	(a, x, *, *, *)
(*, a, b, c, *)	(*, a, b, x, *)
(*, a, c, *, *)	(*, a, x, *, *)

Figure 7: 2, 3-Gram profiles of $\mathbf{T}_1$ and $\mathbf{T}_2$.

We subsequently define the *pq*-gram distance as a measure for the similarity of two trees. The *pq*-gram distance is based on the number of *pq*-grams that the profiles of the compared trees have in common.

Definition 4.6 (*pq*-Gram Distance) *For* $p > 0$ *and* $q > 0$*, the* pq-gram distance, $\Delta^{p,q}(\mathbf{T}_1, \mathbf{T}_2)$*, between two trees* $\mathbf{T}_1$ *and* $\mathbf{T}_2$ *is defined as follows:*

$$\Delta^{p,q}(\mathbf{T}_1, \mathbf{T}_2) \;=\; 1 \;-\; 2\frac{|\,\mathrm{P}^{p,q}(\mathbf{T}_1) \cap \mathrm{P}^{p,q}(\mathbf{T}_2)|}{|\,\mathrm{P}^{p,q}(\mathbf{T}_1) \cup \mathrm{P}^{p,q}(\mathbf{T}_2)|} \qquad (1)$$

Example 4.6 *Consider the* 2, 3*-gram distance between* $\mathbf{T}_1$ *and* $\mathbf{T}_2$*. The corresponding* 2, 3*-gram profiles are shown in Figure 7. The bag-intersection of the two profiles is* $\{(*, \mathsf{a}, *, *, \mathsf{a}), (\mathsf{a}, \mathsf{a}, *, *, \mathsf{e}), (\mathsf{a}, \mathsf{e}, *, *, *), (\mathsf{a}, \mathsf{a}, *, \mathsf{e}, \mathsf{b}), (\mathsf{a}, \mathsf{b}, *, *, *), (\mathsf{a}, \mathsf{a}, \mathsf{e}, \mathsf{b}, *), (\mathsf{a}, \mathsf{a}, \mathsf{b}, *, *), (*, \mathsf{a}, *, \mathsf{a}, \mathsf{b}), (\mathsf{a}, \mathsf{b}, *, *, *)\}$*, which yields* $|\,\mathrm{P}^{2,3}(\mathbf{T}_1) \cap \mathrm{P}^{2,3}(\mathbf{T}_2)| = 9$*. For the cardinality of the bag-union we get* $|\,\mathrm{P}^{2,3}(\mathbf{T}_1) \cup \mathrm{P}^{2,3}(\mathbf{T}_2)| = |\,\mathrm{P}^{2,3}(\mathbf{T}_1)| + |\,\mathrm{P}^{2,3}(\mathbf{T}_2)| = 26$*. Thus, the pq-gram distance is*

$$\Delta^{2,3}(\mathbf{T}_1, \mathbf{T}_2) = 1 - 2\frac{9}{26} = 0.31.$$

The *pq*-gram distance is 1 if two trees share no *pq*-grams. Trees at distance 0 have the same *pq*-gram profile. Note that distance 0 does not imply equality of trees. An example of two different trees with the same *pq*-gram profile is shown in Figure 8. The *pq*-grams responsible for detecting the swapped children of the root nodes of T' and T'' are those anchored in the root nodes. However, as all children of the root nodes have the same label, the *pq*-grams remain unchanged.

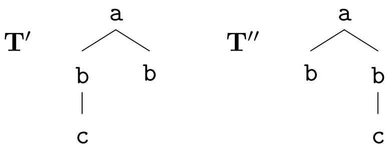


Figure 8: Different trees with the same *pq*-gram profile.

The *pq*-gram distance can be computed in $O(n \log n)$ time by computing the bag intersection of the *pq*-gram profiles of size $O(n)$. Theorem 4.1 shows, how the size of the profile is related to the number of leaf and non-leaf nodes.

Theorem 4.1 *Let $p > 0$, $q > 0$, and $\mathbf{T}$ be a tree with l leaf nodes and i non-leaf nodes. The size of the pq-gram profile is*

