Proc. 9th Int. Conf. on Extending Database Technology, pp. 676-693, Heraklion, Greece, 2004

Efficient Similarity Search for Hierarchical Data in Large Databases

Karin Kailing[1] Hans-Peter Kriegel[1] Stefan Schönauer[1] and Thomas Seidl[2]

[1] University of Munich
Institute for Computer Science
{kailing,kriegel,schoenauer}@informatik.uni-muenchen.de
[2] RWTH Aachen University
Department of Computer Science IX
seidl@informatik.rwth-aachen.de

Abstract. Structured and semi-structured object representations are getting more and more important for modern database applications. Examples for such data are hierarchical structures including chemical compounds, XML data or image data. As a key feature, database systems have to support the search for similar objects where it is important to take into account both the structure and the content features of the objects. A successful approach is to use the edit distance for tree structured data. As the computation of this measure is NP-complete, constrained edit distances have been successfully applied to trees. While yielding good results, they are still computationally complex and, therefore, of limited benefit for searching in large databases. In this paper, we propose a filter and refinement architecture to overcome this problem. We present a set of new filter methods for structural and for content-based information in tree-structured data as well as ways to flexibly combine different filter criteria. The efficiency of our methods, resulting from the good selectivity of the filters is demonstrated in extensive experiments with real-world applications.

1 Introduction

Recently, databases are used more and more to store complex objects from scientific, engineering or multimedia applications. In addition to a variety of content-based attributes, complex objects typically carry some kind of internal structure which often forms a hierarchy. Examples of such data include chemical compounds, CAD drawings, XML documents, web sites or color images. The efficient search for similar objects in such databases, for example to classify new objects or to cluster database objects, is a key feature in those application domains. Beside the internal structure of the objects, the content information stored in the tree structure determines the similarity of different objects, too. Whereas the concept of feature vectors has proven to be very successful for unstructured content data, we particularly address the internal structure of similar objects. For this purpose we discuss several similarity measures for trees as proposed in the literature [1–3]. These measures are well suited for hierarchical objects and have been applied to web site analysis [4], structural similarity of XML documents [5], shape recognition [6] and chemical substructure search [4], for instance. A general problem of all those measures is their computational complexity, which makes them unsuitable for large databases. The core idea of our approach is to apply a filter criterion to the database objects in order to obtain a small set of candidate answers to a query. The final result is then retrieved from this candidate set through the use of the original complex similarity measure. This filter-refinement architecture reduces the number of expensive similarity distance calculations and speeds up the search process. To extend this concept to the new problem of searching similar tree structures, efficient and effective filters for structural properties are required. In this paper, we propose several new filter methods for tree structures and also demonstrate how to combine them with filters for content information in order to obtain a high filter selectivity.

In the next section, we discuss several measures for structural similarity. In section 3, the concept of a filter-refinement architecture is presented, while section 4 deals with our filter methods. Finally, we present an experimental evaluation of our filters, before we conclude the paper.

2 Structural similarity

Quantifying the similarity of two trees requires a structural similarity measure. There exist several similarity measures for general graphs in the literature [7–9]. All of them either suffer from a high computational complexity or are limited to special graph types. Papadopoulos and Manolopoulos presented a measure based on certain edit operations for general graphs [10]. They use the degree sequence of a graph as feature vector and the Manhattan distance between the feature vectors as similarity measure. While their measure can be calculated efficiently, it is not applicable to attributed graphs. Consequently, special distance measures for labeled trees which exploit the structure and content of trees become necessary. Jiang, Wang and Zhang [1] suggested a measure based on a structural alignment of trees. They also prove that the structural alignment problem for trees is NP-hard if the degree of the trees is not bounded. Selkow [2] presented a tree-to-tree editing algorithm for ordered labeled trees. It is a first step towards the most common approach to measure tree similarity, which is the edit distance. The edit distance, well known from string matching [11, 12], is the minimal number of edit operations necessary to transform one tree into the other. There are many variants of the edit distance known, depending on which edit operations are allowed. The basic form allows two edit operations, i.e. the insertion and the deletion of a tree node. The insertion of a node n in a tree below a node p means that p becomes the parent of n and a subset of p's children become n's children. The deletion of a node is the inverse operation to the insertion of the node. In the case of attributed nodes, as they appear in most real world applications, the change of a node label is introduced as a third basic operation. Using those operations, we can define the edit distance between two trees as follows.

Definition 1 (edit sequence, cost of an edit sequence). *An edit operation e is the insertion, deletion or relabeling of a node in a tree t. Each edit operation e is assigned a non-negative cost $c(e)$. The cost of a sequence of edit operations $S = \langle e_1, \ldots, e_m \rangle$, $c(S)$, is defined as the sum of the cost of each edit operation in S, i.e. $c(S) = c(e_1) + \ldots + c(e_m)$.*

Definition 2 (edit distance). *The edit distance between two trees t_1 and t_2, $ED(t_1, t_2)$, is the minimum cost of all edit sequences that transform t_1 into t_2:*
$ED(t_1, t_2) = min\{c(S)|S\ a\ sequence\ of\ edit\ operations\ transforming\ t_1\ into\ t_2\}$

A great advantage of using the edit distance as a similarity measure is that along with the distance value, a mapping between the nodes in the two trees is provided in terms of the edit sequence. The mapping can be visualized and can serve as an explanation of the similarity distance to the user. This is especially important in the context of similarity search, as different users often have a different notion of similarity in mind. Here, an explanation component can help the user to adapt weights for the distance measure in order to reflect the individual notion of similarity. Zhang, Statman and Shasha, however, showed that computing the edit distance between unordered labeled trees is NP-complete [13]. Obviously, such a complex similarity measure is unsuitable for large databases. To overcome this problem, Zhang proposed a constrained edit distance between trees, the degree-2 edit distance. The main idea behind this distance measure is that only insertions or deletions of nodes with a maximum number of two neighbors are allowed.

