# A Survey on Tree Edit Distance Lower Bound Estimation Techniques for Similarity Join on XML Data

Fei Li
Harbin Institute of Technology
lifei@umich.edu

Hongzhi Wang*
Harbin Institute of Technology
wangzh@hit.edu.cn

Jianzhong Li
Harbin Institute of Technology
lijzh@hit.edu.cn

Hong Gao
Harbin Institute of Technology
honggao@hit.edu.cn

## ABSTRACT

When integrating tree-structured data from autonomous and heterogeneous sources, exact joins often fail for the same object may be represented differently. Approximate join techniques are often used, in which similar trees are considered describing the same real-world object. A commonly accepted metric to evaluate tree similarity is the tree edit distance. While yielding good results, this metric is computationally complex, thus has limited benefit for large databases. To make the join process efficient, many previous works take filtering and refinement mechanisms. They provide lower bounds for the tree edit distance in order to reduce unnecessary calculations. This work explores some widely accepted filtering and refinement based methods, and combines them to form multi-level filters. Experimental results indicate that string-based lower bounds are tighter yet more computationally complex than set-based lower bounds, and multi-level filters provide the tightest lower bound efficiently.

## 1. INTRODUCTION

For the ability to represent data from heterogeneous sources, XML is widely used for web data representation and exchange. For its flexibility, data representing the same object may not be exactly the same. For duplication detection and data integration, approximate join techniques are in demand. That is, similar XML fragments are joined for they are considered as representing the same real-world object.

XML fragments are often modeled as ordered labeled trees. Tree edit distance is a widely used metric to evaluate the similarity between trees [17]. The tree edit distance is the minimum number of node insertions, deletions, or relabels to transform one tree to another [1]. Two trees are considered as a *similar tree pair* if their tree edit distance is below a predefined threshold. It is effective but computationally expensive. Many researches have been performed to improve the efficiency [22, 14, 7, 8]. Unfortunately, the time complexity is still at least $O(n^3)$, where $n$ is the tree size. When there are large numbers of trees and the trees are huge, the join process needs a lot of time.

Filtering and refinement mechanisms are often used to overcome this problem. The main idea is to compute lower bounds for tree edit distances and filter out *dissimilar tree pairs* without computing their exact tree edit distances. Since lower bounds are much easier to compute than the exact value, the overall efficiency is improved significantly.

To our knowledge, existing lower bounds are computed based on transformation. Trees are transformed into other data structures whose distances serve as lower bounds to tree edit distance. String is a relatively simple data structure which contains order for structure as well as content information in each entry. In [10], XML documents are transformed into their corresponding preorder and postorder traversal sequences. Then the string edit distance is used as the lower bound of the tree edit distance. This method has high filter quality but relatively low efficiency. Set (multi-set) is even simpler than string. In [12], three kinds of histograms are proposed based on the node height (leaf height), node degree, and node label, respectively, to compute relatively rough lower bounds. In the method of binary branch [21], trees are transformed into binary branch sets and the binary branch distance between these sets is used to compute the lower bound of the tree edit distance. These two set-based methods are

*corresponding author

[1]In this article, we mainly discuss unit cost tree edit distance, in which all operations have the same cost.

very efficient but can not provide the lower bound as tight as the string-based methods do.

Since all the lower bounds of tree edit distance are definitely lower than or equal to the exact tree edit distance, these methods can be combined to give tighter lower bounds. The maximum value of all the lower bounds in different methods serves as the tightest lower bound. Instead of computing all the lower bounds independently, a multi-level filtering mechanism can be applied. Efficient lower bounds are computed first to wipe some dissimilar tree pairs out. Then more expensive yet tighter lower bound are computed only for the remaining tree pairs. After all the lower bound methods are applied, tree edit distance is compute for the remaining tree pairs. While having the same filtering quality, multi-level filter is conducted more efficiently than computing all lower bounds independently and choosing the highest one.

**Contributions:** This paper presents a comparative study of these filtering and refinement methods for tree similarity join. We implement the string-based lower bounds [10], Histogram [12], and binary branch distance [21] respectively to test the bound tightness and computational efficiency. From the comparisons, each of these three methods has special benefits. As a result, they could be combined to form a multi-level filter to achieve tighter lower bound efficiently. Such a combined mechanism could be more effective and efficient than each single one.

The rest of the paper is organized as follows. In Section 2, related work is discussed. In Section 3, some background knowledge is introduced. Three widely accepted methods in computing the lower bound of tree edit distance are described in detail in Section 4. We analyze the properties of each method in Section 5. The combined strategy is discussed in Section 6. We test the efficiency and effectiveness of each method experimentally in Section 7. The conclusions are drawn in Section 8.

## 2. RELATED WORK

Approximate joining techniques for trees are often based on similarity evaluation. A well-known distance function for trees is the tree edit distance. To describe time complexity, we use $n$, $l$, and $h$ to denote the number of nodes, leaves, and the height of a tree, respectively. [17] presented the first algorithm for computing tree edit distance in time $O(n^2l^4)$. [22] improved this result to $O(n^2min^2(l,h))$ running time with $O(n^4)$ in the worst case. [14] improved it to $O(n^3 \log n)$. Both [22] and [14] achieved their improvements based on closely related dynamic programming, presenting different ways to compute only a subset of relevant subproblems. [7] presented a different approach based on the results of fast matrix multiplication and give an algorithm with time complexity $O(n^{3.5})$ in the worst case. A recent development is by [8] which compute the tree edit distance in time $O(n^3)$.

Obviously, the tree edit distance computation is expensive and does not scale for large trees in massive data-sets. Therefore, many previous works take the filtering and refinement mechanisms to accelerate the similarity join process. In the filtering step, many pairs of dissimilar trees are filtered out. In the refinement step, tree edit distance is only computed for the remaining tree pairs. The overall join process is accelerated since fewer tree edit distances need to be computed directly.