$$|\,\mathrm{P}^{p,q}(\mathbf{T})| = 2l + qi - 1.$$

Proof 4.1 *By structural induction:*
$|V(\mathbf{T})| = 1$*: The tree consists of the root node only, and according to Definition 4.3 the pq-gram profile contains exactly one pq-gram. The number of leaves is* 1*, while the number of non-leaf nodes is* 0*, thus* $|\,\mathrm{P}^{p,q}(\mathbf{T})| = 2l + qi - 1 = 1$.
$|V(\mathbf{T})| > 1$*: In this case $i \geq 1$ (at least the root node) and $l \geq 1$. First we delete all non-leaf nodes (except the root* r*) and get $\mathbf{T}'$. $|\,\mathrm{P}^{p,q}(\mathbf{T})| - |\,\mathrm{P}^{p,q}(\mathbf{T}')| = (i-1) * q$. (Deleting a non-leaf node decreases the cardinality of the pq-gram profile by q). The number of leaves does not change with this operation, and the tree now consists of only the leaves and the root node. Now we delete all leaf nodes and get $\mathbf{T}''$, $|\,\mathrm{P}^{p,q}(\mathbf{T}')| - |\,\mathrm{P}^{p,q}(\mathbf{T}'')| = 2(l-1) + q$. (Deleting a leaf node decreases the cardinality of the pq-gram profile by q if the leaf has no siblings, otherwise by 2). $\mathbf{T}''$ consists only of the root node and $|\,\mathrm{P}^{p,q}(\mathbf{T}'')| = 1$. This means,* $|\,\mathrm{P}^{p,q}(\mathbf{T})| = 1 + [2(l-1)+q] + [(i-1)*q] = 2l + qi - 1$.

5 Algorithms

5.1 An Algorithm for the pq-Gram-Profile

The basic idea of the *pq*-Gram-Profile algorithm in Figure 9 is to move the *pq*-gram pattern vertically and horizontally over the tree (see Figure 10a). After each move the nodes covered by the pattern form a *pq*-gram.

We use two shift registers, *anc* of size p and *sib* of size q, to represent the labels of the ancestor and the leaf nodes that are covered by the *pq*-gram pattern, respectively. 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, *anc* ∘ *sib*, is a tuple in the *pq*-gram profile, i.e., for $anc = (l_1, \ldots, l_p)$ and $sib = (l_{p+1}, \ldots, l_{p+q})$ the label-tuple of the *pq*-gram is $(l_1, \ldots, l_p, l_{p+1}, \ldots, l_{p+q})$.

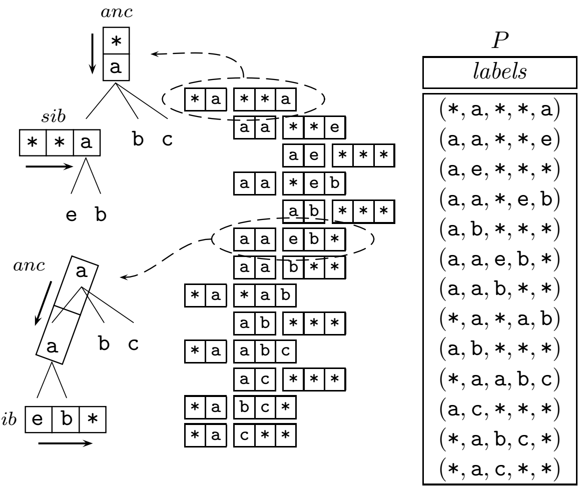
pq-Gram-Profile takes as input a tree $\mathbf{T}$ and the two values p and q and returns a relation that contains the *pq*-gram profile of $\mathbf{T}$. After the initialization, profile calculates the *pq*-grams starting from the root node of $\mathbf{T}$. First profile shifts the label of anchor node r into the register *anc*, which corresponds to moving the *pq*-gram pattern one step down. Now *anc* contains the labels of r and its $p-1$ ancestors. The loop at line 13 moves the register *sib* from left to right over the children of r in order to produce all the *pq*-grams with anchor point r and calls profile recursively for each child of r. Overall, profile adds $f_{\mathsf{r}} + q - 1$ label-tuples to P for each non-leaf node r, and 1 label-tuple for each leaf node. The *pq*-extended tree is calculated on the fly by an adequate initialization of the shift registers (lines 3, 8, 17-19).

```
1   pq-GRAM-PROFILE($\mathbf{T}, p, q$)
2       $P$ : empty relation with schema ($labels$)
3       $anc$: shift register of size p (filled with $*$)
4       $P$ = PROFILE($\mathbf{T}, p, q, P$, root($\mathbf{T}$), $anc$)
5       return $P$

6   PROFILE($\mathbf{T}, p, q, P$, r, $anc$)
7       $anc$ := shift($anc$, l(r))
8       $sib$: shift register of size q (filled with $*$)
9
10      if r is a leaf then
11          $P := P \cup (anc \circ sib)$
12      else
13          for each child c (from left to right) of r do
14              $sib$ := shift($sib$, l(c))
15              $P := P \cup (anc \circ sib)$
16              $P$ :=PROFILE($\mathbf{T}, p, q, P$, c, $anc$)
17          for $k := 1$ to $q - 1$
18              $sib$ := shift($sib$, $*$)
19              $P := P \cup (anc \circ sib)$
20
21      return $P$
```

Figure 9: Calculating the *pq*-gram profile of a tree.