Definition 3 (degree-2 edit distance). *The edit distance between two trees t_1 and t_2, $ED_2(t_1, t_2)$, is the minimum cost of all degree-2 edit sequences that transform t_1 into t_2 or vice versa. A degree-2 edit sequence consists only of insertions or deletions of nodes n with $degree(n) \leq 2$, or of relabelings:*
$ED_2(t_1, t_2) = min\{c(S)|S\ is\ a\ degree\text{-}2\ edit\ sequence\ transforming\ t_1\ into\ t_2\}$

One should note that the degree-2 edit distance is well defined in the sense that two trees can always be transformed into each other by using only degree-2 edit operations. In [14] an algorithm is presented to compute the degree-2 edit distance in $O(|t_1||t_2|D)$ time, where D is the maximum of the degrees of t_1 and t_2 and $|t_i|$ denotes the number of nodes in t_i. Whereas this measure has a polynomial time complexity, it is still too complex for the use in large databases. To overcome this problem, we extend the paradigm of filter-refinement architectures to the context of structural similarity search.

3 Multistep query processing

The main goal of a filter-refinement architecture, as depicted in figure 1, is to reduce the number of complex and time consuming distance calculations in the query process. To achieve this goal, query processing is performed in two or more steps. The first step is a filter step which returns a number of candidate objects from the database. For those candidate objects the exact similarity distance is then determined in the refinement step and the objects fulfilling the query predicate are reported. To reduce the overall search time, the filter step has to fulfill certain constraints. First, it is essential, that the filter predicate is considerably easier to determine than the exact similarity measure. Second, a substantial part of the database objects must be filtered out. Obviously, it depends on the complexity of the similarity measure what filter selectivity is sufficient. Only if both conditions are satisfied, the performance gain through filtering is greater than the cost for the extra processing step.

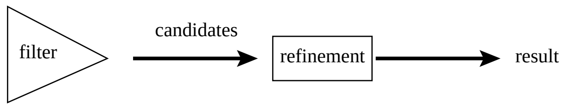


Fig. 1. The filter-refinement architecture.

Additionally, the completeness of the filter step is an important property. Completeness in this context means that all database objects satisfying the query condition are included in the candidate set. Available similarity search algorithms guarantee completeness if the distance function in the filter step fulfills the following lower-bounding property. For any two objects p and q, a lower-bounding distance function d_{lb} in the filter step has to return a value that is not greater than the exact distance d_e of p and q, i.e. $d_{lb}(p, q) \leq d_e(p, q)$. With a lower-bounding distance function it is possible to safely filter out all database objects which have a filter distance greater than the current query range because the similarity distance of those objects cannot be less than the query range.

Using a multi-step query architecture requires efficient algorithms which actually make use of the filter step. Agrawal, Faloutsos and Swami proposed such an algorithm for range search [15]. In [16] a multi-step algorithm for k-nearest-neighbor search is presented, which is optimal in the sense that the minimal number of exact distance calculations are performed during query processing.

4 Structural and content-based filters for unordered trees

In this section, we introduce several filtering techniques that support efficient similarity search for tree-structured data. Whereas single-valued features including the height of a tree, the number of nodes, or the degree of a tree, are of limited use, as we learned from preliminary experiments, we propose the use of feature histograms The advantage of this extension is that there is more information provided to the filter step for the purpose of generating candidates and, thus, the discriminative power is increased. Additionally, a variety of multidimensional index structures and efficient search algorithms are available for vector data including histograms. The particular feature histograms which we propose in the following are based on the height, the degree or the label of individual nodes.

4.1 Filtering based on the height of nodes

A promising way to filter unordered trees based on their structure is to take the height of nodes into account. A very simple technique is to use the height of a tree as a single feature. The difference of the height of two trees is an obvious lower bound for the edit distance between those trees, but this filter clearly is very coarse, as two trees with completely different structure but the same height cannot be distinguished.

A more fine-grained and more sensitive filter can be obtained by creating a histogram of node heights in a tree and using the difference between those histograms as

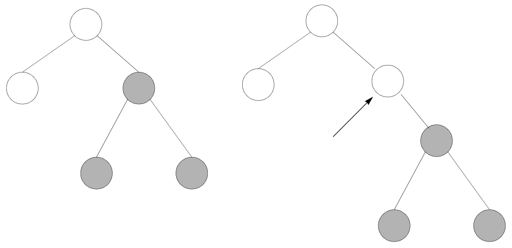


Fig. 2. A single insertion can change the distance to the root for several nodes.

a filter distance. A first approach is to determine the distance of each node in the tree to the root node and then to store the distribution of those values in a histogram. Unfortunately, the distance between two such histograms is not guaranteed to be a lower bound for the edit distance or the degree-2 edit distance between the original trees. As can be seen in figure 2, the insertion of a single node may change the height of all nodes in its subtree. Thus, the number of affected histogram bins is only bounded by the height of the tree.

Therefore, we propose a different approach to consider the height of a node. Instead of the distance of a node from the root, its leaf distance is used to approximate the structure of a tree.

Definition 4 (leaf distance). *The leaf distance $d_l(n)$ of a node n is the maximum length of a path from n to any leaf node in the subtree rooted at n.*

Based on this definition, we introduce the leaf distance histogram of a tree as illustrated in figure 3.

Definition 5 (leaf distance histogram). *The leaf distance histogram $h_l(t)$ of a tree t is a vector of length $k = 1 + height(t)$ where the value of any bin $i \in 0, \ldots, k$ is the number of nodes that share the leaf distance i, i.e. $h_l(t)[i] = |n \in t, d_l(n) = i|$.*

For the proof of the following theorem the definition of a maximum leaf path is useful:

Definition 6 (maximum leaf path). *A maximum leaf path (MLP) of a node n in a tree t is a path of maximum length from n to a leaf node in the subtree rooted by n.*

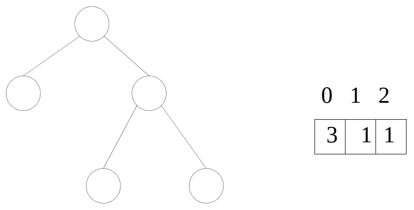


Fig. 3. Leaf distance of nodes and leaf distance histogram.

