Split-Order Distance for Clustering and Classification Hierarchies

Qi Zhang, Eric Yi Liu, Abhishek Sarkar, and Wei Wang

Department of Computer Science
University of North Carolina at Chapel Hill

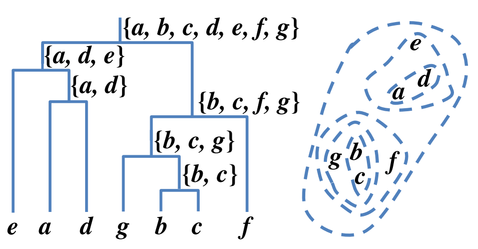
Abstract. Clustering and classification hierarchies are organizational structures of a set of objects. Multiple hierarchies may be derived over the same set of objects, which makes distance computation between hierarchies an important task for summarization and similarity search of hierarchical patterns. In this paper, we model the classification and clustering hierarchies as rooted, leaf-labeled, unordered trees. We propose a novel distance metric Split-Order distance to evaluate the organizational structure difference between two hierarchies over the same set of leaf objects. The Split-Order distance reflects the order in which subsets of the tree leaves are differentiated from each other and can be used to explain the relationships between the leaf objects. We also propose an efficient algorithm for computing Split-Order distance between two trees in $O(n^2d^4)$ time, where n is the number of leaves, and d is the maximum number of children of any node. Our experiments on both real and synthetic data demonstrate the efficiency and effectiveness of our algorithm.

Key words: ENTER KEYWORDS HERE

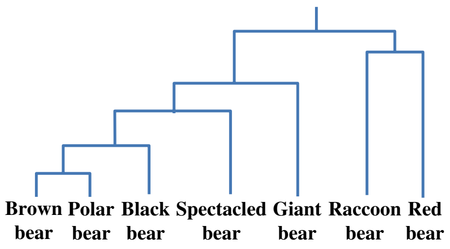
1 Introduction

Clustering and classification hierarchies are important tools for capturing the relationships among objects. Two representative examples are *dendrograms* illustrating the hierarchical clustering result (Fig. 1(a)) and *taxonomies* classifying the biological species (Fig. 1(b)). A hierarchy organizes the set of objects in a tree, where objects with higher similarity are grouped together at a lower level, before objects that are more distant merge into bigger groups. The groups at the intermediate levels represent subclusters or subclasses.

Clustering and classification hierarchies can be automatically constructed given the pair-wise distance matrix over the set of objects. However, different models and methods may yield different estimates for the hierarchical structure. Quantifying the differences between hierarchies becomes crucial for tasks such as summarization of hierarchical patterns, computation of consensus hierarchies, and comparison of hierarchically structured data. A motivating example is the comparison of *phylogenetic taxomonies*, or *phylogeny trees*. A phylogeny tree (Fig. 1(b)) describes the evolutionary relationship between different organisms.



(a) *A dendrogram over 7 objects*



(b) *A taxnomony over 7 species*

Fig. 1. Example clustering and classification hierarchies.

The leaves of the tree represent the species, and the internal nodes correspond to the specialization events, where the evolution diverge in different directions to generate subspecies. The root of the tree is the most recent common ancestor for all the species. Different hierarchies can be derived for a given set of species using different phylogeny tree construction algorithms, such as UPGMA [Sneath73], neighbor-join [Saitou87], maximum parsimony [Felsenstein78], etc. Comparing the similarity of different phylogeny trees is important for evaluating different algorithms and deriving the consensus phylogeny tree.

Many algorithms have been proposed for comparing general tree topologies. Tree edit distance is a classic metric to compare two trees with both internal nodes and leaf nodes labeled. As opposed to a general tree structure, clustering and classification hierarchies are leaf-labeled trees; leaves represent the objects. The tree edit distance computation for unordered trees is NP-complete [Zhang89]. Polynomial-time algorithms [Zhang89,Demaine07,Touzet03,Bille05,Klein98,Lu79,Wang94] only exist for ordered trees. As for classification or clustering hierarchies, there is no specific order among siblings. If we were to consider classification or clustering hierarchies as ordered, fully-labeled trees, metrics such as the tree edit distance may produce results which do not agree with our understanding of the hierarchies. Consider four trees presented in Fig. 2. T_1, T_2, T_3, and T_4 are different hierarchies over a common object set $\{a, b, c, d\}$. To apply the ordered tree edit distance, we also impose labels for the internal nodes. As classification hierarchies, T_2 and T_4 are the same as T_1. The objects are classified in the same way in each of the trees; specifically, the order in which the objects are differentiated from each other is the same. However, the tree edit distance between T_1 and T_2 is 2 since leaves a and b are transposed. The tree edit distance between T_1 and T_4 is also 2 since two internal nodes are transposed. In addition, T_3 has a tree edit distance of 2 from T_1, which implies that T_1, T_2, and T_3 are equidistant from T_1. However, the way in which a and d are classified in T_3 is very different from the way they are classified in T_1. On the other hand, a and d are classified in the same way in both T_1 and T_2.

In this paper, we propose a novel distance metric, Split-Order distance, for comparing clustering/classification hierarchies. Different from a general tree

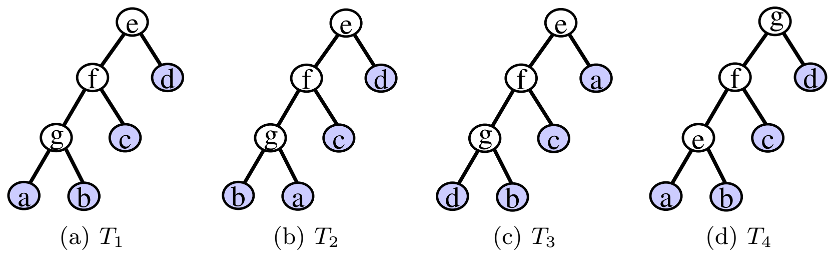


Fig. 2. An example with four trees. T_1, T_2, T_3, and T_4 are four hierarchies with the same set of objects(leaf labels) $\{a, b, c, d\}$. To apply the tree edit distance, we also impose labels for internal nodes. Trees are considered as ordered.

structure, a clustering/classification hierarchy can be modeled as rooted, unordered, leaf-labeled trees. Essential to these hierarchical structures is the set of relationship among the leaf objects captured by the hierarchy: an object is more closely related to another object which it merges with at a lower level than another object which it merges with at a higher level. We refer to a *split* between two objects as the smallest subcluster/subclass in the hierarchy where both objects belong to. A split corresponds to an internal node in the tree which is the most recent common ancestor of the two leaf nodes representing the two objects. All the splits form a partial order which uniquely determines the relationship among all the objects captured by the hierarchy. For example, in Fig. 1(a), the split between O_2 and O_3 happens at a lower level than the split between O_3 and O_6 does; in Fig. 1(b), the split between polar bear and red bear occurs at a higher level than the split between polar bear and brown bear. We define the Split-Order distance between two hierarchical structures based on the order of the splits occuring in the tree. Our contributions can be summarized as follows:

1. We define a novel distance metric, Split-Order distance, between two clustering or classification hierarchies. We prove that Split-Order distance is a metric.
2. We prove that a complete set of split orders uniquely defines a hierarchical structure. In addition, we propose an algorithm for reconstructing the hierarchy using a set of split orders.
3. We propose an efficient algorithm for computing the Split-Order distance between any two hierarchies. Our algorithm takes $O(n^2d^4)$ time, where n is the number of leaves, d is the maximum degree of any node.