To the best of our knowledge, existing filtering and refinement methods are based on transformation. Trees are transformed into simpler data structures whose distance is lower than the tree edit distance but much easier to compute. String is a relatively simple data structure that contains order for structure information as well as content information in each entry. In [10], XML documents are transformed into their corresponding preorder (or postorder) traversal sequences. Then the string edit distance between two sequences serves as the lower bound of their tree edit distance. [2, 1] use half of the string edit distance between Euler traversals as the lower bound of the tree edit distance. However, this lower bound is often lower than the maximum of the two lower bounds (provided by their preorder traversal sequences and postorder traversal sequences, respectively) proposed in [10], thus cannot be tighter lower bounds in most cases. The Euler traversal is twice as long as the preorder (or postorder) traversal, which would cause 4 times in running time. So we use [10] to represent string based lower bounds.

Set (multi-set) is a even simpler data structure. In Histogram [12], three kinds of histograms are proposed based on the node height, node degree, and node label, respectively, to compute rough lower bounds for tree edit distance. Another set-based method is Binary branch [21]. In that method, trees are first transformed into binary trees and then into sets. The binary branch distance between these sets is then used to compute the lower bound of the tree edit distance.

Recently, some works adopted different distance functions to evaluate the similarity between trees directly. *pq*-gram distance is first proposed to evaluate the distance between ordered trees directly [4].

In [3], the *pq*-gram method is extended to evaluate the similarity between unordered trees. Recently in [18], each tree is transformed into a set of pivots and the Jaccard Coefficient between two sets of pivots are used to approximate the tree edit distance. As is shown in [18], for unordered trees, their method approximates tree edit distance more accurately than *pq*-gram. In the case of ordered trees, their matching quality is lower than that using *pq*-gram [11]. These methods are proposed to evaluate the similarity between trees directly. Although some of them approximate tree edit distance well, they do not have any guarantee of being lower bounds to tree edit distance. Hence we do not discuss these methods in this paper. Later in [5], the *pq*-gram distance is modified to serve as a lower bound to the fanout-weighted tree edit distance, but not to the widely used unit cost tree edit distance or general case. We do not consider it in this paper.

## 3. PRELIMINARY DEFINITIONS

Definition 1. (Tree Edit Distance). *Given a pair of trees $T_1$ and $T_2$, the tree edit distance between them is the minimum cost of a series of tree edit operations to transform one into another. The three standard tree edit operations [17] includes:*

1. *relabeling (changing the label) a node $v$.*
2. *deleting a node $v$ (and moving all the children of $v$ to $v$'s parent).*
3. *inserting a node $v$ to $w$ (and moving a contiguous sequence of $w$'s children under $v$).*

In order to determine the distance between trees, a cost model must be defined. In this paper, we discuss the unit cost model: the cost of each standard operation is 1. We use the symbol $TD(T_1, T_2)$ to denote the unit tree edit distance between $T_1$ and $T_2$.

Definition 2. (Approximate Join on Trees). *]Given two tree sets, $F_1$ and $F_2$, the Join between $F_1$ and $F_2$ on Tree Edit Distance is the set $\{(T_i, T_j)|(T_i, T_j) \in F_1 \times F_2, TD(T_i, T_j) \leq \tau\}$, where $\tau$ is a predefined threshold.*

Example 1. *Figure 1 shows two tree sets $F_1 = \{T_{11}, T_{12}\}$ and $F_2 = \{T_{21}, T_{22}\}$. Suppose the predefined threshold is 2. Only $TD(T_{11}, T_{21})$ and $TD(T_{12}, T_{22})$ are lower or equal to that threshold. Then the tree edit distance join on them is $\{(T_{11}, T_{21}), (T_{12}, T_{22})\}$.*

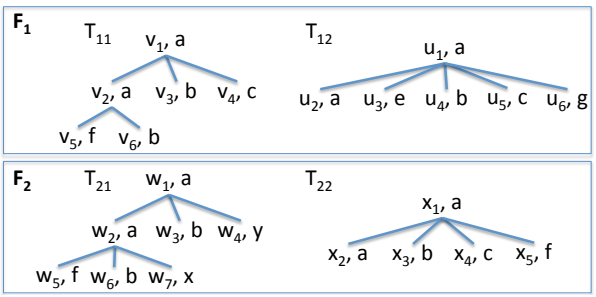



**Figure 1: Approximate Tree Matching**

Join based on tree edit distance is effective but computationally expensive. To accelerate the efficiency, filtering and refinement mechanisms are often used. That is, if the lower bound of the tree edit distance is above the predefined threshold, that tree pair must be dissimilar and can be safely eliminated. Since lower bounds are much easier to be computed than the tree edit distance, the whole join efficiency is improved significantly.

## 4. LOWER BOUNDS FOR TREE EDIT DISTANCE

In this section, we introduce three commonly accepted methods for computing the lower bounds of tree edit distance: string-based lower bound [10], histogram [12] and binary branch distance [21].

### 4.1 String-based Lower Bound

Let $T$ be an ordered labeled tree, where $pre(T)$ and $post(T)$ are the preorder and postorder traversals of $T$, respectively. Both $pre(T)$ and $post(T)$ are viewed as strings. With $ed(s_1, s_2)$ denoting the edit distance between two strings, the relationship between the unit tree edit distance and the string edit distance is shown as follows:

$$ed(pre(T_1), pre(T_2)) \leq TD(T_1, T_2)$$

$$ed(post(T_1), post(T_2)) \leq TD(T_1, T_2)$$

Example 2. *Figure 2 shows the preorder and postorder of $T_{12}$ and $T_{21}$ in Figure 1. Suppose the predefined threshold is 2. Since $ed(pre(T_{12}), pre(T_{21}))$ = 4 and $ed(post(T_{12}), post(T_{21}))$ = 6, $TD(T_{12}, T_{21})$ is at least 6. So $T_{12}$ and $T_{21}$ are definitely dissimilar.*

String edit distance is computed in time $O(n^2)$, where $n$ is the tree size. This is much faster than computing the tree edit distance. However, when the trees in the databases is too large, the computation of the string edit distance is also costly. Vectors and sets (bags) are data structures even simpler

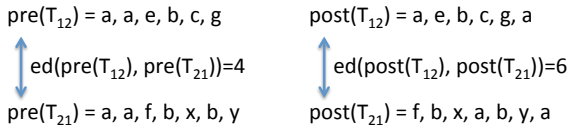



**Figure 2: String-based Lower Bound**

than strings. So many following researches transform trees into vectors or sets to estimate the lower bound faster [12, 21, 5].