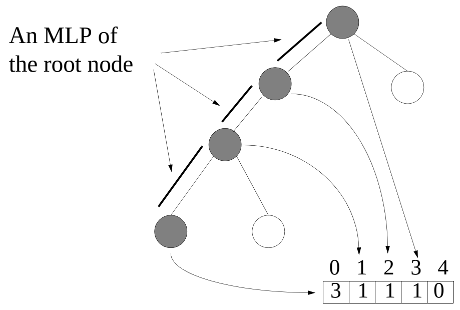


Fig. 4. A maximum leaf path.

An important observation is that adjacent nodes on an MLP are mapped to adjacent bins in the leaf distance histogram as illustrated in figure 4.

Theorem 1. *For any two trees t_1 and t_2, the L_1-distance of the leaf distance histograms is a lower bound of the edit distance of t_1 and t_2:*

$$L_1(h_l(t_1), h_l(t_2)) \leq ED(t_1, t_2)$$

Proof. Given two arbitrary trees t_0 and t_m, let us consider an edit sequence $S = \langle S_1, \ldots, S_m \rangle$ that transforms t_0 to t_m. We proceed by induction over the length $m = |S|$. If $m = 0$, i.e. $S = \langle \rangle$ and $t_0 = t_m$, the values of $L_1(h_l(t_0), h_l(t_m))$ and of $c(S)$ both are equal to zero. For $m > 0$, let us assume that the lower-bounding property already holds for the trees t_0 and t_{m-1}, i.e. $L_1(h_l(t_0), h_l(t_{m-1})) \leq c(\langle S1, \ldots, S_{m-1} \rangle)$. When extending the sequence $\langle S_1, \ldots, S_{m-1} \rangle$ by S_m to S, the right hand side of the inequality is increased by $c(S_m) = 1$.

The situation on the left hand side is as follows. The edit step S_m may be a relabeling, an insertion or a deletion. Obviously, the effect on the leaf distance histogram $h_l(t_{m-1})$ is void in case of a relabeling, i.e. $h_l(t_m) = h_l(t_{m-1})$, and the inequality $L_1(h_l(t_0), h_l(t_m)) = L_1(h_l(t_0), h_l(t_{m-1})) \leq c(S)$ holds.

The key observation for an insert or a delete operation is that only a single bin is affected in the histogram in any case. When a node ν is inserted, for all nodes below the insertion point, clearly, the leaf distance does not change. Only the leaf distance of any predecessor of the inserted node may or may not be increased by the insertion. Therefore, if ν does not belong to an MLP of any of its predecessors, only the bin affected by the inserted node is increased by one. This means that in the leaf distance histogram exactly one bin is increased by one. On the other hand, if an MLP of any of the predecessors of ν containing ν exists, then we only have to consider the longest of those MLPs. Due to the insertion, this MLP grows in size by one. As all nodes along the MLP are mapped into consecutive histogram bins, exactly one more bin than before is influenced by the nodes on the MLP. This means that exactly one bin in the leaf distance histogram changes due to the insertion. As insertion and deletion are symmetric operations, the same considerations hold for the deletion of a node.

The preceding considerations hold for all edit sequences transforming a tree t_1 into a tree t_2 and particularly include the minimum cost edit sequence. Therefore, the lower-

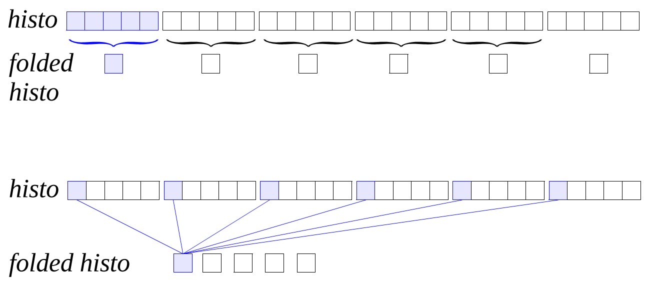


Fig. 5. Folding techniques for histograms: The technique of Papadopoulos and Manolopoulos (top) and the modulo folding technique (bottom).

bounding relationship immediately holds for the edit distance $ED(t_1, t_2)$ of two trees t_1 and t_2, too.

It should be noticed that the above considerations do not only hold for the edit distance but also for the degree-2 edit distance. Therefore, the following theorem allows us also to use leaf-distance histograms for the degree-2 edit distance.

Theorem 2. *For any two trees t_1 and t_2, the L_1-distance of the leaf distance histograms is a lower bound of the degree-2 edit distance of t_1 and t_2:*

$$L_1(h_l(t_1), h_l(t_2)) \leq ED_2(t_1, t_2)$$

Proof. Analogously to the proof of theorem 1.

Theorem 1 and 2 also allow us to use leaf distance histograms as a filter for the weighted edit and weighted degree-2 edit distance. This statement is justified by the following considerations. As shown above, the L_1-distance of two leaf distance histograms gives a lower bound for the insert and delete operations that are necessary to transform the two corresponding trees into each other. This fact also holds for weighted relabeling operations, as weights do not have any influence on the necessary structural modifications. But even when insert/delete operations are weighted, our filter can be used as long as their exists a smallest possible weight w_{min} for an insert or delete operation. In this case, the term $(L_1(h_l(t_1), h_l(t_2)) \cdot w_{min})$ is a lower bound for the weighted edit and degree-2 edit distance between the trees t_1 and t_2. Since we assume metric properties as well as the symmetry of insertions and deletions for the distance, the triangle inequality guarantees the existence of such a minimum weight. Otherwise, any relabeling of a node would be performed cheaper by a deletion and a corresponding insertion operation. Moreover, structural differences of objects would be reflected only weakly if structural changes are not weighted properly.