The rest of the paper is organized as follows. We review the related work in Section 2 and introduce the preliminaries in Section 3. In Section 4, we discuss the formal definition of Split-Order distance. We present our algorithm for efficient computation of the Split-Order distance in Section 5. The experimental results are reported in Section 6, and Section 7 concludes the paper.

2 Related Work

Many algorithms have been proposed for comparing tree-like or hierarchically structured data. One of the tree distance metrics which has been extensively studied is the tree edit distance. The tree edit distance evaluates the cost of transforming a tree into another tree through a sequence of operations, such as deleting, inserting, and relabeling nodes. A cost is defined for each operation, and the minimum of the total cost of all operations in a sequence is the tree edit distance. Among the different tree edit distance computation algorithms [Demaine07,Touzet03,Bille05,Klein98,Lu79,Wang94,Zhang89], two representative ones are Zhang-Shasha [Zhang89] and Klein [Klein98]. Both algorithms use dynamic programming techniques with worst-case complexities $O(n^4)$ and $O(n^3 logn)$, respectively. Besides tree edit distance, another type of tree distance metrics compare the trees based on the structures they share such as maximum agreement subtrees [Amir97], cousin-pairs [Shasha04], etc. However, as mentioned in the previous section, these general tree distance metrics might not be readily used for comparing clustering or classification hierarchies which are usually modeled as rooted, unordered, leaf-labeled trees, and are able to capture the relationship among leaf objects. For phylogeny trees, several distance metrics have been proposed incorporating biologically meaningful definitions [Wang03,Robinson81,Robinson79,Allen01,Estabrook85]. But they may not work well on general clustering/classification hierarchies.

3 Preliminaries

We model the clustering and classification hierarchies as rooted, unordered, leaf-labeled trees, as shown in Fig. 3.

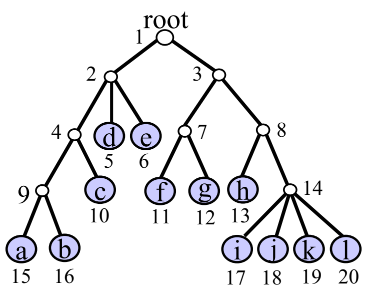


Fig. 3. A rooted, unordered, leaf-labeled tree representing a clustering or classification hierarchy over a set of objects $\Sigma = \{a, b, c, d, e, f, g, h, i, j, k, l\}$.

The leaf nodes represent the objects. Each internal node represents the split of a cluster/class into several subclusters/subclasses. The root represents the entire set of objects. We consider unordered trees, since the sibling order does not exist in classification/clustering hierarchies. We also assume that each internal node has at least two children; each child represents a different subclass/subcluster resulting from the split.

Let the set of leaf labels be $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$. We refer to a leaf node using its label in the following discussion. All nodes in the tree are uniquely numbered for easy reference (see Fig. 3). For a leaf node $\sigma_i \in \Sigma$, the internal nodes on the path from σ_i to the root are ancestors of σ_i. The ancestors represent the subclusters/subclasses containing σ_i. For any two leaf nodes σ_i, σ_j, we define the *Split* between σ_i, σ_j as follows:

Definition 1. *The* ***Split*** *between σ_i, σ_j (denoted as $Split(\sigma_i, \sigma_j)$). Given a rooted, unordered tree T with a set of leaf labels Σ, for any two leaf nodes $\sigma_i, \sigma_j \in \Sigma$, the Split between σ_i, σ_j is defined as the most recent common ancestor of σ_i and σ_j in T.*

$Split(\sigma_i, \sigma_j)$ represents the smallest cluster/class which includes both σ_i and σ_j, *i.e.*, the split during the top-down hierarchical clustering/classification process which divides σ_i, σ_j into different subclusters/classes. For example, in Fig. 3, nodes 1, 2, 4, and 9 are ancestors of leaf node a, nodes 1, 2, and 4 are ancestors of leaf node c. Therefore, $Split(a, c)$ is node 4.

In the following section, we define the Split-Order distance for any two hierarchies T_1, T_2 over the same set of leaf objects Σ.

4 Split-Order Distance

Given a rooted, unordered tree T with a set of leaf labels Σ, for any leaf node σ_i, the order of the splits which separate σ_i and the remaining leaf nodes determines the relationship between σ_i and other leaves. By "order", we mean that the split happens "earlier" (closer to the root) or "later" (closer to the leaf σ_i). Formally, for any leaf node σ_i, we define the order relationship of any two splits $Split(\sigma_i, \sigma_j)$,$Split(\sigma_i, \sigma_k)$ as follows:

Definition 2. *Split-Order Relations. Given a rooted, unordered tree with a set of leaf labels Σ, for any leaf node $\sigma_i \in \Sigma$, and any two other leaf nodes σ_j, $\sigma_k \in \Sigma$, we define*

1. *$SplitOrder(\sigma_i, \sigma_j, \sigma_k) =$'$\prec$', if $Split(\sigma_i, \sigma_j)$ is an ancestor of $Split(\sigma_i, \sigma_k)$, ($Split(\sigma_i, \sigma_j)$ happens earlier than $Split(\sigma_i, \sigma_k)$);*
2. *$SplitOrder(\sigma_i, \sigma_j, \sigma_k) =$'$\succ$', if $Split(\sigma_i, \sigma_k)$ is an ancestor of $Split(\sigma_i, \sigma_j)$, ($Split(\sigma_i, \sigma_j)$ happens later than $Split(\sigma_i, \sigma_k)$);*
3. *$SplitOrder(\sigma_i, \sigma_j, \sigma_k) =$'$=$', if $Split(\sigma_i, \sigma_j) = Split(\sigma_i, \sigma_k)$, ($Split(\sigma_i, \sigma_j)$ happens at the same time with $Split(\sigma_i, \sigma_k)$).*

For example, in Fig. 3, we have $SplitOrder(a, b, c) =$'$\succ$', $SplitOrder(a, g, d) =$'$\prec$', and $SplitOrder(a, f, h) =$'$=$'. The topology of T determines a map $\Sigma \times \Sigma \times \Sigma \mapsto \{\prec, \succ, =\}$. In the following discussion, we show that the complete set of the Split-Order relations $\Theta = \{SplitOrder(\sigma_i, \sigma_j, \sigma_k) = \theta_{i,j,k}\}, \forall \sigma_i, \sigma_j, \sigma_k \in \Sigma$, $\theta_{i,j,k} \in \{\prec, \succ, =\}$ can serve as a unique signature of T.