## 4.2 Histogram

Histogram was firstly proposed in [12] to compute lower bounds for the tree edit distance efficiently. In their method, three kinds of histograms (leaf distance histogram, degree histogram, and label histogram) are developed. The basic idea of all these methods is to transform trees into vectors and use the $L_1$ distance between these vectors to estimate the lower bounds.

### 4.2.1 Leaf Distance Histogram

The height of the nodes in a tree is an important structural property. Leaf distance histogram defines the height from the leaves to the root as follows:

DEFINITION 3. (LEAF DISTANCE). *The leaf distance $d_l(v)$ of a node $v$ is the maximum length of a path from $v$ to any leaf node in the subtree rooted at $v$.*

DEFINITION 4. (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 entry $i \in 0, ..., k$ is the number of nodes that share the leaf distance $i$, i.e. $h_l(T)[i] = |v \in T, d_l(v) = i|$.*

THEOREM 1. *For any two trees $T_1$ and $T_2$, the $L_1$-distance of the leaf distance histogram is a lower bound of the edit distance between $T_1$ and $T_2$ [12]. That is:*

$$L_1(h_l(T_1), h_l(T_2)) \leq TD(T_1, T_2).$$

EXAMPLE 3. *Figure 3 shows the leaf distance histogram of $T_{12}$ and $T_{21}$ in Figure 1. Take $T_{21}$ as an example. $T_{21}$ has 5 nodes with leaf distance 5 ($w_3$, $w_4$, $w_5$, $w_6$, $w_7$), 1 node with leaf distance 1 ($w_2$), and 1 node with leaf distance 2 ($w_1$). So the leaf distance histogram of $T_{21}$ is (5, 1, 1). Suppose the predefined threshold is 2. If we use leaf distance histogram to compute a lower bound, which is $L_1(h_l(T_{12}), h_l(T_{21})) = 1$, we can not filter this dissimilar tree pair off.*

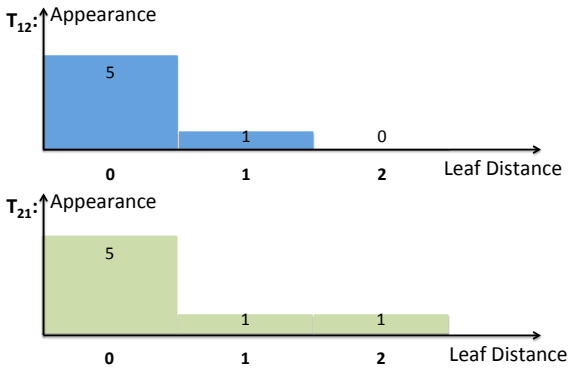



**Figure 3: Leaf Distance Histogram**

### 4.2.2 Degree Histogram

The degrees of the nodes are another structural property. The degree histogram uses the information of node degrees and gives a rough lower bound for the unit tree edit distance.

DEFINITION 5. (DEGREE HISTOGRAM). *The degree histogram $h_d(T)$ of a tree $T$ is a vector with length $k = 1 + degree_{max}(T)$ where the value of any entry $i \in 0, ..., k$ is the number of nodes that share the degree $i$, i.e. $h_d(T)[i] = |v \in T, degree(v) = i|$.*

THEOREM 2. *$L_1(h_d(T_1), h_d(T_2))/3$ provide a lower bound for the edit distance between two trees $T_1$ and $T_2$ [12]. That is:*

$$\frac{L_1(h_d(T_1), h_d(T_2))}{3} \leq TD(T_1, T_2).$$

EXAMPLE 4. *Figure 4 shows the degree histogram of $T_{12}$ and $T_{21}$ in Figure 1. Take $T_{21}$ as an example. $T_{21}$ has 5 nodes in 0 degree ($w_3$, $w_4$, $w_5$, $w_6$, $w_7$), 2 nodes in 3 degree ($w_1$, $w_2$). So the degree histogram of $T_{21}$ is (5, 0, 0, 2, 0, 0). Suppose the predefined threshold is 2. Since $\frac{L_1(h_d(T_{12}), h_d(T_{21}))}{3}$=3/3=1, we can only tell that the tree edit distance between $T_{12}$ and $T_{21}$ is at least 1. In this example, similar to leaf distance histogram, degree histogram cannot filter this tree pair off either.*

### 4.2.3 Label Histogram

Apart from the structure information, the content features, which are stored as tree labels, can also be used to distinguish dissimilar trees. Intuitively, if two trees share many labels, they are very likely to be similar.