Histogram folding. Another property of leaf distance histograms is that their size is unbounded as long as the height of the trees in the database is also unbounded. This problem arises for several feature vector types, including the degree histograms presented in section 4.2. Papadopoulos and Manolopoulos [10] address this problem by folding the histograms into vectors with fixed dimension. This is done in a piecewise grouping process. For example, when a 5-dimensional feature vector is desired, the first one fifth of the histogram bins is summed up and the result is used as the first component of the feature vector. This is done analogously for the rest of the histogram bins. The above approach could also be used for leaf distance histograms, but it has the disadvantage that the maximal height of all trees in the database has to be known in advance. For dynamic data sets, this precondition cannot be fulfilled. Therefore, we propose a different technique that yields fixed-size n-dimensional histograms by adding up the values of certain entries in the leaf distance histogram. Instead of summing up adjacent bins in the histogram, we add up those with the same index modulo n, as depicted in figure 5. This way, histograms of distinct length can be compared, and there is no bound for the length of the original histograms.

Definition 7 (folded histogram). *A folded histogram $h_{fn}(h)$ of a histogram h for a given parameter n is a vector of size n where the value of any bin $i \in 0, \ldots, n-1$ is the sum of all bins k in h with $k \ mod \ n = i$, i.e.*

$$h_{fn}(h)[i] = \sum_{k=0\ldots(|h|-1) \wedge k \ mod \ n=i} h[k]$$

The following theorem justifies to use folded histograms in a multi-step query processing architecture.

Theorem 3. *For any two histograms h_1 and h_2 and any parameter $n \geq 1$, the L_1-distance of the folded histograms of h_1 and h_2 is a lower bound for the L_1-distance of h_1 and h_2:*

$$L_1(h_{fn}(h_1), h_{fn}(h_2)) \leq L_1(h_1, h_2)$$

Proof. Let $len = n \cdot \lceil \frac{max(h_1,h_2)}{n} \rceil$ be the length of h_1 and h_2. If necessary, h_1 and h_2 are extended with bins containing 0 until $|h_1| = len$ and $|h_2| = len$. Then the following holds:

$$\begin{aligned}
&L_1(h_{fn}(h_1), h_{fn}(h_2)) \\
&= \sum_{i=0}^{n-1} \left| \sum_{\substack{k=0\ldots((|h_1|-1) \\ \wedge k \ MOD \ n=i}} h_1[k] - \sum_{\substack{k=0\ldots((|h_2|-1) \\ \wedge k \ MOD \ n=i}} h_2[k] \right| \\
&= \sum_{i=0}^{n-1} \left| \sum_{j=0}^{(len \ DIV \ n)-1} h_1[i+j\cdot n] - \sum_{j=0}^{(len \ DIV \ n)-1} h_2[i+j\cdot n] \right| \\
&\leq \sum_{i=0}^{n-1} \sum_{j=0}^{(len \ DIV \ n)-1} |h_1[i+j\cdot n] - h_2[i+j\cdot n]| \\
&= \sum_{j=0}^{len} |h_1[k] - h_2[k]| \\
&= L_1(h_1, h_2)
\end{aligned}$$

4.2 Filtering based on degree of nodes

The degrees of the nodes are another structural property of trees which can be used as a filter for the edit distances. Again, a simple filter can be obtained by using the maximal degree of all nodes in a tree t, denoted by $degree_{max}(t)$, as a single feature. The difference between the maximal degrees of two trees is an obvious lower bound for the edit distance as well as for the degree-2 edit distance. As before, this single-valued filter is very coarse and using a degree histogram clearly increases the selectivity.

Definition 8 (degree histogram). *The degree histogram $h_d(t)$ of a tree t is a vector of length $k = 1 + degree_{max}(t)$ where the value of any bin $i \in 0, \ldots, k$ is the number of nodes that share the degree i, i.e. $h_d(t)[i] = |n \in t, degree(n) = i|$.*

Theorem 4. *For any two trees t_1 and t_2, the L_1-distance of the degree histograms divided by three is a lower bound of the edit distance of t_1 and t_2:*

$$\frac{L_1(h_d(t_1), h_d(t_2))}{3} \leq ED(t_1, t_2)$$

Proof. Given two arbitrary trees t_0 and t_m, let us consider an edit sequence $S = \langle S1, \ldots, S_m \rangle$ that transforms t_0 into t_m. We proceed by induction over the length of the sequence $m = |S|$. If $m = 0$, i.e. $S = \langle\rangle$ and $t_0 = t_m$, the values of $\frac{L_1(h_d(t_0), h_d(t_m))}{3}$ and of $c(S)$ both are equal to zero. For $m > 0$, let us assume that the lower-bounding property already holds for t_0 and t_{m-1}, i.e. $\frac{L_1(h_d(t_0), h_d(t_{m-1}))}{3} \leq c(\langle S1, \ldots, S_{m-1} \rangle)$. When extending the sequence $\langle S_1, \ldots, S_{m-1} \rangle$ by S_m to S, the right hand side of the inequality is increased by $c(S_m) = 1$. The situation on the left hand side is as follows. The edit step S_m may be a relabeling, an insert or a delete operation. Obviously, for a relabeling, the degree histogram $h_d(t_{m-1})$ does not change, i.e. $h_d(t_m) = h_d(t_{m-1})$ and the inequality $\frac{L_1(h_d(t_0), h_d(t_m))}{3} = \frac{L_1(h_d(t_0), h_d(t_{m-1}))}{3} \leq c(S)$ holds.

The insertion of a single node affects the histogram and the L_1-distance of the histograms in the following way:

1. The inserted node n causes an increase in the bin of n's degree. That may change the L_1-distance by at most one.
2. The degree of n's parent node p may change. In the worst case this affects two bins. The bin of p's former degree is decreased by one while the bin of its new degree is increased by one. Therefore, the L_1-distance may additionally be changed by at most two.
3. No other nodes are affected.