Example 5.1 *Assume $p = 2$, $q = 3$, and the tree $\mathbf{T}_1$ from Figure 4. The main data structures of the* profile *algorithm are visualized in Figure 10. After the initialization,* profile$(\mathbf{T}_1, 2, 3, \{\}, \mathsf{v}_1, (*, *))$ *is called.*



(a) (b) (c)

Figure 10: (a) Moving the *pq*-gram pattern in the tree, (b) Shift registers *anc* and *sib*, (c) Relation P produced by profile.

Line 7 shifts the label of v_1 *into the register anc, yielding* $anc = (*, \mathsf{a})$*, and line 8 initializes* $sib = (*, *, *)$*. Since* v_1 *is not a leaf we enter the loop at line 13 and process all children of* v_1*. The label of the first child,* v_2*, is shifted into sib, yielding* $sib = (*, *, \mathsf{a})$*, and the first label-tuple* $(*, \mathsf{a}, *, *, \mathsf{a})$ *is added to the result set* P*. Figure 10b shows the values of anc and sib each time a label-tuple is added to* P*. The indentation illustrates the recursion. The table in Figure 10c shows the result relation* P *with the label-tuples in the order in which they are produced by the algorithm.*

pq-Gram-Profile has runtime complexity $O(n)$ for a tree $\mathbf{T}$, where $n = |V(\mathbf{T})|$: Each recursive call of profile processes one node, and each node is processed exactly once.

5.2 Relational Implementation

The algorithm described above requires no particular encoding of trees. This section gives a scalable implementation for trees stored in a relational database. We use an interval representation of trees, where each node of a tree is represented by a pair of numbers (interval). The interval encoding is a technique for storing hierarchical data in relations [4, 5] and has been used to store and query XML data [1, 8, 23].

We associate a unique index number to each tree in the set. Each node of a tree is then represented as a quadruple of tree index, node label, and left and right endpoint of the node's interval.

Definition 5.1 (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* $(id(\mathbf{T}), \mathrm{l}(\mathsf{v}), lft, rgt)$*;* $id(\mathbf{T})$ *is a unique identifier of the tree* $\mathbf{T}$*,* $\mathrm{l}(\mathsf{v})$ *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* d*, and* $(id(\mathbf{T}), \mathrm{l}(\mathsf{a}), lft_{\mathsf{a}}, rgt_{\mathsf{a}}) \in R$*, and* $(id(\mathbf{T}), \mathrm{l}(\mathsf{d}), lft_{\mathsf{d}}, rgt_{\mathsf{d}}) \in R$*,*
- $rgt_{\mathsf{v}} < lft_{\mathsf{w}}$ *if node* v *is a left sibling of node* w*, and* $(id(\mathbf{T}), \mathrm{l}(\mathsf{v}), lft_{\mathsf{v}}, rgt_{\mathsf{v}}) \in R$*, and* $(id(\mathbf{T}), \mathrm{l}(\mathsf{w}), lft_{\mathsf{w}}, rgt_{\mathsf{w}}) \in R$*,*
- $rgt = lft + 1$ *if node* v *is a leaf node, and* $(id(\mathbf{T}), \mathrm{l}(\mathsf{v}), lft, rgt) \in R$*.*

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 11 shows an address tree of our application, where each node is annotated with the endpoints of the interval.

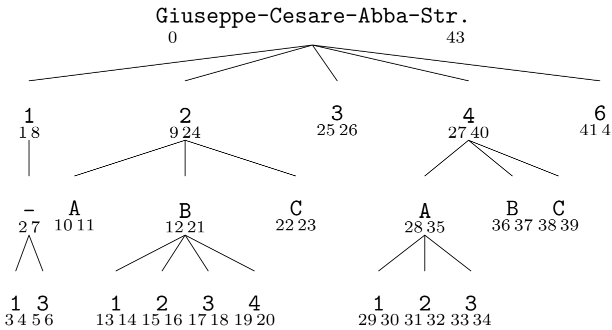


Figure 11: Address tree in interval encoding.

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

```
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 7.1 confirm the scalability of this approach to large trees.

Figure 12 shows the algorithm adapted for interval encoding with the changes highlighted. Instead of a tree *pq*-Gram-Profile gets a cursor as an argument. profile 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.