DEFINITION 6. (LABEL HISTOGRAM). *The label histogram $h_{lab}(T)$ of a tree $T$ is a (multi-)set consists of all the node labels in $T$.*

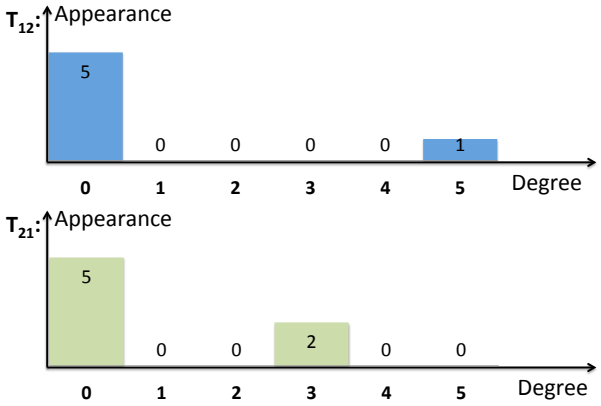



**Figure 4: Degree Histogram**

THEOREM 3. *For two trees $T_1$ and $T_2$ [12]:*

$$\frac{L_1(h_{lab}(T_1), h_{lab}(T_2))}{2} \leq TD(T_1, T_2).$$

EXAMPLE 5. *The label histogram of $T_{12}$ in Figure 1 is {a, a, e, b, c, g}, while the histogram of $T_{21}$ is {a, a, b, y, f, b, x}. Suppose the predefined threshold is 2. The lower bound provided by label histogram is $|h_{lab}(T_{12}) \bigcup h_{lab}(T_{21})$ - $h_{lab}(T_{12}) \bigcap h_{lab}(T_{21})|/2 = 4$, which is higher than the threshold. In this example, label histogram successfully filters this tree pair off.*

Label histogram can effectively filter the tree pairs whose node labels are very different. And, in most cases, label histogram can provide a much tighter lower bound than leaf distance histogram and degree histogram.

For a pair of trees with $n$ nodes, height $h$ and degree $d$, the length of their leaf distance histogram and degree histogram is $h + 1$ and $d + 1$, respectively. Thus their $L_1$ distance can be computed in time $O(h)$ and $O(d)$. Also, the size of each label histogram is $n$, thus the symmetric difference between them can be computed in time $O(n \log n)$. Furthermore, in the case of similarity join two tree sets, the efficiency can be further enhanced when applying some well-known techniques (e.g., sort merge and hash join) to avoid nested loop. So all the three kinds of histograms can give rough lower bounds and wipe out some dissimilar trees very efficiently.

## 4.3 Binary Branch Distance

Leaf distance histogram and degree histogram consider only structural information while the label histogram considers only content information. Thus they can only give relatively rough lower bounds. Binary branch [21] is a set-based method which considers both structure and content information at the same time.

There is a natural correspondence between a tree and its binary tree. For each node in a tree, its left most child (if any) becomes it left child in its binary tree while its right sibling (if any) becomes its right child in its binary tree. In this paper, we use the symbol $B(T)$ to denote the binary tree transformed from $T$.

EXAMPLE 6. *Figure 5 shows binary trees $B(T_{12})$ and $B(T_{21})$ of $T_{12}$ and $T_{21}$, respectively. Note that we further transform each binary tree into a full binary tree by adding dummy nodes (labeled $*$).*

DEFINITION 7. (BINARY BRANCH). *Let $B$ be a binary tree. $\forall u \in B$ has a binary branch $Br(u)$ composed by $u$ and its two children.*

DEFINITION 8. (BINARY BRANCH VECTOR). *A binary branch vector $BRV(T)$ of a tree $T$ is a vector $(b_1, b_2, ..., b_{|B|})$, with each element $b_i$ representing the number of occurrences of the $i$th binary branch. $|B|$ is the size of the binary branch space of the dataset.*

DEFINITION 9 (BINARY BRANCH DISTANCE). *Let $BRV(T_1)$=$(b_1, b_2, ..., b_{|B|})$, $BRV(T_2) = (b'_1, b'_2, ..., b'_{|B|})$ be the binary branch vectors of tree $T_1$ and $T_2$, respectively. Their binary branch distance is $BDist(T_1, T_2) = \Sigma_{i=1}^{B} |b_i - b'_i|$.*

THEOREM 4. *For any two trees $T_1$ and $T_2$ [21]:*

$$\frac{BDist(T_1, T_2)}{5} \leq TD(T_1, T_2).$$

EXAMPLE 7. *Figure 6 shows all binary branches of $T_{12}$ and $T_{21}$. Their corresponding binary branch vectors are shown in Figure 7. The binary branch distance between the two binary branch vectors is 11. Thus the estimate lower bound is 3.*

# 5. ANALYSIS

## 5.1 Running Time

In the previous section, we describe altogether 6 lower bound functions: two string-based lower bounds and four set-based lower bounds. For string-based lower bounds, the computation of string edit distance needs potentially quadratic time. The latter 4 distance function compute the $L_1$ distance between vectors, which is equal to compute the symmetric difference between the (multi-)sets of the entries in these vectors. The symmetric difference between the (multi-)sets can be computed in time $O(n \log n)$, which is much faster than the $O(n^2)$

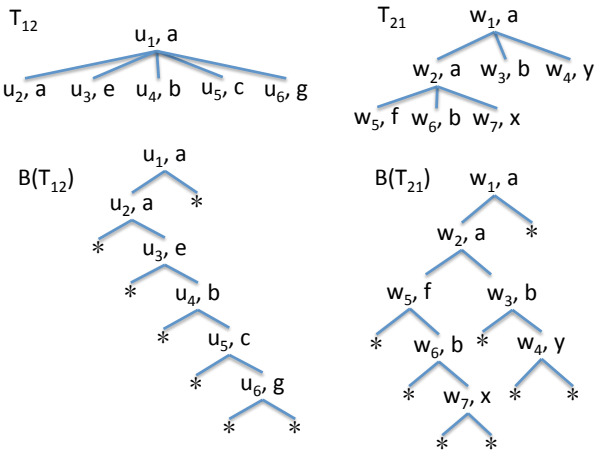



**Figure 5: Binary Tree of $T_{12}$ and $T_{21}$**

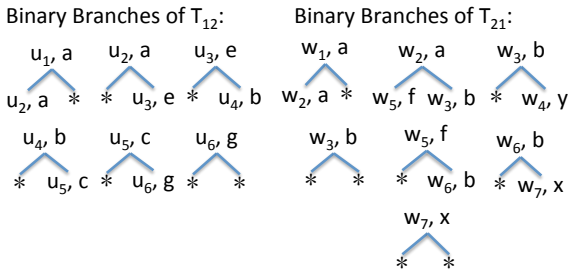



**Figure 6: Binary Branches of $T_{12}$ and $T_{21}$**

of computing the string edit distance. When applying well-known techniques (e.g., sort merge and hash join), all the lower bounds between each tree pair in $F_1 \times F_2$ can be computed without nested-loop. This makes the filtering process using set-based method much more efficient than that based on strings. Here we take the label histogram distance as an example to discuss the efficiency of set-based join methods.