From the above three points it follows that the L_1-distance of the two histograms $h_d(t_{m-1})$ and $h_d(t_m)$ changes by at most three. Therefore, the following holds:

$$\frac{L_1(h_d(t_0), h_d(t_m))}{3} \leq \frac{L_1(h_d(t_0), h_d(t_{m-1})) + 3}{3}$$

$$\frac{L_1(h_d(t_0), h_d(t_m))}{3} \leq \frac{L_1(h_d(t_0), h_d(t_{m-1})}{3} + 1$$

$$\frac{L_1(h_d(t_0), h_d(t_m))}{3} \leq c(\langle S1, \ldots, S_{m-1} \rangle) + 1$$

$$\frac{L_1(h_d(t_0), h_d(t_m))}{3} \le c(\langle S1, \ldots, S_{m-1}, S_m \rangle)$$
$$\frac{L_1(h_d(t_1), h_d(t_2))}{3} \le ED(t_1, t_2)$$

As the above considerations also hold for the degree-2 edit distance, theorem 4 holds analogously for this similarity measure.

4.3 Filtering based on node labels

Apart from the structure of the trees, the content features, expressed through node labels, have an impact on the similarity of attributed trees. The node labels can be used to define a filter function. To be useful in our filter-refinement architecture, this filter method has to deliver a lower bound for the edit cost when transforming two trees into each other. The difference between the distribution of the values within a tree and the distribution of the values in another tree can be used to develop a lower-bounding filter. To ensure efficient evaluation of the filter, the distribution of those values has to be approximated for the filter step.

One way to approximate the distribution of values is to use histograms. In this case, an n-dimensional histogram is derived by dividing the range of the node label into n bins. Then, each bin is assigned the number of nodes in the tree whose value is in the range of the bin. To estimate the edit distance or the degree-2 edit distance between two trees, half of the L_1-distance of their corresponding label histograms is appropriate. A single insert or delete operation changes exactly one bin of such a label histogram, a single relabeling operation can influence at most two histogram bins. If a node is assigned to a new bin after relabeling, the entry in the old bin is decreased by one and the entry in the new bin is increased by one (cf. figure 6). Otherwise, a relabeling does not change the histogram. This method also works for weighted variants of the edit distance and the degree-2 edit distance as long as there is a minimal weight for a relabeling operation. In this case, the calculated filter value has to be multiplied by this minimal weight in order to gain a lower-bounding filter.

This histogram approach applies to discrete label distributions very well. However, for continuous label spaces, the use of a continuous weight function which may become arbitrarily small, can be reasonable. In this case, a discrete histogram approach can not be used. An example for such a weight function is the Euclidean distance in the color

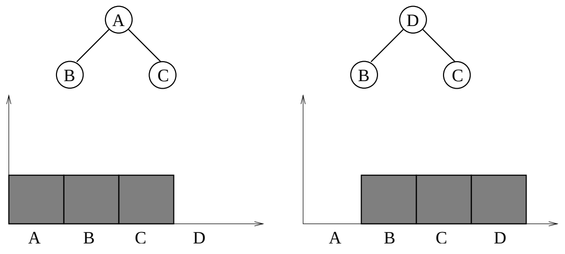


Fig. 6. A single relabeling operation may result in a label histogram distance of two.

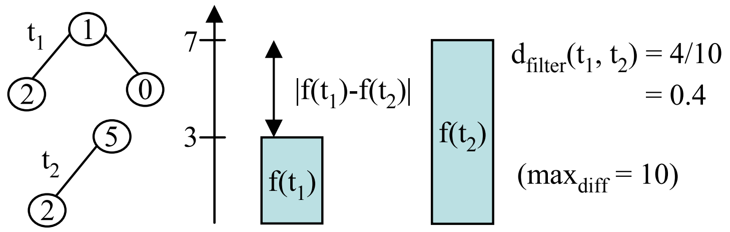


Fig. 7. Filtering for continuous weight functions.

space, assuming trees where the node labels are colors. Here, the cost for changing a color value is proportional to the Euclidean distance between the original and the target color. As this distance can be infinitely small, it is impossible to estimate the relabeling cost based on a label histogram as in the above cases.

More formally, when using the term ’continuous weight function’ we mean that the cost for changing a node label from value x_1 to value x_2 is proportional to $|x_1 - x_2|$. Let max_{diff} be the maximal possible difference between two attribute values. Then $|x_1 - x_2|$ has to be normalized to $[0, 1]$ by dividing it through max_{diff}, assuming that the maximal cost for a single insertion, deletion or relabeling is one. To develop a filter method for attributes with such a weight function, we exploit the following property of the edit distance measure. The cost-minimal edit sequence between two trees removes the difference between the distributions of attribute values of those two trees. It does not matter whether this is achieved through relabelings, insertions or deletions.

For our filter function we define the following feature value $f(t)$ for a tree t:

$$f(t) = \sum_{i=1}^{|t|} |x_i|$$

Here x_i is the attribute value of the i-th node in t and $|t|$ is the size of tree t. The absolute difference between two such feature values is an obvious lower bound for the difference between the distribution of attribute values of the corresponding trees. Consequently, we use

$$d_{filter}(t_1, t_2) = \frac{|f(t_1) - f(t_2)|}{max_{diff}}$$

as a filter function for continuous label spaces, see figure 7 for an illustration. Once more, the above considerations also hold for the degree-2 edit distance.

To simplify the presentation we assumed that a node label consists of just one single attribute. But usually a node will carry several different attributes. If possible, the attribute with the highest selectivity can be chosen for filtering. In practice, there is often no such single attribute. In this case, filters for different attributes can be combined with the technique described in the following section.

4.4 Combining filter methods

All of the above filters use a single feature of an attributed tree to approximate the edit distance or degree-2 edit distance. As the filters are not equally selective in each situation, we propose a method to combine several of the presented filters.

A very flexible way of combining different filters is to follow the inverted list approach, i.e. to apply the different filters independently from each other and then intersect the resulting candidate sets. With this approach, separate index structures for the different filters have to be maintained and for each query, a time-consuming intersection step is necessary. To avoid those disadvantages, we concatenate the different filter histograms and filter values for each object and use a combined distance function as a similarity function.