Theorem 1. *Given a complete set of the Split-Order relations $\Theta = \{SplitOrder(\sigma_i, \sigma_j, \sigma_k) = \theta_{i,j,k}\}, \forall \sigma_i, \sigma_j, \sigma_k \in \Sigma$, $\theta_{i,j,k} \in \{\prec, \succ, =\}$, we can either reconstruct a unique tree or declare that a tree corresponding to Θ does not exist.*

Proof. We prove the theorem by describing the following recursive algorithm to reconstruct a unique tree or report the nonexistence of a tree.

In the beginning, we start with the set of leaf nodes Σ. All the nodes are unprocessed initially. Each time we pick an unprocessed leave node from Σ for processing until all nodes in Σ are processed. Let σ_i be the node we pick. We find the set of leaf nodes $\{\sigma_j\}$ which have the latest split with σ_i:

$$\{\sigma_j | \neg\exists \sigma_k \in \Sigma, \sigma_k \neq \sigma_i, SplitOrder(\sigma_i, \sigma_j, \sigma_k) = ' \prec ' \} \tag{1}$$

It is intuitive to prove that σ_i and all nodes in $\{\sigma_j\}$ must be siblings. Next, we examine whether there are any conflicting Split-Order relations in Θ. For any two leaf nodes $\sigma_{j_1}, \sigma_{j_2} \in \{\sigma_j\}$, we check whether the following two equations hold:

$$\begin{aligned} &SplitOrder(\sigma_i, \sigma_{j_1}, \sigma_k) = SplitOrder(\sigma_i, \sigma_{j_2}, \sigma_k) \\ &\forall \sigma_k \in \Sigma \end{aligned} \tag{2}$$

$$\begin{aligned} &SplitOrder(\sigma_{j_1}, \sigma_k, \sigma_l) = SplitOrder(\sigma_{j_2}, \sigma_k, \sigma_l) = SplitOrder(\sigma_i, \sigma_k, \sigma_l) \\ &\forall \sigma_k, \sigma_l \in \Sigma \end{aligned} \tag{3}$$

If any of the above equations does not hold, which implies conflicting Split-Order relations in Σ, a tree conforming to Σ does not exist. Due to limited space, we omit the proof here. If both equations hold, we create a parent node σ_{new} for σ_i and all leaf nodes in $\{\sigma_j\}$. We add σ_{new} to $\Sigma^{(1)}$, and mark $\{\sigma_j\}$ and σ_i as processed.

If Σ still contains unprocessed nodes, we pick another leaf node, and start the process again. After all nodes in Σ have been processed, we start the next iteration with $\Sigma^{(1)}$. All nodes in $\Sigma^{(1)}$ will be treated as leaf nodes, and their Split-Order relations are determined by the nodes they represent, as follows:

$$SplitOrder(\sigma^{(1)}_{u'}, \sigma^{(1)}_{v'}, \sigma^{(1)}_{w'}) = SplitOrder(\sigma_u, \sigma_v, \sigma_w) \tag{4}$$

where σ_u is any child of $\sigma^{(1)}_{u'}$, σ_v is any child of $\sigma^{(1)}_{v'}$, and σ_w is any child of $\sigma^{(1)}_{w'}$.

We recurse until at iteration p, $\Sigma^{(p)}$ contains only the root node. If at any point, either Equation 2 or 3 does not hold, the process aborts and reports the nonexistence of a tree. The complete algorithm is shown in Algorithm 1.

Definition 3. *Split-Order distance. For two rooted, unordered trees T, T' with the same set of leaf labels Σ, and their corresponding Split-Order relationship sets Θ, Θ', the Split-Order distance between T and T' is defined as*

$$\begin{aligned} &SODist(T, T') = \\ &|\{SplitOrder(\sigma_i, \sigma_j, \sigma_k) \text{ s.t. } SplitOrder(\sigma_i, \sigma_j, \sigma_k) \neq SplitOrder(\sigma'_i, \sigma'_j, \sigma'_k)\}| \end{aligned} \tag{5}$$

Algorithm 1 ReconstructTree(Σ,Θ)
Input Σ: the leaf label set; Θ: the set of Split-Order relations over Σ, $\Theta = \{SplitOrder(\sigma_i, \sigma_j, \sigma_k) = \theta_{i,j,k}\}, \forall \sigma_i, \sigma_j, \sigma_k \in \Sigma$, $\theta_{i,j,k} \in \{\prec, \succ, =\}$

```
1: $\Sigma_{curr} \leftarrow \Sigma$
2: while do
3:   if $|\Sigma_{curr}| = 1$ then
4:     The only node in $\Sigma_{curr}$ is the root, return;
5:   else
6:     while $\Sigma_{curr}$ contains unprocessed nodes do
7:       Pick an unprocessed node $\sigma_i \in \Sigma_{curr}$.
8:       Find the set of nodes $\{\sigma_j\}$ which have the latest split with $\sigma_i$, according to
         Equation 1.
9:       Test conflict, according to Equations 2 and 3.
10:      if there is a conflict then
11:        Tree does not exist, return
12:      else
13:        Create a new node $\sigma_{new}$ as the parent of $\sigma_i$ and all nodes in $\{\sigma_j\}$
14:        Add $\sigma_{new}$ to $\Sigma_{next}$
15:        Mark $\sigma_i$ and all nodes in $\{\sigma_j\}$ as processed
16:      end if
17:    end while
18:    $\Sigma_{curr} \leftarrow \Sigma_{next}$
19:  end if
20: end while
```

where $(SplitOrder(\sigma_i, \sigma_j, \sigma_k) = \theta_{i,j,k}) \in \Theta$, *and* $(SplitOrder(\sigma'_i, \sigma'_j, \sigma'_k) = \theta'_{i,j,k}) \in \Theta'$.

The relative Split-Order distance is defined as:

$$SODistRel(T, T') = SODist(T, T')/n^3 \tag{6}$$

where $n = |\Sigma|$. *In other words,* $n^3 = |\Theta| = |\Theta'|$. *Note that* $SODistRel(T, T') \in [0, 1]$.