Suppose that $F_1$ and $F_2$ are two sets of XML fragments. The goal of filtering process is to find all the tree pairs in $F_1 \times F_2$ with lower bound within the threshold $\tau$. Algorithm 1 describes the filtering process. All the trees are firstly transformed into node-sets (multi-sets). Then we merge all the node-sets transformed from trees in $F_i$ into $List_i$ (line 3). Note that the two $List$s are lists sorted by the $label - value$ of each node (to be brief, all alphanumeric labels are converted to number labels method [13]). The size of each node-set in $F_i$ is computed and stored in the $List$s (line 4). We check for each node label in which pairs of trees it appears and count the number of node labels that each tree pair shares (line 5-6). That number equals to the size of the intersection of a pair of node-sets. The

| a,a,*\|a,*,e\|a,f,b\|b,*,c\|b,*,x\|b,*,y\|c,*,g\|e,*,b\|f,*,b\|g,*,*\|x,*,*\|y,*,* | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| BRV($T_{12}$): | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| BRV($T_{21}$): | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |

**Figure 7: Binary Branch Vectors of $T_{12}$ and $T_{21}$**

sum of the size of two trees minus twice the size of their intersection equals to the size of symmetric difference(line 7). Then all the lower bounds of edit distance between in $F_1 \times F_2$ are computed without nested-loop (line 7). The tree pairs with lower bound lower than $\tau$ are returned (line 7).

**Algorithm 1 Filtering Algorithm for Set or Vector Based Methods**

**Input:** $F_1$, $F_2$, $\tau$
**Output:** $CandidateTreePairs$

1: **for all** trees in $F_i$ **do**
2: **for** all the node labels in this tree **do**
3: $List_i = List_i \uplus (tID_i, \text{label-value}, count_i)$
4: **end for**
5: **end for**
6: $\Gamma_{tID_i, SUM(count_i) \rightarrow size_i}(List_i)$
7: $List' = List_1 \bowtie List_2$
8: $List'' = \Gamma_{tId_1, tId_2, sum(min(count_1, count_2)) \rightarrow \cap}(List')$
9: $candidate \leftarrow \pi_{tId_1, tId_2}(\sigma_{\frac{size_1 + size_2 - 2*\cap}{2} \leq \tau}(List''))$
10: **return** $candidate$

To be brief, it is supposed that the two XML sets have $N$ trees for each and all the trees have $n$ nodes. To analyze the time complexity, we summary the filtering algorithm algorithm to two steps:

1. All the trees are transformed to their corresponding node sets.

2. Sort-merge and hash join is applied to the sets and the tree pairs with lower bound distance lower than $\tau$ are returned.

In the first step, since each tree can be transformed to its corresponding label set in time $O(n)$, the running time in the first step is $O(Nn)$. In the second step, the diversity of the trees would affect the running time. In the best case, when no tree pair shares any element, the run time in this step is the time of merging all sets into $List_1$ and $List_2$. That is $O(Nnlog(Nn))$. In the worst case, when all the transformed sets are exactly the same. Each element in one List would match $N$ tuples in the other List. Thus the run time is $O(Nnlog(Nn) + N^2n)$. From our experiments on various real-world data sets, the running time in this step is usually close to the best case. Therefore, the average time complex-

| Methods | Worst Case | Average |
|---|---|---|
| String-based | $N^2n^2$ | $N^2n^2$ |
| Leaf Distance Histogram | $Nh*log(Nh) + N^2h$ | $Nh*log(Nh)$ |
| Degree Histogram | $Nd*log(Nd) + N^2d$ | $Nd*log(Nd)$ |
| Label Histogram | $Nn*log(Nn) + N^2n$ | $Nn*log(Nn)$ |
| Binary Branch | $Nn*log(Nn) + N^2n$ | $Nn*log(Nn)$ |

**Figure 8: Time Complexity of each Method ($N$ denotes the number of trees in each data source, $n$ denotes the number of nodes in each tree, $h$ denotes the height of a tree, and $d$ denotes the highest fanout of a tree).**

ity of set-based filtering algorithm can be estimated as $O(Nnlog(Nn))$.

The time complexity of each method is summarized in Figure 8.

## 5.2 Tightness of each Lower Bound

Since different lower bound functions are suitable in different cases, it is hard to analyze the overall tightness of each lower bound theoretically. Intuitively, leaf distance histogram and degree histogram give very rough lower bounds, while label histogram and binary branch provide much tighter lower bounds. Except in some extreme examples, the string-based lower bounds are much tighter than set-based lower bound functions. Here, we analyze the tightness of each lower bound function using extreme examples.

### 5.2.1 Leaf Distance Histogram and Degree Histogram

Leaf distance histogram (degree histogram) can only detect the differences in leaf distance (degree) information between trees but entirely disregard the label information. As long as the leaf distance (degree) information between trees are similar, leaf distance histogram (degree histogram) cannot detect the distance. Here we illustrate this point in two examples.