```
1    pq-Gram-Profile(cur, p, q)
2        P : empty relation with schema (labels)
3        anc: shift register of size p (filled with *)
4        P = profile(cur, p, q, P, fetch(cur), anc)
5        return P


6    profile(cur, p, q, P, r, anc)
7        anc := shift(anc, l(r))
8        sib: shift register of size q (filled with *)
9
9a       cur := next(cur)
10       if isLeaf(r) then
11           P := P ∪ (anc ∘ sib)
12       else
12a          c := fetch(cur)
13           while isDescendant(c, r) do
14               sib := shift(sib, l(c))
15               P := P ∪ (anc ∘ sib)
16               P := profile(cur, p, q, P, c, anc)
16a              c := fetch(cur)
17           for k := 1 to q − 1
18               sib := shift(sib, *)
19               P := P ∪ (anc ∘ sib)
20
21       return P
```

Figure 12: Implementation of profile using a cursor.

PROFILE calls the following two functions:

- ISLEAF(v): Returns true iff v is a leaf node, i.e., $\mathrm{lft}(\mathsf{v}) + 1 = \mathrm{rgt}(\mathsf{v})$.

- ISDESCENDANT(d, a): Returns true iff d is a descendant of a, i.e., $\mathrm{lft}(\mathsf{a}) < \mathrm{lft}(\mathsf{d})$ **and** $\mathrm{rgt}(\mathsf{a}) > \mathrm{rgt}(\mathsf{d})$ **and** $\mathrm{treeId}(\mathsf{a}) = \mathrm{treeId}(\mathsf{d})$ **and** $\mathsf{d} \neq$ `null`.

With the interval encoding it is easier to check whether a node is a descendant than whether it is a child. In our algorithm this amounts to the same thing: When the loop in line 13 is entered the first time, c is the next node after r in preorder (or `null`). Thus, if c is a descendant of r, it must be a child. The recursive call in line 16 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 r, then it is a child. Thus the while-loop in Figure 12 is equivalent to the for-loop in Figure 9.

6 Sensitivity to Structural Changes

In this section we discuss the main properties of the *pq*-gram distance and compare it with the tree edit distance. We investigate two cases where the *pq*-gram distance behaves differently from the tree edit distance: structural and local changes. We consider the following standard edit operations [24]:

Update($\mathbf{T}, \mathsf{v}, \sigma$): Updating a node $\mathsf{v} \in V(\mathbf{T})$ means changing its label to $\sigma \in \mathbf{\Sigma}$.

Delete($\mathbf{T}, \mathsf{v}$): Deleting a node $\mathsf{v} \in V(\mathbf{T}) \setminus \{\mathrm{root}(\mathbf{T})\}$ means substituting v with its children (preserving the order), i.e., remove v and connect v's children directly with v's parent node.

Insert($\mathbf{T}, \mathsf{v}, \mathsf{p}, i, k$): Inserting a new node $\mathsf{v} \notin V(\mathbf{T})$ as a child of a node $\mathsf{p} \in V(\mathbf{T})$ at position i means substituting k consecutive children $\mathsf{v}_i, \mathsf{v}_{i+1}, \ldots, \mathsf{v}_{i+k-1}$ of p with v, and inserting them as children of v (preserving the order). If $k = 0$, a leaf node is inserted, and the number of p's children increases by one.

The tree edit distance assigns a fixed cost to each operation. This disregards the fact that operations which change the structure (insert and delete) might have side effects on other nodes. For example, if a node is deleted, all children of this node are moved with their descendants to the parent node. This behavior leads to non-intuitive results, as shown in Figure 13: Tree $\mathbf{T}'$ is the result of deleting the leaves with labels g and k from $\mathbf{T}$, $\mathbf{T}''$ 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 tree edit distance is 2 in both cases for a unit cost model.

The *pq*-gram distance depends directly on the number of affected *pq*-grams, which depends on the number of descendants of v within distance p. Thus, changes to non-leaf nodes cost more than changes to leaves. The following theorem gives the number of *pq*-grams that contain a node v, which corresponds to the number of affected *pq*-grams if v is modified.

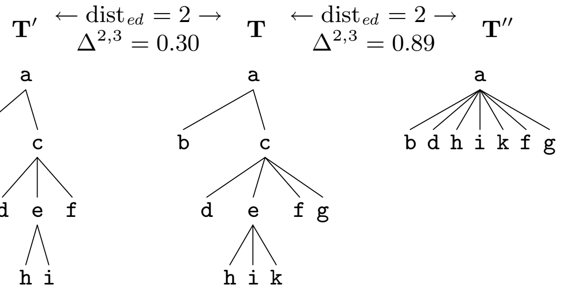


Figure 13: Tree edit distance and *pq*-gram distance for structural changes.

Theorem 6.1 *For a tree* $\mathbf{T}$ *with all leaf nodes at level* $d = \mathrm{depth}(\mathbf{T})$ *and a fixed fanout* $f > 1$ *for the non-leaf nodes, the number of pq-grams* ($p > 0, q > 0$) *that contain a node* v *of level* $l = \mathrm{level}(\mathsf{v})$ *is:*