Theorem 2. *The Split-Order distance is a metric.*

Proof. Assume that we have three rooted, unordered trees T_1, T_2, T_3 with the same set of leaf labels Σ. The proof for symmetry, identity and non-negativity properties are intuitive and omitted here. We will prove the triangle inequality: $SODist(T_1, T_2) \leq SODist(T_1, T_3) + SODist(T_2, T_3)$. Denote the total number of Split-Order relations for each tree as m. Then the number of common Split-Order relations shared between T_1 and T_3 is $m - SODist(T_1, T_3)$, and the number of common Split-Order relations shared between T_2 and T_3 is $m - SODist(T_2, T_3)$. Therefore, the number of common Split-Order relations shared between T_1, T_2, and T_3 is at least

$$\begin{aligned}&(m - SODist(T_1, T_3)) + (m - SODist(T_2, T_3)) - m\\ &= m - (SODist(T_1, T_3) + SODist(T_2, T_3))\end{aligned}$$

Thus, the number of Split-Order relations which are different between T_1 and T_2 is at most

$$\begin{aligned}&m - (m - (SODist(T_1, T_3) + SODist(T_2, T_3)))\\&= SODist(T_1, T_3) + SODist(T_2, T_3)\end{aligned}$$

5 Split-Order Distance Computation

We propose an efficient algorithm for computing the Split-Order distance between any two trees T_1, T_2 over the same set of leaf objects Σ.

The naive algorithm computes the complete set of the Split-Order relations for both trees and counts the number of different Split-Order relations in two trees. Since there are $O(n^3)$ Split-Order relations for each tree, counting the different Split-Order relations alone will take $O(n^3)$ time. In fact, computing a single Split-Order relation has more than $O(1)$ complexity. In the following discussion, we propose an efficient algorithm for computing Split-Order distance which takes only $O(n^2d^4)$ time, where d is the maximum degree of any node.

Let $\Theta(T)$ denote the complete set of the Split-Order relations of tree T. For each internal node n_p, we compute a subset of $\Theta(T)$, denoted as $SplitOrderSet(T, n_p)$:

$$\begin{aligned}&SplitOrderSet(T, n_p) = \{SplitOrder(\sigma_{i_1}, \sigma_{i_2}, \sigma_{i_3}) = \theta_{i_1,i_2,i_3}\ where\\&(Split(\sigma_{i_1}, \sigma_{i_2}) = n_p) \wedge ((SplitOrder(\sigma_{i_1}, \sigma_{i_2}, \sigma_{i_3}) = \theta_{i_1,i_2,i_3}) \in \Theta(T))\}\end{aligned} \tag{7}$$

It is easy to prove that $\Theta(T)$ can be divided into disjoint sets of $SplitOrderSet(T, n_p)$:

$$\cup_{n_p} SplitOrderSet(T, n_p) = \Theta(T) \tag{8}$$

$$SplitOrderSet(T, n_{p_1}) \cap SplitOrderSet(T, n_{p_2}) = \phi \tag{9}$$

The basic idea of our Split-Order distance computation algorithm is to count, for any internal node n_p in T and any internal node $n_{p'}$ in T', the number of common Split-Order relations associated with them, *i.e.*, $|SplitOrderSet(T, n_p) \cap SplitOrderSet(T', n_{p'})|$. In the following discussion, we only consider the Split-Order relations of type '$\succ$' and '$=$', since we have

$$\begin{aligned}&SplitOrder(\sigma_{i_1}, \sigma_{i_2}, \sigma_{i_3}) = '\prec' \Leftrightarrow\\&SplitOrder(\sigma_{i_1}, \sigma_{i_3}, \sigma_{i_2}) = '\succ'\end{aligned} \tag{10}$$

The total number of common Split-Order relations is twice the number of total common '$\succ$'-type Split-Order relations plus the number of total common '$=$'-type Split-Order relations. Furthermore, we only consider Split-Order relations where σ_{i_1}, σ_{i_2}, and σ_{i_3} are three different leaf labels. If any two of them are the same we already know that the relation is common between $\Theta(T)$ and $\Theta(T')$.

We first explain how to compute the Split-Order relations of type '$\succ$' and '$=$' in $SplitOrderSet(T, n_p)$ given a tree T and an internal node n_p. Denote the set of leaf nodes which are inside the subtree rooted at n_p as $Leaves(n_p)$, and the k^{th} child of n_p as $Child(n_p, k)$ (see Fig. 4). It is easy to prove that $Leaves(n_p) =$

$\cup_k Leaves(Child(n_p,k))$, and $Leaves(Child(n_p,k)) \cap Leaves(Child(n_p,k')) = \phi$. Here we assume that all child nodes are in an ordered list for the convenience of discussion. The child nodes can be of any order. For example in Fig. 3, consider n_4, $Leaves(n_4) = \{a,b,c\}$. n_9 and n_{10} are the children of n_4, and $Leaves(n_9) = \{a,b\}$, $Leaves(n_{10}) = \{c\}$. Now we determine the necessary conditions for $SplitOrder(\sigma_{i_1},\sigma_{i_2},\sigma_{i_3})$ to be of '$\succ$' or '$=$' type. If $\sigma_{i_1} \in Leaves(Child(n_p,k_1))$, and $\sigma_{i_2} \in Leaves(Child(n_p,k_2))$, we know that $k_1 \neq k_2$. Otherwise, $Split(\sigma_{i_1},\sigma_{i_2})$ is $Child(n_p,k_1)$ or a descendent of $Child(n_p,k_1)$, but not n_p. Therefore, σ_{i_1},σ_{i_2} are from different child nodes of n_p. For the requirement of σ_{i_3}, we have $\sigma_{i_3} \in \Sigma\backslash Leaves(Child(n_p,k_1))$. Otherwise, if $\sigma_{i_3} \in Leaves(Child(n_p,k_1))$, we have $Split(\sigma_{i_1},\sigma_{i_2},\sigma_{i_3}) =$'$\prec$'. Furthermore, depending on the type of the Split-Order relationship ('$\succ$' or '$=$'), σ_{i_3} should belong to either $\Sigma\backslash Leaves(n_p)$ or $Leaves(n_p)\backslash Leaves(Child(n_p,k_1))$. In summary, σ_{i_1}, σ_{i_2}, σ_{i_3} in a '$\succ$'-type Split-Order relation in $SplitOrderSet(T,n_p)$ should satisfy (see Fig. 4):

$$\begin{cases} \sigma_{i_1} \in Leaves(Child(n_p,k_1)) \\ \sigma_{i_2} \in Leaves(Child(n_p,k_2)) \wedge k_1 \neq k_2 \\ \sigma_{i_3} \in \Sigma\backslash Leaves(n_p) \end{cases}$$

where $1 \leq k_1, k_2 \leq K$, and K is the number of children of node n_p. σ_{i_1}, σ_{i_2}, σ_{i_3} in an '$=$'-type *SplitOrder* relation in $SplitOrderSet(T,n_p)$ should satisfy (see Fig. 4):

$$\begin{cases} \sigma_{i_1} \in Leaves(Child(n_p,k_1)) \\ \sigma_{i_2} \in Leaves(Child(n_p,k_2)) \wedge k_1 \neq k_2 \\ \sigma_{i_3} \in Leaves(Child(n_p,k_3)) \wedge k_1 \neq k_3 \wedge \sigma_{i_2} \neq \sigma_{i_3} \end{cases}$$

where $1 \leq k_1, k_2 \leq K$, and K is the number of children of n_p. As an example, for node n_7 in Fig. 3, we have $SplitOrder(f,h,i) =$'$=$' and $SplitOrder(f,h,a) =$'$\succ$'.