EXAMPLE 8. *In Figure 9, $T_1$ and $T_2$ are very different trees, but their leaf distance histograms are exactly the same. Both of them have 5 nodes ($v_4$, $v_5$, $v_6$, $v_7$, $v_8$ in $T_1$ and $w_4$, $w_5$, $w_6$, $w_7$, $w_8$ in $T_2$) at leaf height 0, 2 nodes ($v_2$, $v_3$ in $T_1$ and $w_2$, $w_3$ in $T_2$) at leaf height 1, and one node ($v_1$ in $T_1$ and $w_1$ in $T_2$) at leaf height 2. Leaf distance histograms fail to detect the differences between these two trees, thus cannot provide tight lower bounds in this case. In Figure 10, $T_3$ and $T_4$ are also very different trees. Using degree histogram, their label and structural differences cannot be detected at all, since the two trees have exactly the same degree histograms: one node ($v_1$ in $T_3$ and $w_3$ in $T_4$) has 3 children, one node ($v_2$ in $T_3$ and $w_1$ in $T_4$) has 2 children, and other five nodes do not have children.*

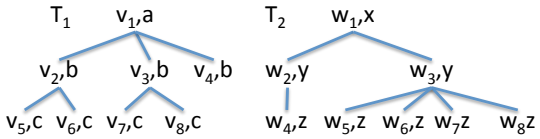



**Figure 9: Mismatch using Leaf Distance Histogram**

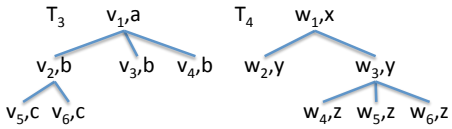



**Figure 10: Mismatch using Degree Histogram**

### 5.2.2 Label Histogram

Since the label histogram only considers the label information of trees, it cannot work well when most changes are structural changes.

EXAMPLE 9. *In Figure 11, $T_5$ and $T_6$ are very different trees. But their histograms are exactly the same since they share the same label set. So in this case, label histograms fail to provide a tight lower bound.*

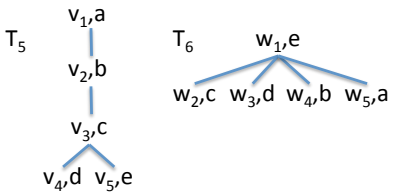



**Figure 11: Mismatch using Label Histogram**

### 5.2.3 Binary Branch

Although both label and structure information are considered, the lower bounds provided by Binary Branch is also rough.

THEOREM 5. *Let $T_1$ and $T_2$ be two trees with $n$ and $m$ nodes, respectively. The lower bound distance between $T_1$ and $T_2$ provided by the binary branch is at most $0.2(n+m)$.*

PROOF. The number of binary branch of a tree equals to the tree size. The binary branch distance between them is at most $n+m$ (only in the case

that the two trees share no binary branch). Thus the lower bound distance provide by binary branch is at most $0.2(n+m)$. □

In many cases, especially when the predefined threshold is above $0.2(m+n)$, the method binary branch cannot filter out any tree pairs. Now we analyze the provided lower bounds in different cases. A small change in a node would affect all its binary branches. Consider the ratio between binary branch distance and number of changed nodes. This ratio would be higher in the case of many small changes than a big change (a subtree move or deletion). Thus the lower bounds provided by binary branch would be relatively tighter in the cases when a lot of small differences exist between trees.

#### 5.2.4 String-based Lower Bound

String is a data structure which contains order for structure as well as content information in each entry. Although computationally more complex than sets, in most cases, string-based lower bounds are tight. Here we illustrate this point intuitively. Since the discussed trees are ordered trees, the child order information is important to identify similar trees. In both preorder and postorder traversal, the child order information is fully contained. Also, the label of each tree node appears exactly once, which describes the label information properly. The hierarchical information of trees is the most difficult information to describe. To describe the hierarchical information, string-based lower bounds use two kinds of traversals: preorder traversal and postorder traversal, in which each node is visited before (after) all its children are visited. Since the maximum value of the two string edit distance is chosen as the lower bound, the lower bound is not accurate only when neither of the two traversals can properly describe the hierarchical information.

Also, string-based lower bound is always no worse than that provided by label histogram. That is because label differences cost the same in both string-based method and label histogram while string-based lower bound also detects some structure differences.

## 6. COMBINING FILTERING METHODS

Since lower bounds are definitely lower than the exact tree edit distance, the lower bound which has the maximum value is the one closest to the exact value. This inspires us to use different methods to compute different lower bounds and use the maximum lower bound as the final lower bound.

Definition 10. (Combined Distance Function). *Let $D = d_i$ (i from 1 to n) be a set of lower bound distance functions. The combined distance function $d_c$ is defined as the maximum of the component functions:*

$$d_C(T_1, T_2) = max\{d_i(T_1, T_2) | 1 < i < n\}.$$

Also, a tree pair is definitely dissimilar if any of its lower bounds is higher than the threshold. In the case of approximate join two tree sets, we can first use some very efficient yet rough lower bound functions to wipe out a large part of dissimilar tree pairs and then use a slower yet more accurate filtering method to further filter the remained tree pairs.

Definition 11. (Multi-level filtering). *Let $F_1$ and $F_2$ be two tree sets, $\tau$ be the threshold, $D = d_i$ (i from 1 to n) be a list of lower bound distance function (or combined distance function), $C_i$ be tree pairs remained after the ith filtering. The result of multi-level filtering method using $D$ is $C_n$ where $C_0$=$F_1 \times F_2$, $C_i = \{t | t \in C_{i-1}, d_i(t) \leq \tau\}$.*

The filtering effect of using multi-level filtering method is equal to using all the methods one by one, while saving much of the overall running time. Since set-based lower bounds functions are computed more efficiently. All set-based functions are combined to form a set-based combined distance function, which serves as the first round of filter. Two string-based functions $ed(pre(T_1), pre(T_2))$ and $ed(post(T_1), post(T_2))$ are used as the second and third distance function, respectively. The overall efficiency is enhanced, since most dissimiar tree pairs are wiped out in the first round.