Definition 9 (Combined distance function). *Let $C = d_i$ be a set of distance functions for trees. Then, the combined distance function d_c is defined to be the maximum of the component functions:*

$$d_C(t_1, t_2) = max\{d_i(t_1, t_2)\}$$

Theorem 5. *For every set of lower-bounding distance functions $C = \{d_{low}(t_1, t_2)\}$, i.e. for all trees t_1 and t_2 $d_i(t_1, t_2) \leq ED(t_1, t_2)$, the combined distance function d_C is a lower bound of the edit distance function d_{ED}:*

$$d_C(t_1, t_2) \leq ED(t_1, t_2)$$

Proof. For all trees t_1 and t_2, the following equivalences hold:

$$\begin{aligned} d_C(t_1, t_2) \leq ED(t_1, t_2) &\Leftrightarrow \\ max\{d_i(t_1, t_2)\} \leq ED(t_1, t_2) &\Leftrightarrow \\ \forall d_i : d_i(t_1, t_2) \leq ED(t_1, t_2) & \end{aligned}$$

The final inequality represents the precondition.

Justified by theorem 5, we apply each separate filter function to its corresponding component of the combined histogram. The combined distance function is derived from the results of this step.

5 Experimental evaluation

For our tests, we implemented a filter and refinement architecture according to the optimal multi-step k-nearest-neighbor search approach as proposed in [16]. Naturally, the positive effects which we show in the following experiments for k-nn-queries also hold for range queries and for all data mining algorithms based on range queries or k-nn-queries (e.g. clustering, k-nn-classification). As similarity measure for trees, we implemented the degree-2 edit distance algorithm as presented in [14]. The filter histograms were organized in an X-tree [17]. All algorithms were implemented in Java 1.4 and the experiments were run on a workstation with a Xeon 1,7 GHz processor and 2 GB main memory under Linux.

To show the efficiency of our approach, we chose two different applications, an image database and a database of websites which are described in the following.

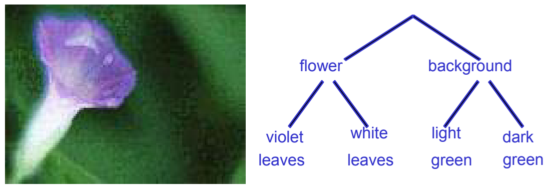


Fig. 8. Structural and content-based information of a picture represented as a tree.

5.1 Image databases

As one example of tree structured objects we chose images, because for images, both, content-based as well as structural information are important. Figure 8 gives an idea of the two aspects which are present in a picture.

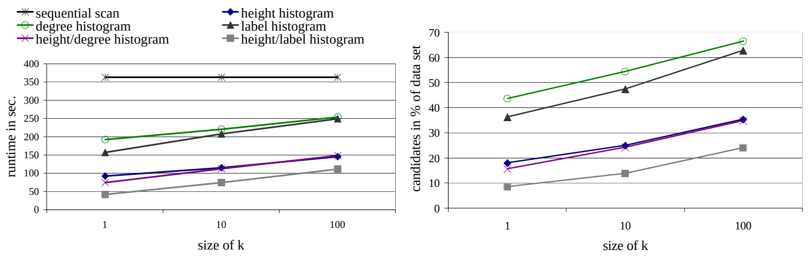
The images we used for our experiments were taken from three real-world databases: a set of 705 black and white pictographs, a set of 8,536 commercially available color images and a set of 43,000 color TV-Images. We extracted trees from those images in a two-step process. First, the images were divided into segments of similar color by a segmentation algorithm. In the second step, a tree was created from those segments by iteratively applying a region-growing algorithm which merges neighboring segments if their colors are similar. This is done until all segments are merged into a single node. As a result, we obtain a set of labeled unordered trees where each node label describes the color, size and horizontal as well as vertical extension of the associated segment. Table 1 shows some statistical information about the trees we generated.

Table 1. Statistics of the data set.

	number	number of nodes			height			maximal degree		
	of images	max	min	Ø	max	min	Ø	max	min	Ø
commercial color images	8,536	331	1	30	24	0	3	206	0	18
color TV-images	43,000	109	1	24	13	0	3	71	0	11
black and white pictographs	705	113	3	13	2	1	1	112	2	12

For the first experiments, we used label histograms as described in section 4.3. To derive a discrete label distribution, we reduced the number of different attribute values to 16 different color values for each color channel and 4 different values each for size and extensions. We used a relabeling function with a minimal weight of 0.5. Later on we also show some experiments where we did not reduce the different attribute values and used a continuous weight function for relabeling.

Comparison of our filter types. For our first experiment we used 10,000 TV-images. We created 10-dimensional height and degree histograms and combined them as described in section 4.4. We also built a 24-dimensional combined label histogram which



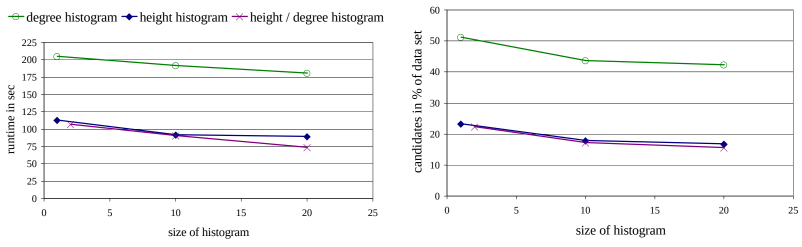
(a) (b)

Fig. 9. Runtime and number of candidates for k-nn-queries on 10,000 color TV-images.

considered the color, size and extensions of all node labels (6 attributes with histograms of size 4). Finally, the combination of this combined label histogram and a 4-dimensional height histogram was taken as another filter criterion. Let us note, that the creation of the filter X-trees took between 25 sec. for the height histogram and 62 sec. for the combined height-label histogram.