$$\mathrm{cnt}_{pq}(\mathbf{T}, \mathsf{v}) = q\,\mathrm{sgn}(l) + \begin{cases} \frac{f^p - 1}{f-1}(f + q - 1) & \text{if } p \leq d - l \\ \frac{f^{d-l}-1}{f-1}(f + q - 1) + f^{d-l} & \text{if } p > d - l. \end{cases}$$

Proof 6.1 *Consider how the pq-gram pattern with q leaves and p non-leaves is shifted over the tree. The leaves of the pattern are shifted over all nodes of the tree but the root node, which gives q pq-grams for each non-root node (*$\mathrm{sgn}(l)$ *is* 0 *for the root,* 1 *for non-root nodes). If* v *is a non-leaf node, it appears in* $f + q - 1$ *pq-grams as the anchor node, otherwise in a single pq-gram. While* v *is in the pq-gram we recursively move the pattern down the tree. We exit the recursion earlier if the anchor node of the pq-gram pattern is a leaf. For the case* $p \leq d - l$, *we get* $(f + q - 1) \sum_{i=0}^{p-1} f^i$, *and for the case* $p > d - l$, $(f + q - 1) \sum_{i=0}^{d-l-1} f^i$ *additional pq-grams that contain* v. *For the latter case we add the term* f^{d-l} *that accounts for the pq-grams that have one of the* f^{d-l} *leaf descendants of* v *as an anchor node. We evaluate the partial sum of the geometric series to get the formula in Theorem 6.1.*

Theorem 6.1 assumes a tree with all leaves at the same depth and a fixed fanout. If f is the *maximum* fanout of v and its descendants within distance p, then $\mathrm{cnt}_{pq}(\mathbf{T}, \mathsf{v})$ is an upper bound for the number of *pq*-grams that contain v.

According to Theorem 6.1 the cost for changing a leaf node ($d = l$) is $q + 1$, i.e., depends only on q. For non-leaf nodes the impact of p is prevalent, and we can control the sensitivity of the *pq*-gram distance 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. For example, two streets with different house numbers (with subnumbers and apartment numbers) are considered more different than streets in which only apartment numbers differ.

We further investigate the case when part of a tree is missing, i.e., a subtree is deleted. The effect on the structure is limited as the remaining part of the tree is unchanged. An example of a subtree is a subnumber with all its apartment numbers. If it is missing in one address tree, a relatively high number of nodes changes. These changes should be weighted less than the same number of changes on different house numbers.

If a subtree is deleted, several modifications are applied within a small neighborhood. The affected sets of *pq*-grams overlap each other, and hence, these changes have less impact on the *pq*-gram distance than changes that are uniformly distributed over the tree. The following theorem gives the number of *pq*-grams that change with a subtree deletion.

Theorem 6.2 *Let* $\mathbf{S}$ *be the subtree of* $\mathbf{T}$ *consisting of* $\mathsf{v} \in V(\mathbf{T}) \setminus \{\mathrm{root}(\mathbf{T})\}$ *and all its descendants, and let* l *be the number of leaves of* $\mathbf{S}$*, and let* i *be the number of non-leaf nodes. If all nodes of* $\mathbf{S}$ *are deleted or updated, then* $2l + iq + q - 1$ *pq-grams change.*

Proof 6.2 *All pq-grams of the subtree change. This are* $2l + iq - 1$ *pq-grams (Theorem 4.1). Further* v *appears as a sibling in* q *pq-grams. The sum is* $2l + iq + q - 1$.

Example 6.1 *We refer to Figure 13 and discuss the deletion of the subtree of* $\mathbf{T}$ *that consists of the node with label* `e` *(lets call the node* v*) and all its descendants. An effect of this operation is that the following nodes are deleted:* v *plus the nodes labeled* `h`*,* `i`*, and* `k`*. The number of* $2,3$*-grams that contain the node* v *is* $\mathrm{cnt}_{2,3}(\mathbf{T}, \mathsf{v}) = 11$*, and* $q + 1 = 4$ *for the three other nodes. If these nodes did not share any pq-grams, the total number of affected pq-grams would be* $11 + 3 \times 4 = 23$*. However, as the deleted nodes build a subtree with* $l = 3$ *leaves and* $i = 1$ *non-leaf nodes, they do share pq-grams, and the total number of changing* $2,3$*-grams is only* $2l + iq + q - 1 = 11$.

7 Experiments

7.1 Scalability

We compare the scalability of our algorithm with the tree edit distance [24] and the tree embedding distance [9], 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 $|V(\mathbf{T}_1)| = |V(\mathbf{T}_2)| = n$, where n ranges from 3 to 2×10^6 nodes. The depth of the trees is $\log(n)$ and the labels for each tree are randomly chosen from a set of n different labels.