## 7. EXPERIMENTS

In this section, we test the efficiency and effectiveness of all the reviewed lower bound functions and the combined lower bound functions. All of our experiments were performed on a PC with Intel Core Duo 2GHz, 1GB main memory and 250GB hard disk. The OS is Windows XP Professional. We implemented our experiments using CodeBlocks.

We use four real-world data sets ranging from apartment data (street), bioinformatics (Swissprot), linguistics (Treebank), and bibliography (DBLP).

- **Street:** We use the application data from the Municipality of Bozen. The scene is that the Office wants to integrate the apartment data stored in two databases and display that information on a map. The data is hierarchically organized, in which the root of the tree is the street name, the children of the street name are the house numbers, the children of house

numbers are the entrance numbers, and the children of entrance numbers are the apartment numbers. We choose subsets from them which has 100 trees for each. The tree size is from 50 to 200. We denote the two sources as $R$ and $L$.

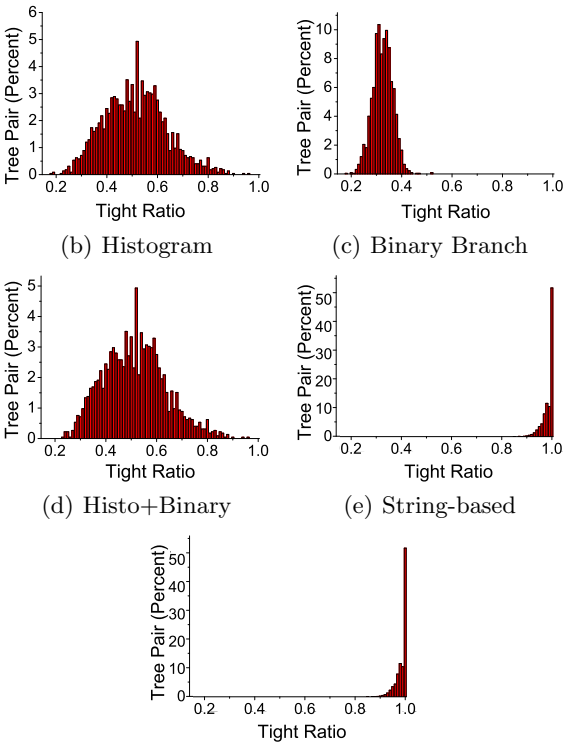
- **SwissProt:** SwissProt is a database which describe protein sequence. Each SwissProt document contains trees with about 100 nodes and about 4 depth on average. The documents in SwissProt show high degree of similarity for they share large numbers of labels.
- **TreeBank:** TreeBank is a database storing parts of speech tagged English sentences. Its documents have deep recursive structure (about 50 nodes and about 7 depth on average).
- **DBLP:** DBLP is a bibliography database that consists of large numbers of small and flat documents (about 15 nodes and 2 depth on average).

## 7.1 Tightness of each Lower Bound

In this section, we test the tightness of each lower bound function. For two trees $T_1$ and $T_2$, $TD(T_1, T_2)$ is the exact tree edit distance, while $d_i(T_1, T_2)$ is the lower bound provided by $i$th method. We use *tight ratio* ($tr = \frac{d_i(T_1,T_2)}{TD(T_1,T_2)}$) to evaluate the tightness for each lower bound. The closer the tight ratio is to 1, the tighter the lower bound is. The average tight ration for each database is shown in Figure 12(a). The detailed tight ratios for the street database are shown in Figure 12(b) to Figure 12(f). Binary branch serves as the roughest lower bound distance while histogram often gives much tighter lower bounds. String-based lower bounds always provide the most accurate lower bounds. The combination of histogram and binary branch performs slightly better than that of histogram, while the combination of all the lower bounds slightly outperforms string-based lower bounds.

| | Histogram | Binary | Histo+Binary | String | All Methods |
|---|---|---|---|---|---|
| Street | 0.502 | 0.310 | 0.503 | 0.983 | 0.983 |
| SwissProt | 0.795 | 0.344 | 0.795 | 0.998 | 0.998 |
| TreeBank | 0.748 | 0.325 | 0.748 | 0.996 | 0.996 |
| DBLP | 0.894 | 0.414 | 0.894 | 1.000 | 1.000 |

(a) Average Tightness in each Database




(b) Histogram (c) Binary Branch (d) Histo+Binary (e) String-based (f) All the Methods

**Figure 12: Tightness of each Lower Bound Function**

## 7.2 Filter Quality

In this section, we test the filtering quality provided by each method. For two tree sets $F_1$ and $F_2$ and a threshold $\tau$, the goal of the filtering process is to wipe out as many dissimilar tree pairs as possible. In this section, we test the size of remaining tree sets provided by each method. We also compute the size of exact result by computing the exact edit distance for the smallest remaining set. The closer the size of remaining set to the result size, the higher filter quality is. We compute the size of remaining set for different thresholds from $0.05(m+n)$ to $0.5(m+n)$ in all the databases, where $m$ and $n$ are the size of the two current trees. The average size of remaining set is shown in Figure 13. The sizes of remaining set under different thresholds are shown in Figure 14. The result is that binary branch has the lowest filter quality while histogram works much better. String-based lower bounds always give the highest filter quality and nearly filter out all the dissimilar tree pairs. The combination of histogram and binary branch performs slightly better than histogram while the combination of all the lower bounds works slightly better string-base lower bounds.