Now we explain how to compute the size of $SplitOrderSet(T,n_p) \cap SplitOrderSet(T',n_{p'})$ for a pair of internal nodes n_p, $n_{p'}$ in tree T and T', respectively. As discussed earlier, we consider the '$\succ$' and '$=$' Split-Order relations. The number of common '$\succ$' Split-Order relations $Shared_{\succ}(n_p,n_{p'})$ can be computed as follows:

$$\begin{aligned} &Shared_{\succ}(n_p,n_{p'}) = \Sigma_{1\leq k_1\neq k_2\leq K, 1\leq k_1'\neq k_2'\leq K'} \\ &(|Leaves(Child(n_p,k_1)) \cap Leaves(Child(n_{p'},k_1'))| \\ &\times|Leaves(Child(n_p,k_2)) \cap Leaves(Child(n_{p'},k_2'))| \\ &\times|(\Sigma\backslash Leaves(n_p)) \cap (\Sigma\backslash Leaves(n_{p'}))|) \end{aligned} \tag{11}$$

In both trees, a '$\succ$'-type Split-Order relation satisfies the following conditions: σ_{i_1}, σ_{i_2} are descendants of two different children of n_p and also of two different children of $n_{p'}$. In addition, σ_{i_3} is not in either $Leaves(n_p)$ or $Leaves(n_{p'})$ (see Fig. 4). Assume that computing the size of a set intersection takes constant time (we will explain later how to compute it in constant time). Let the maximum number of children for a node be d. The computation of $Shared_{\succ}(n_p,n_{p'})$ takes $O(d^4)$ time.

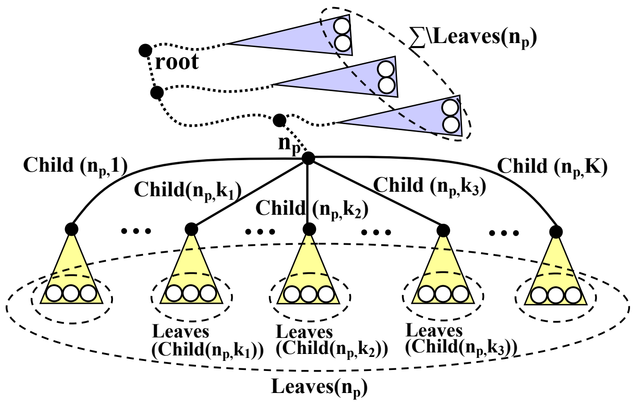


Fig. 4. Illustration of Split-Order distance computation.

The number of common '='-type Split-Order relations $Shared_{=}(n_p, n_{p'})$ can be computed as follows:

$$\begin{aligned}&Shared_{=}(n_p, n_{p'}) = \Sigma_{1 \le k_1 \le K, 1 \le k'_1 \le K'} \\ &(|Leaves(Child(n_p, k_1)) \cap Leaves(Child(n_{p'}, k'_1))| \\ &\times (\Sigma_{1 \le k_2 \le K, 1 \le k'_2 \le K', k_2 \ne k_1, k'_2 \ne k'_1} \\ &|Leaves(Child(n_p, k_1)) \cap Leaves(Child(n_{p'}, k'_2))|) \\ &\times ((\Sigma_{1 \le k_2 \le K, 1 \le k'_2 \le K', k_2 \ne k_1, k'_2 \ne k'_1} \\ &|Leaves(Child(n_p, k_1)) \cap Leaves(Child(n_{p'}, k'_2))|) - 1))\end{aligned} \tag{12}$$

Similarly, in either tree, an '='-type Split-Order relation satisfies the following conditions: σ_{i_1}, σ_{i_2}, σ_{i_3} are all descendants of n_p and children of n'_p; also, σ_{i_2} and σ_{i_3} must be descended from children which are not ancestors of σ_{i_1}; finally, σ_{i_2} are σ_{i_3} are different leaf labels (see Fig. 4). Again, if a set intersection computation takes constant time, the computation of $Shared_{=}(n_p, n_{p'})$ takes $O(d^4)$ time.

Therefore, let the total number of common '$\succ$'-type Split-Order relations shared between two trees be $Shared_{\succ}(T, T')$, and the total number of common '='-type Split-Order relations shared between two trees be $Shared_{=}(T, T')$. Then we have:

$$Shared_{\succ}(T, T') = \Sigma_{\forall n_p \in T, n_{p'} \in T'} Shared_{\succ}(n_p, n_{p'}) \tag{13}$$

$$Shared_{=}(T, T') = \Sigma_{\forall n_p \in T, n_{p'} \in T'} Shared_{=}(n_p, n_{p'}) \tag{14}$$

Thus, the total number of common Split-Order relations shared between $SplitOrderSet(T, n_p)$ and $SplitOrderSet(T', n_{p'})$ is:

$$Shared(T, T') = Shared_{=}(T, T') + 2 \times Shared_{\succ}(T, T') \tag{15}$$

Therefore, the Split-Order distance between T and T' is:

$$SODist(T, T') = n^3 - (n + 3n(n-1)) - Shared(T, T') \tag{16}$$

Here n^3 is the total number of Split-Order relations for a leaf label set of size n, and $n + 3n(n-1)$ is the number of Split-Order relations of which at least two of σ_{i_1},σ_{i_2},σ_{i_3} have the same label. As mentioned before, these Split-Order relations must be common for both trees.