Figure 14(a) shows the runtimes of tree edit distance and $2,3$-gram distance calculations for different tree sizes. For the tree edit distance we use the implementation of Zhang and Shasha[2], whereas for the *pq*-gram distance we use the relational implementation described in Section 5.2. For very small trees edit distance is faster than *pq*-gram distance. The reason being that our algorithm writes all intermediate results to the disk, while the edit distance algorithm runs in the main memory. Therefore the overhead for disk access in this range masks the actual computing time for the distance. This effect can easily be prevented by keeping all data in main memory. For large trees the computation time for the tree edit distance grows very fast. For trees of size 10,000 it is already more than 27 hours, therefore we could not run our experiment for even larger trees. For the *pq*-gram distance the computation time is almost linear in the tree size.

Figure 14(b) compares the *pq*-gram distance for varying parameters with the tree embedding distance. We use our own implementation for tree embedding distance according to the algorithm of Garofalakis and Kumar [9]. For the comparison both algorithms run in main memory. The *pq*-gram distance is slightly faster, and varying values for p and q have little impact on the scalability of the *pq*-gram distance calculation.

7.2 Sensitivity to Structural Changes

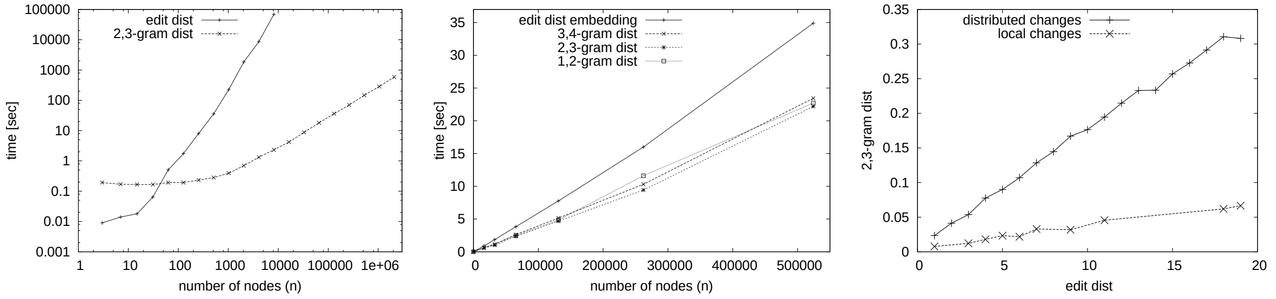
In Section 6 we point out that the *pq*-gram distance weights deletions of non-leaf nodes more than deletions of leaves, and the sensitivity to structural changes is controlled by the parameters p and q. We show this property in an experiment, where only non-leaf nodes or only leaf nodes are deleted 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 depth 6. Each non-leaf has a fanout of between 2 and 5. Figure 15 shows the *pq*-gram distance for different numbers of leaf and non-leaf 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. Sensitivity to changes in the leaves depends only on q, and we can emphasize structural sensitivity with higher values of p. For deletions of non-leaf nodes the *pq*-gram distance is longer than for deletions of leaf nodes.

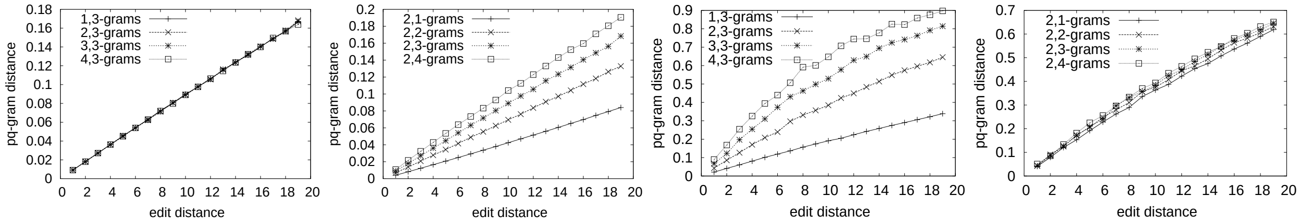
We further investigate the difference in the *pq*-gram distance for deleting a subtree or the same number of nodes randomly distributed all over the tree. For the experiment we use the same tree $\mathbf{T}$ as above. We randomly choose a node $\mathsf{v} \in \mathbf{T} \setminus \{\mathrm{root}(\mathbf{T})\}$ and delete v and all its descendants. The tree edit distance between $\mathbf{T}$ and the resulting tree $\mathbf{T}'$ is the number of nodes in

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



(a) Tree edit distance. (b) Tree embedding distance. (c) Distributed vs. local changes.

Figure 14: Scalability results and subtree deletions.



(a) Deletion of leaf nodes. (b) Deletion of non-leaf nodes.

Figure 15: Properties of the *pq*-gram distance.

the deleted subtree. In Figure 14(c) we compare the results to distributed changes (average on 100 runs). We can see that local changes (subtree deletions) are cheaper than distributed changes.

7.3 Matchmaking with Real Data

To test the accuracy for real world data we use the address tables RO and LR described in Section 2. We build the address trees for all streets in both tables and get the sets R and L. Each tree $\mathbf{T}$ in one of the tree sets R and L represents a street with all the addresses in that street. Set R from RO consists of 302 trees with 52,509 nodes in total, reflecting 43,187 addresses. Set L from table LR consists of 300 trees with 53,464 nodes and 44,447 addresses.

We say that two trees $\mathbf{T} \in \mathsf{F}$ and $\mathbf{T}' \in \mathsf{F}'$ *match* if $\mathbf{T}$ has only one nearest neighbor in F', namely $\mathbf{T}'$, and vice versa. For each distance function dist_x we compute a mapping $M_x \in \mathsf{F} \times \mathsf{F}'$ between all pairs of matching trees. Furthermore, we create a mapping, M_c, by hand with the correct pairs of trees, i.e., with all pairs of trees that represent the same street in the real world. We define the *accuracy* of M_x with respect to M_c as $a = \frac{|M_x \cap M_c|}{|M_c|}$. The false positives are computed as $M_x \setminus M_c$.