## 7.3 Computing Efficiency

As we analyzed in Section 5, the string-based lower bounds are relatively costly, while set-based lower bounds can be computed efficiently. Also, the multi-level filter, which is at least as effective as

| | Histogram | Binary | Histo+Binary | String | All Methods | Result |
|---|---|---|---|---|---|---|
| Street | 1441 | 1808 | 1440 | 543 | 543 | 526 |
| SwissProt | 220 | 649 | 220 | 108 | 108 | 107 |
| TreeBank | 288 | 658 | 288 | 105 | 105 | 105 |
| DBLP | 223 | 681 | 223 | 136 | 136 | 136 |

**Figure 13: Average Filter Quality in each Database.**

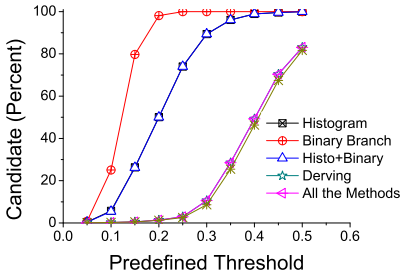



**Figure 14: Detail Filter Quality in Street Database.**

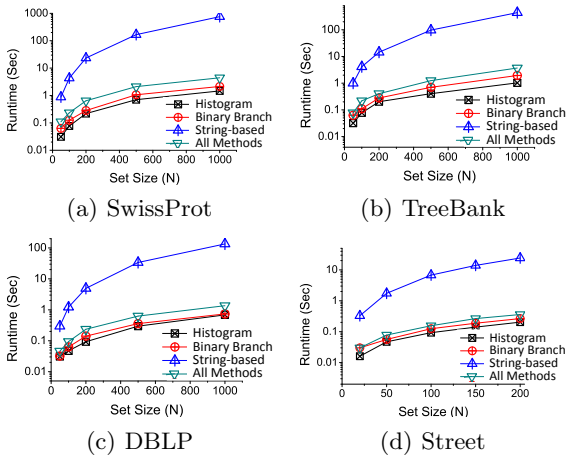
string-based methods since the latter is included by the former, benefits high efficiency. In this section, we test the efficiency of each individual method and the multi-level filter.

We use the Swissprot, Treebank, DBLP and street databases to test the efficiency. We set the threshold $\tau = 0.1(m+n)$ ($m$ and $n$ are the size of the two current trees) and increase the number of trees in each database to test the filter time. In the multi-level filter, all set-based lower bounds are used in the first round while the $ed(pre(T_1), pre(T_2))$ and $ed(post(T_1), post(T_2))$ serve as the second and third filters. The results are shown in figure 15(a) - figure 15(d). Histogram and binary branch have much higher efficiency than string-based lower bounds. The multi-level filter also outperforms string-based lower bounds significantly in efficiency.

## 8. CONCLUSION

In this paper, we have compared and analyzed the performance of string-based lower bounds, histogram, and binary branch for giving the lower bound to tree edit distance. String-based lower bounds is the tightest and thus have the highest filter quality. Although relatively rough, the lower bounds provided by histogram and binary branch can be computed very efficiently. We also combine these methods to form multi-level filters to get tight lower bound efficiently. Experiment results confirm the analytical results.

## 9. ACKNOWLEDGEMENTS




(a) SwissProt (b) TreeBank

(c) DBLP (d) Street

**Figure 15: Filter Efficiency**

Many thanks to Michael H. Böhlen [4, 3, 5] for his source code and test data. He has been a great help in this research. This paper was partially supported by NGFR 973 grant 2012CB316200, NSFC grant 61003046, 61111130189, 61133002 and NGFR 863 grant 2012AA011004. Doctoral Fund of Ministry of Education of China (No. 20102302120054). Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), Ministry of Education (No.KF2011003). the Fundamental Research Funds for the Central Universities(No. HIT. NSRIF. 2013064).

## 10. REFERENCES

[1] Tatsuya Akutsu. A relation between edit distance for ordered trees and edit distance for euler strings. *Inf. Process. Lett.*, 100(3):105–109, 2006.

[2] Tatsuya Akutsu, Daiji Fukagawa, and Atsuhiro Takasu. Approximating tree edit distance through string edit distance. In *ISAAC*, pages 90–99, 2006.

[3] Nikolaus Augsten, Michael H. Böhlen, Curtis E. Dyreson, and Johann Gamper. Approximate joins for data-centric XML. In *ICDE*, pages 814–823, 2008.

[4] Nikolaus Augsten, Michael H. Böhlen, and Johann Gamper. Approximate matching of hierarchical data using *pq*-grams. In *VLDB*, pages 301–312, 2005.

[5] Nikolaus Augsten, Michael H. Böhlen, and Johann Gamper. The *pq*-gram distance between ordered labeled trees. *ACM Trans. Database Syst.*, 35(1): 1–36, 2010.

[6] Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*,

337(1-3):217–239, 2005.

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

[8] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. In *ICALP*, pages 146–157, 2007.

[9] Minos N. Garofalakis and Amit Kumar. XML stream processing using tree-edit distance embeddings. *ACM Trans. Database Syst.*, 30(1):279–332, 2005.

[10] Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. Approximate XML joins. In *SIGMOD Conference*, pages 287–298, 2002.

[11] Fei Li, Hongzhi Wang, Cheng Zhang, Liang Hao, Jianzhong Li, and Hong Gao. Approximate joins for XML using $g$-string. In *XSym*, pages 3–17, 2010.

[12] Karin Kailing, Hans-Peter Kriegel, Stefan Schönauer, and Thomas Seidl. Efficient similarity search for hierarchical data in large databases. In *EDBT*, pages 676–693, 2004.

[13] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[14] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *ESA*, pages 91–102, 1998.

[15] Tetsuji Kuboyama. *Matching and Learning in Trees*. Doctoral Dissertation. The University of Tokyo, 2007.

[16] Bruce A. Shapiro and Kaizhong Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer Applications in the Biosciences*, 6(4):309–318, 1990.

[17] Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[18] Shirish Tatikonda and Srinivasan Parthasarathy. Hashing Tree-Structured Data: Methods and Applications. In *ICDE*, pages 429-440, 2010.

[19] Gabriel Valiente. An efficient bottom-up distance between trees. In *SPIRE*, pages 212–219, 2001.

[20] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[21] Rui Yang, Panos Kalnis, and Anthony K. H. Tung. Similarity evaluation on tree-structured data. In *SIGMOD Conference*, pages 754–765, 2005.

[22] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.