Now we explain how we can compute the size of the set intersection in constant time. The basic idea is that we compute in advance $|Leaves(n_p) \cap Leaves(n_{p'})|$ for all possible pairs of n_p and $n_{p'}$, which are the internal nodes of T and T', respectively. This can be done in $O(n^2)$ time as follows:

1. Initially, we compute $|Leaves(n_p) \cap Leaves(n_{p'})|$ where n_p and $n_{p'}$ are leaf nodes in T and T', respectively. Each $|Leaves(n_p) \cap Leaves(n_{p'})|$ can be done in constant time.
2. We compute $|Leaves(n_p) \cap Leaves(n_{p'})|$ if either of the following cases holds:
   (a) $|Leaves(Child(n_p, k)) \cap Leaves(n_{p'})|$ has been computed for all children of n_p
   (b) $|Leaves(n_p) \cap Leaves(Child(n_{p'}, k'))|$ has been computed for all children of $n_{p'}$

   This can be done in $O(d)$ time where d is the maximum degree of any node. For case (a):
$$\begin{aligned}&|Leaves(Child(n_p, k)) \cap Leaves(n_{p'})| = \\ &\Sigma_k |Leaves(Child(n_p, k)) \cap Leaves(n_{p'})|\end{aligned} \tag{17}$$
   For case (b):
$$\begin{aligned}&|Leaves(Child(n_p, k)) \cap Leaves(n_{p'})| = \\ &\Sigma_{k'} |Leaves(n_p) \cap Leaves(Child(n_{p'}, k'))|\end{aligned} \tag{18}$$
3. Repeat 2) until $|Leaves(n_p) \cap Leaves(n_{p'})|$ is computed for any pair of nodes $n_p \in T$, and $n_{p'} \in T'$

Therefore, computing $Leaves(n_p) \cap Leaves(n_{p'})$ for all possible pairs of n_p and $n_{p'}$ takes $O(dn^2)$ time. Note that in the computation of $Shared_{\succ}(n_p, n_{p'})$, we also need to compute $|(\Sigma \backslash Leaves(n_p)) \cap (\Sigma \backslash Leaves(n_{p'}))|$, which can be computed in constant time as follows:
$$\begin{aligned}&|(\Sigma \backslash Leaves(n_p)) \cap (\Sigma \backslash Leaves(n_{p'}))| = \\ &n - |Leaves(n_p)| - |Leaves(n_{p'})| + \\ &|Leaves(n_p) \cap Leaves(n_{p'})|\end{aligned} \tag{19}$$
We can easily compute $|Leaves(n_p)|$ and $|Leaves(n_{p'})|$ in advance for all $n_p \in T$ and $n_{p'} \in T'$ using a post-order traversal for both trees, which takes $O(nd)$ for each tree.

The complete algorithm for computing Split-Order distance is in Algorithm 2.

Theorem 3. *Algorithm 2 runs in $O(n^2d^4)$ time.*

Proof. To compute $SODist(T, T')$, we consider every pair of nodes $(n_p, n_{p'})$. The number of internal nodes for both trees are $O(n)$, therefore, we have $O(n^2)$ node pairs. Since computing $Shared(n_p, n_{p'})$ takes $O(d^4)$ time, the total time complexity of Algorithm 2 is $O(n^2d^4)$.

Algorithm 2 SODist(T,T',Σ)
Input Σ: the leaf label set; T, T': two trees

```
1: Compute all possible set intersections: $|Leaves(n_p) \cap Leaves(n_{p'})|$, for any internal
   node $n_p \in T$ and any internal node $n_{p'} \in T'$
2: Compute all $|Leaves(n_p)|$, $|Leaves(n_{p'})|$, for any internal node $n_p \in T$ and any
   internal node $n_{p'} \in T'$
3: for any internal node $n_p \in T$ and any internal node $n_{p'} \in T'$ do
4:    Compute $Shared_{\succ}(n_p, n_{p'})$ and $Shared_{=}(n_p, n_{p'})$ according to Equation 10 and
      11
5: end for
6: Compute $SODist(T, T')$ according to Equation 12,13,14 and 15.
```

6 Experimental Results

We test the performance of our algorithm on both real and synthetic data sets.

- **Iyer's Data[Iyer96]:** Iyer's Data contains gene expression levels of 517 human genes in response to serum stimulation over 12 time points. The 517 genes can be clustered into hierarchical structures based on their co-expressions [Iyer96,Jiang03].
- **Synthetic Data:** 1) **SYN-Random:** The synthetic dataset contains a set of randomly generated, rooted, leaf labeled, unordered trees. The simulation is controlled by two parameters: the number of leaves n, and the maximum degree d of any internal node. 2) **SYN-2D:** The 2-D synthetic dataset contains 1000 sampled pixels from 4 shapes in an image, including 1% of background noises. Clustering hierarchy for Syn-2D is obtained by applying CLUTO software[1]. The 4 clusters (representing the four big branches in the hierarchy) are illustrated using different colors in Fig. 6(a).

We compared the performance of our algorithm against the tree edit distance. We used a Java implementation of Zhang-Shasha's tree edit distance algorithm [Zhang89] available online[2]. Zhang-Shasha's algorithm applies to ordered, labeled trees. Our Split-Order algorithm is implemented in both C++ and Java, and the experiments are performed on an Intel Core 2 Duo 1.6GHz machine with 3GB memory. The Java version of the Split-Order algorithm is mainly used for the comparison of running time performance with the Java implementation of Zhang-Shasha's tree edit distance algorithm.

6.1 Distance Evaluation

Distance Comparison on Iyer's Data Iyer's Data contains gene expression levels of 517 human genes over 12 time points. We refer to the complete Iyer's Data as Iyer12. Six additional data sets are generated from Iyer12 by randomly

[1] http://glaros.dtc.umn.edu/gkhome/views/cluto
[2] http://www.ics.mq.edu.au/ swan/howtos/treedistance/package.html

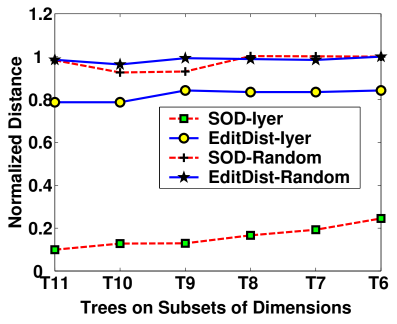


Fig. 5. Comparison of Split-Order distance and tree edit distance (Zhang-Shasha) on Iyer's data