We compute a mapping for the tree edit distance dist_{ed}, the *pq*-gram distance $\Delta^{p,q}$, the tree embedding distance dist_{emb}, and the node intersection dist_i. The node intersection 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 profile of a tree consists of the bag of all its node labels.

The results for the address tables RO and LR are shown in Table 1. There are two streets in RO that do not exist in LR, thus $|M_c| = 300$ for the calculation of the accuracy. The efficiency of the approximations is clearly greater than that of the tree edit distance: All of them can be computed within about five minutes, whereas the tree edit distance takes more than 52 hours.

	accuracy	correct	false pos.	runtime
dist_{ed}	82.7%	248	9	187,538s
$\Delta^{1,2}$	78.3%	235	5	181s
$\Delta^{2,3}$	77.3%	232	4	204s
$\Delta^{3,2}$	79.3%	238	2	180s
dist_{emb}	69.0%	207	8	313s
dist_i	66.3%	199	12	82s

Table 1: Accuracy of the tree edit distance and its approximations.

The *pq*-gram distance clearly outperforms the other approximations with respect to both, number of correct matches and number of false positives for all tested parameters. The number of false positives is even smaller than with the tree edit distance. The tree embedding distance does not perform much better than the simple node intersection. We will now briefly discuss how the tree embedding distance works, and why it performs poorly on typical address trees.

The tree embedding distance is computed by building a parsing hierarchy for a tree $\mathbf{T}$. In each phase i

a tree $\mathbf{T}^i$ is obtained by nodes of the tree $\mathbf{T}^{i-1}$. The parsing procedure starts with the tree $\mathbf{T}^0 = \mathbf{T}$, and it stops if $|\mathbf{T}^i| = 1$. Figure 16 shows the parse trees $\mathbf{T}^0$, $\mathbf{T}^1$ and $\mathbf{T}^2$ for an example tree $\mathbf{T}$ that is shaped like a typical address tree. In our illustration we use different types of brackets to label the newly created nodes for the different situations in which nodes are merged:

- Contiguous sequences of children are split into blocks of length 2 and 3, and the blocks are contracted. The nodes 1,3,5,10,11 of $\mathbf{T}^0$ become the two new nodes (1,3,5) and (10,11) of $\mathbf{T}^1$.

- A lone leaf child is merged with the parent node if it is the leftmost lone leaf. The nodes SN and 3 in $\mathbf{T}^0$ become the new node {SN,3} in $\mathbf{T}^1$.

- Chains (paths of degree-two nodes) are split into blocks of length 2 and 3, and the blocks are contracted. The nodes 2,A,(1,3,7) of $\mathbf{T}^1$ become the new node [2,A,(1,3,7)] of $\mathbf{T}^2$.

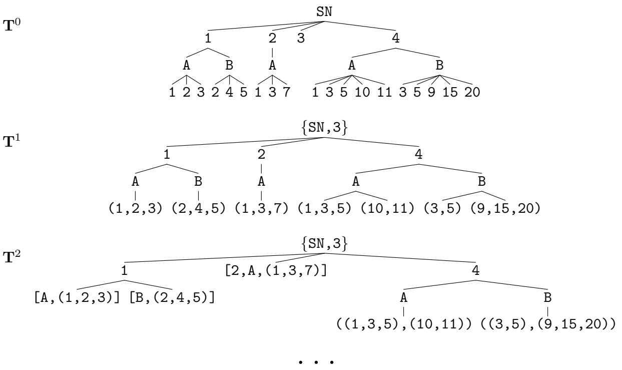


Figure 16: Parse trees for an example tree $\mathbf{T}$.

Each node in the parsing hierarchy corresponds to a set of nodes ("valid subtree") in the original tree. The bag P of all valid subtrees corresponding to all nodes of the final hierarchical parsing structure (tagged with a phase label to distinguish between subtrees in different phases) is treated the same way we treat the *pq*-gram profile in order to calculate the distance.

The resulting bag P contains nodes corresponding to (1) single nodes, (2) node chains with parent-child relationship, (3) contiguous leaf children, and (4) subtrees. Single nodes contain no structural information, parent-child chains only vertical, leaf sequences contain only horizontal structure information. Only subtrees reflect both, horizontal and vertical structure. Table 2 gives an overview of how many nodes of each type are obtained in each phase for the example tree. We can see that 65% of all nodes are single nodes containing no structural information. Only 19% of nodes correspond to subtrees.

Trees with many leaves at the deepest level are parsed bottom-up, and the structure of the inner nodes has less impact on the distance. For this reason the tree embedding distance performs only slightly better than a simple node intersection on our real world data.

phase	single node	chain	cont. leaf	subtree
0	29	-	-	-
1	8	1	7	-
2	4	1	2	3
3	2	-	-	4
4	1	-	-	3
5	-	-	-	2
6	-	-	-	1
total	44 65%	2 3%	9 13%	13 19%

Table 2: Types of valid subtrees in the different phases.

8 Conclusions