We ran 70 k-nearest-neighbor queries (k = 1, 10, 100) for each of our filters. Figure 9 shows the selectivity of our filters, measured in the average number of candidates with respect to the size of the data set. The figures show that filtering based solely on structural (height or degree histogram) or content-based features (label histogram) is not as effective as their combination. Figure 9 also shows that for this data the degree filter is less selective than the height filter. The method which combines the filtering based on the height of the nodes and on content features is most effective. Figure 5.1 additionally depicts the average runtime of our filters compared to the sequential scan. As one can see, we reduced the runtime by a factor of up to 5. Furthermore, the comparison of the two diagrams in figure 9 shows that the runtime is dominated by the number of candidates, whereas the additional overhead due to the filtering is negligible.

Influence of histogram size. In a next step we tested to what extent the size of the histogram influences the size of the candidate set and the corresponding runtime. The results for nearest neighbor queries on 10,000 color TV-images are shown in figure 10. With increasing dimension, the number of candidates as well as the runtime decrease.



(a) (b)

Fig. 10. Influence of dimensionality of histograms.

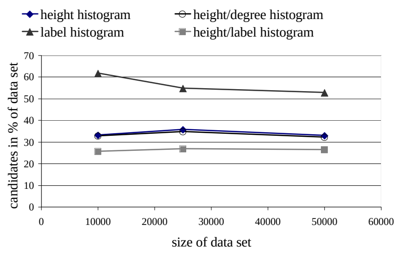


Fig. 11. Scalability versus size of data set.

The comparison of the two diagrams in figure 10 shows that the runtime is again dominated by the number of candidates, while the additional overhead due to higher dimensional histograms is negligible.

Scalability of filters versus size of data set. For this experiment we united all three image data sets and chose three subsets of size 10,000, 25,000 and 50,000. On these subsets we performed several representative 5-nn queries. Figure 11 shows that the selectivity of our structural filters does not depend on the size of the data set.

Comparison of different filters for a continuous weight function. As mentioned above, we also tested our filters when using a continuous weight function for relabeling. For this experiment, we used the same 10,000 color images as in 5.1. Figure 12 shows the results averaged over 200 k-nn queries. In this case, both the height histogram and the label filter are very selective. Unfortunately, the combination of both does not further enhance the runtime. While there is a slight decrease in the number of candidates, this is used up by the additional overhead of evaluating two different filter criteria.

Comparison with a metric tree. In [18] other efficient access methods for similarity search in metric spaces are presented. In order to support dynamic datasets, we use the X-tree that can be updated at any time. Therefore, we chose to compare our filter

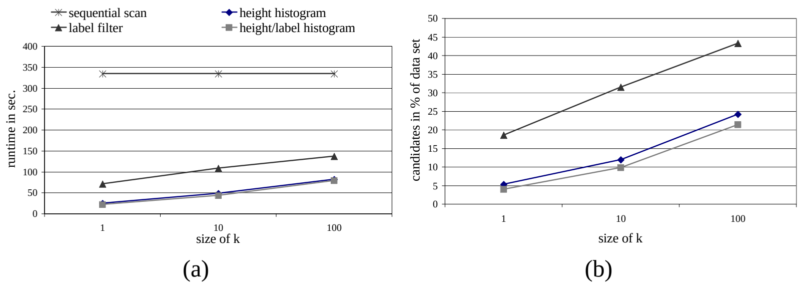


Fig. 12. Runtime and number of candidates when using a continuous weight function.

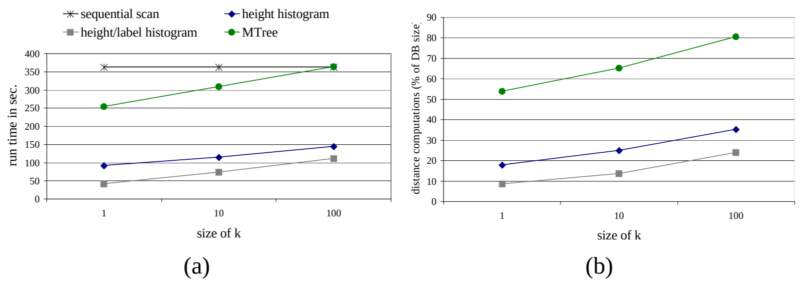


Fig. 13. Runtime and number of distance computations of filter methods compared to the M-Tree.

methods to the M-tree which analogously is a dynamic index structure for metric spaces. We implemented the M-tree as described in [19] by using the best split policy mentioned there.

The creation of an M-tree for 1,000 tree objects already took more than one day, due to the split policy that has quadratic time-complexity. The time for the creation of the filter vectors, on the other hand, was in the range of a few seconds. As can be seen in figure 13, the M-tree outperformed the sequential scan for small result sizes. However, all of our filtering techniques significantly outperform the sequential scan and the M-tree index for all result set sizes. This observation is mainly due to the fact that the filtering techniques reduce the number of necessary distance calculations far more than the M-tree index. This behavior results in speed-up factors between 2.5 and 6.2 compared to the M-tree index and even higher factors compared to a simple sequential scan. This way, our multi-step query processing architecture is a significant improvement over the standard indexing approach.

5.2 Web site graphs

As demonstrated in [20], the degree-2 edit distance is well suitable for approximate website matching. In website management it can be used for searching similar websites. In [21] web site mining is described as a new way to spot competitors, customers and suppliers in the world wide web.

By choosing the main page as the root, one can represent a website as a rooted, labeled, unordered tree. Each node in the tree represents a webpage of the site and is labeled with the URL of that page. All referenced pages are children of that node and the borders of the website where chosen carefully. See figure 14 for an illustration.

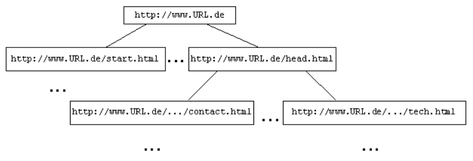

http://www.URL.de
http://www.URL.de/start.html
http://www.URL.de/head.html
http://www.URL.de/.../contact.html
http://www.URL.de/.../tech.html

Fig. 14. Part of a website tree.