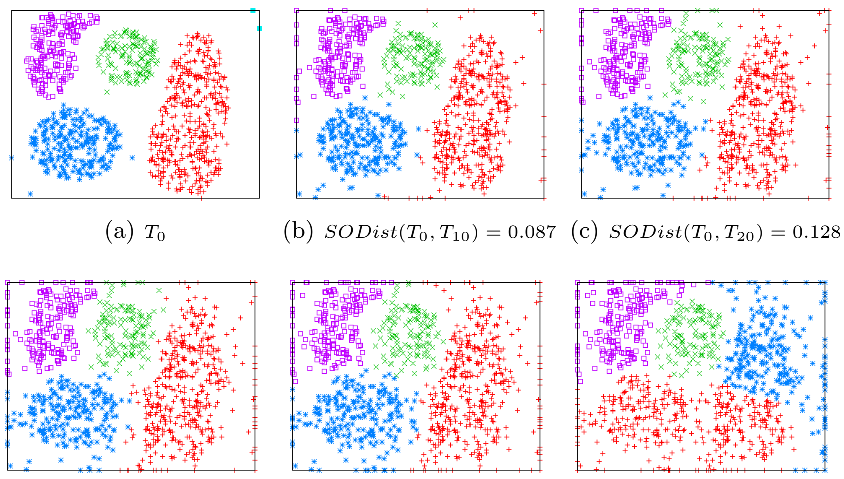
choosing 11, 10, 9, 8, 7, 6 dimensions respectively. These data sets are referred to as Iyer11, Iyer10, Iyer9, Iyer8, Iyer7, and Iyer6. We apply the same hierarchical clustering algorithm on these 7 data sets with Euclidean distance and single linkage, and obtain 7 trees: T_{12}, T_{11}, T_{10}, T_9, T_8, T_7, and T_6. As we remove more and more dimensions, the correlation captured by the resulting tree differs more and more from the original tree T_{12}. We would therefore expect that the distances of T_{11}, T_{10}, T_9, T_8, T_7, and T_6 to T_{12} are in an increasing order.

Fig.5 compares the distance scores computed using Split-Order distance and tree edit distance on the Iyer's Data. The Split-Order scores are normalized with the Split-Order distance of a random tree to T_{12} which is 0.664. Similarly, the tree edit distance scores are normalized with the tree edit distance between a random tree and T_{12} which is 1433. The Split-Order distance curve demonstrates a clear increasing trend for distances of T_{11}, T_{10}, T_9, T_8, T_7, and T_6 to the original tree T_{12}, which is consistent with our expectation. Compared to tree edit distance, Split-Order distance generates a much smaller distance for similar trees. The Split-Order distance between T_{11} and T_{12} is 10% of the distance between a random tree and T_{12}. The tree edit distance between T_{11} and T_{12} is however 80% of the distance between a random tree and T_{12}.

We also generated a random matrix of the same size (517 by 12) and performed the same experiment on this data. The corresponding curves are plotted in Fig.5. The two curves on random data are flat. The different behaviors of the random data curves and the Iyer's data curves can be explained by the gene correlation existing in the Iyer's data and the captured tree similarity due to the data correlation.

The average running time of Split-Order distance and tree edit distance in this experiment is 0.374 sec and 120.2 sec respectively.

Distance Evaluation on Synthetic Data We performed the distance evaluation on clustering hierarchies generated using SYN-2D. SYN-2D contains 1000 pixels uniformly sampled from 4 different shapes in an image, including 1% of background noise. We generated 5 additional data sets by randomly choosing



(a) T_0 (b) $SODist(T_0, T_{10}) = 0.087$ (c) $SODist(T_0, T_{20}) = 0.128$

(d) $SODist(T_0, T_{30}) = 0.176$ (e) $SODist(T_0, T_{40}) = 0.217$ (f) $SODist(T_0, T_{50}) = 0.458$

Fig. 6. Illustration of 4 big clusters in the clustering hierarchies computed for SYN-2D and its perturbed data sets. T_0 is the clustering hierarchy computed on SYN-2D, T_{10}, T_{20}, T_{30}, T_{40}, T_{50} are the clustering hierarchies computed on SYN-2D with 10%, 20%, 30%, 40%, 50% of points applied perturbation.

10%, 20%, 30%, 40%, and 50% of pixels in SYN-2D and applyed a random perturbation on each pixel. Clustering hierarchies are generated on all 6 data sets using the hierarchical clustering function equipped with CLUTO [Karypis02]. We refer to the corresponding trees as T_0, T_{10}, T_{20}, T_{30}, T_{40}, and T_{50}. The four big clusters (the four big branches) in each tree are plotted in Fig.6(a)-Fig.6(f).

The Split-Order distances of each of T_0 to T_{10}, T_{20}, T_{30}, T_{40}, and T_{50} are 0.087, 0.128, 0.176, 0.216, and 0.458, respectively. The visual differences among Fig.6(a)-Fig.6(f) reflect the clustering hierarchy differences among the corresponding trees. The increasing order of the Split-Order differences are accordant with the increasing visual differences shown in Fig.6(b) to Fig.6(f). Particularly, from T_{50} to T_{40}, the Split-Order distance increases dramatically by more than 100%. Comparing Fig. 6(e) and Fig. 6(f), we also observe a big change: the blue cluster in 6(e) becomes part of the red cluster in 6(f); part of the red cluster in 6(e) becomes blue cluster in 6(f). This implies that the big branch representing the blue cluster in T_{40} switches positions with a subbranch of the red cluster. This change impacts the relationship between a large number of pixels, which results in a big difference in the corresponding Split-Order distances.

6.2 Running Time Performance

We compare the running time performance of our algorithm (the Java version) with tree edit distance (Java implementation of Zhang-Shasha algorithm) on the SYN-Random dataset.

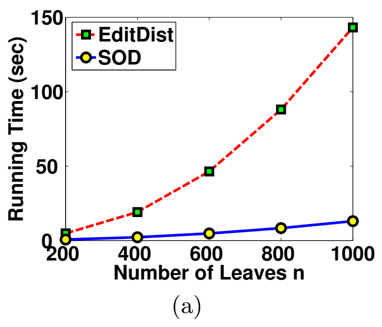


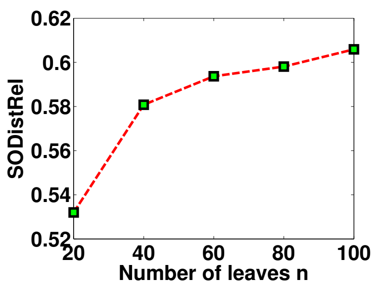
Fig. 7. Running time comparison of Split-Order distance (the Java implementation) and tree edit distance (Java implementation of Zhang-Shasha) with varying number of leaf nodes n ((a)) and with varying max degree d ((b)).

Running time vs. *n*: Fig. 7(a) compares the running time of Split-Order and Zhang-Shasha (both implemented in Java) on trees with varying number of leaves n. As n increases from 200 to 1000, the running time for both algorithms increases. The Split-Order distance is up to 10 times faster than the Zhang-Shasha algorithm and demonstrates better scalability with increasing n. Each data point is the average time taken for 10 distance computations for a given n. Although the running time of Zhang-Shasha is acceptable for reasonably large trees (less than 150 seconds for trees with 1000 leaves), the speedup would still be necessary considering large number of tree edit distance computations required for applications such as querying phylogeny tree databases.