Our work is motivated by a data integration scenario from the Municipality of Bozen, where data from different sources have to be integrated and no common keys exist. Data have to be joined over residential addresses, which in practice have some undesirable properties, and exact joins completely fail. To overcome these problems we introduced address trees as a representation of residential addresses. This reduces the integration to an approximate join on address trees.

We presented a new distance measure, the *pq*-gram distance, for ordered labeled trees as an effective and efficient approximation for the well known tree edit distance. We provided an algorithm for the computation of *pq*-grams in $O(n)$ time, where n is the number of tree nodes. Based on the profile 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.

The *pq*-gram distance behaves differently from the tree edit distance for structural and local changes. It gives more weight to edit operations that cause big changes in the tree structure. This property turned out to be relevant in our application domain.

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 from the municipality domain turned out to be clearly better than other approximations of the tree edit distance.

In the future we will investigate additional application areas and apply the *pq*-gram distance for data cleaning and the comparison of XML data.

9 Acknowledgements

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

References

[1] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. Structural joins: A primitive for efficient XML query pattern matching. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, pages 141–152, San Jose, California, 2002. ACM Press.

[2] N. Augsten, M. Böhlen, and J. Gamper. Reducing the integration of public administration databases to approximate tree matching. In R. Traunmüller, editor, *Electronic Government – Third International Conference, EGOV 2004*, LNCS 3183, pages 102–107, Zaragoza, Spain, 2004.

[3] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: Optimal XML pattern matching. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 310–321, Madison, Wisconsin, June 2002. ACM Press.

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

[5] J. Celko. *Trees and Hierarchies in SQL for Smarties.* Morgan Kaufmann Publishers Inc., 2004.

[6] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 493–504, Montreal, Canada, June 1996. ACM Press.

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

[8] D. DeHaan, D. Toman, M. P. Consens, and M. T. Özsu. A comprehensive XQuery to SQL translation using dynamic interval encoding. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 623–634, San Diego, California, June 2003. ACM Press.

[9] M. Garofalakis and A. Kumar. XML stream processing using tree-edit distance embeddings. *ACM Trans. on Database Systems*, 30(1):279–332, 2005.

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

[11] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML joins. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 287–298, Madison, Wisconsin, 2002. ACM Press.

[12] H. Jiang, W. Wang, H. Lu, and J. X. Yu. Holistic twig joins on indexed XML documents. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 273–284, Berlin,Germany, Sept. 2003. Morgan Kaufmann Publishers Inc.

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

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

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

[16] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[17] N. Polyzotis, M. Garofalakis, and Y. Ioannidis. Approximate XML query answers. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 263–274, Paris, France, June 2004. ACM Press.

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

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

[20] E. Tanaka and K. Tanaka. The tree-to-tree editing problem. *Int. Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 2(2):221–240, 1988.

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

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

[23] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. On supporting containment queries in relational database management systems. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 425–436, Santa Barabara, California, 2001.

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