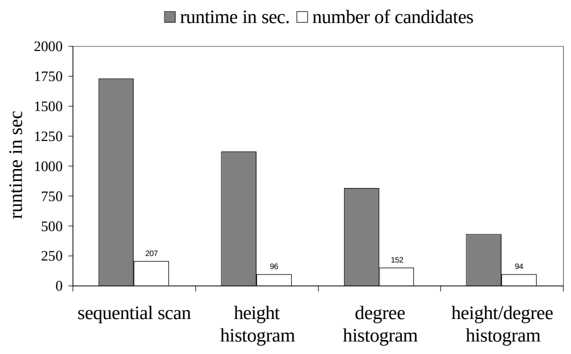


Fig. 15. Average runtime and number of candidates for 5-nn queries.

For our experiment, we used a compressed form of the 207 web sites described in [21], resulting in trees that have 67 nodes on the average. We ran 5-nn-queries on this data. The results are shown in figure 15. We notice that even if the degree filter produces a lot more candidates than the height filter, it results in a better run time. This is due to the fact that it filters out those trees, where the computation of the degree-2 edit distance is especially time-consuming. Using the combination of both histograms, the runtime is reduced by a factor of 4.

6 Conclusions

In this paper, we presented a new approach for efficient similarity search in large databases of tree structures. Based on the degree-2 edit distance as similarity measure, we developed a multi-step query architecture for similarity search in tree structures. For structural as well as for content-based features of unordered attributed trees, we suggested several filter methods. These filter methods significantly reduce the number of complex edit distance calculations necessary for a similarity search. The main idea behind our filter methods is to approximate the distribution of structural and content-based features within a tree by means of feature histograms. Furthermore, we proposed a new technique for folding histograms and a new way to combine different filter methods in order to improve the filter selectivity. We performed extensive experiments on two sets of real data from the domains of image similarity and website mining. Our experiments showed that filtering significantly accelerates the complex task of similarity search for tree-structured objects. Moreover, it turned out that no single feature of a tree is sufficient for effective filtering, but only the combination of structural and content-based filters yields good results.

In our future work, we will explore how different weights for edit operations influence the selectivity of our filter methods. Additionally, we intend to investigate other structural features of trees for their appropriateness in the filter step. In a recent publication [5], an edit distance for XML-documents has been proposed. An interesting question is, how our architecture and filters can be applied to the problem of similarity search in large databases of XML-documents.

References

1. Jiang, T., Wang, L., Zhang, K.: Alignment of trees - an alternative to tree edit. Proc. Int. Conf. on Combinatorial Pattern Matching (CPM), LNCS **807** (1994) 75–86
2. Selkow, S.: The tree-to-tree editing problem. Information Processing Letters **6** (1977) 576–584
3. Zhang, K.: A constrained editing distance between unordered labeled trees. Algorithmica **15** (1996) 205–222
4. Wang, J.T.L., Zhang, K., Chang, G., Shasha, D.: Finding approximate pattersn in undirected acyclic graphs. Pattern Recognition **35** (2002) 473–483
5. Nierman, A., Jagadish, H.V.: Evaluating structural similarity in XML documents. In: Proc. 5th Int. Workshop on the Web and Databases (WebDB 2002), Madison, Wisconsin, USA. (2002) 61–66
6. Sebastian, T.B., Klein, P.N., Kimia, B.B.: Recognition of shapes by editing shock graphs. In: Proc. 8th Int. Conf. on Computer Vision (ICCV'01), Vancouver, BC, Canada. Volume 1. (2001) 755–762
7. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters **19** (1998) 255–259
8. Chartrand, G., Kubicki, G., Schultz, M.: Graph similarity and distance in graphs. Aequationes Mathematicae **55** (1998) 129–145
9. Kubicka, E., Kubicki, G., Vakalis, I.: Using graph distance in object recognition. In: Proc. ACM Computer Science Conference. (1990) 43–48
10. Papadopoulos, A., Manolopoulos, Y.: Structure-based similarity search with graph histograms. In: Proc. DEXA/IWOSS Int. Workshop on Similarity Search. (1999) 174–178
11. Levenshtein, V.: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics-Doklady **10** (1966) 707–710
12. Wagner, R.A., Fisher, M.J.: The string-to-string correction problem. Journal of the ACM **21** (1974) 168–173
13. Zhang, K., Statman, R., Shasha, D.: On the editing distance between unordered labeled trees. Information Processing Letters **42** (1992) 133–139
14. Zhang, K., Wang, J., Shasha, D.: On the editing distance between undirected acyclic graphs. International Journal of Foundations of Computer Science **7** (1996) 43–57
15. Agrawal, R., Faloutsos, C., Swami, A.N.: Efficient similarity search in sequence databases. In: Proc. 4th Int. Conf. of Foundations of Data Organization and Algorithms (FODO). (1993) 69–84
16. Seidl, T., Kriegel, H.P.: Optimal multi-step k-nearest neighbor search. In Haas, L.M., Tiwary, A., eds.: Proc. ACM SIGMOD Int. Conf. on Managment of Data, ACM Press (1998) 154–165
17. Berchtold, S., Keim, D., Kriegel, H.P.: The X-tree: An index structure for high-dimensional data. In: 22nd Conference on Very Large Databases, Bombay, India (1996) 28–39
18. Chavez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.: Searching in metric spaces. ACM Computing Surveys **33** (2001) 273–321
19. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: VLDB'97, Proc. 23rd Int. Conf. on Very Large Databases, August 25-29, 1997, Athens, Greece. (1997) 426–435
20. Wang, J., Zhang, K., Jeong, K., Shasha, D.: A system for approximate tree matching. IEEE Transactions on Knowledge and Data Engineering **6** (1994) 559–571
21. Ester, M., Kriegel, H.P., Schubert, M.: Web site mining: A new way to spot competitors, customers and suppliers in the world wide web. In: Proc. 8th Int. Conf on Knowledge Discovery in Databases (SIGKDD'02), Edmonton, Alberta, Canada. (2002) 249–258