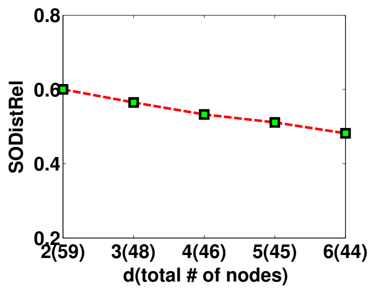
Running time vs. *d*: Fig. 7(b) compares the running time of Split-Order and Zhang-Shasha (both implemented in Java) on trees with varying maximum degree d. The tree edit distance demonstrates a clear decline with increasing d, due to reduced total number of nodes with increasing d and fixed number of leaves n. Split-Order remains almost flat with small variations. The time complexity of our algorithm depends on the number of internal nodes (which is $O(n)$) and the maximum degree of any node d. The running time would increase with larger d. However, with fixed n and increasing d, the number of internal nodes decreases, which cancels out the increase introduced by larger d. The data is averaged over 10 runs of the algorithms.

6.3 *SODist* Distribution

We examined the distribution of *SODist* between any two trees using SYN-Random. 100 pairs of random trees are chosen and the relative Split-Order distances (*SODistRel*) are computed for each pair. Fig. 8(a) and 8(b) illustrate the distribution of the average *SODistRel* with varying parameters. We observe that the average *SODistRel* between any two random trees roughly falls



(a) *Average SODistRel vs. n*



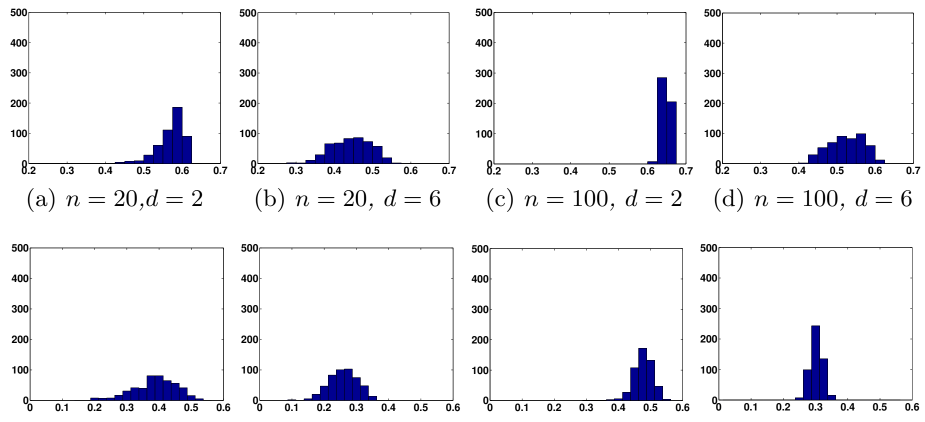
(b) *Average SODistRel vs. d*

Fig. 8. The average *SODistRel* between random trees with varying parameters: n (the number of leaf nodes) and d (the maximum number of children an internal node could have in a tree). d is fixed to be 3 in (a) and n is fixed to be 30 in (b).

within the range [0.5, 0.6]. There are small variations when parameters change. As shown in Fig. 8(a), *SODistRel* ranges between 0.53 and 0.61 as n varies from 20 to 100 and d is fixed to be 3. The increase in the average *SODistRel* between random trees is due to the increased topological variety of trees when n is larger. In Fig. 8(b), the average *SODistRel* between random trees is computed as d varies from 2 to 6, and n is fixed to be 30. We can observe a slow drop in the average *SODistRel* between random trees when d increases. This is due to the decrease in the number of internal nodes with increasing d and fixed n. As tree becomes flatter, the amount of topological variety of trees also decreases.

Fig. 9(a)–9(d) plots the histogram of 500 *SODistRel*s between 500 pairs of random trees with different parameters settings: $n = 20$ and $d = 2$ (Fig. 9(a)), $n = 20$ and $d = 6$ (Fig. 9(b)), $n = 100$ and $d = 2$ (Fig. 9(c)), $n = 100$ and $d = 6$ (Fig. 9(d)). We observe that all distributions have the bell shape. For larger d ($d = 6$ compared to $d = 2$), more diversity exists in the tree space and histogram is more symmetric. For a larger n ($n = 100$ compared to $n = 20$), *SODistRel* becomes larger, and the mean of the distribution moves further to the right. These histograms depict the variation of the tree space when parameters change evaluated by the Split-Order distance.

For comparison, we plot the four corresponding histograms of tree edit distances computed over the same 500 pairs of random trees in Figs. 9(e), 9(f),9(g) and 9(h). The edit distance scores are normalized by $4n$ to get the relative scores. $4n$ is the upper bound of the tree edit distance between two trees with n leaves. We observe that when d increases (from 2 to 6), the variance of the tree edit distance declines. This contradicts the fact that the topological diversity increases as d increases. This suggests that the tree edit distance is not an ideal measure.


(a) $n = 20, d = 2$ (b) $n = 20,\ d = 6$ (c) $n = 100,\ d = 2$ (d) $n = 100,\ d = 6$

(e) $n = 20, d = 2$ (f) $n = 20,\ d = 6$ (g) $n = 100,\ d = 2$ (h) $n = 100,\ d = 6$

Fig. 9. The distribution of *SODistRel* and tree edit distance between random trees in different settings. A total number of 500 Split-Order distance computations are performed. X axis is *SODistRel* or tree edit distance, Y axis is the number of the distance computations contained in each bin.

7 Conclusion and Future Work

In this paper, we propose a novel metric, Split-Order distance, to evaluate the distance between two clustering/classification hierarchies. This metric compares the order of the splits in both trees. We proved that a complete set of the split order relations uniquely determines the hierarchical structure. We also proposed an algorithm for reconstructing the tree using the set of order relations. Furthermore, we presented an efficient algorithm for computing the Split-Order distance between two hierarchies which takes $O(n^2d^4)$ time, where n is the number of leaf nodes (objects), and d is the maximum number of children in a tree.

In the future, we would like to take weighted edges into consideration where weights represent the distance from a node to its parent. This allows us to model the situations where a single edge with different weights can result in different relationships among objects, such as in a dendrogram.

References

[Allen01] Allen, B.L., Steel, M.: Subtree transfer operations and their induced metrics on evolutionary trees. Annals of Combinatorics, 5:1-13 (2001)

[Amir97] Amir, A., Keselman, D.: Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithms. Proc. of the SIAM Journal on Computing, 26(6):1656-1669 (1997)

[Bille05] Bille, P.: A survey on tree edit distance and related problems. Theoretical Computer Science, 217-239 (2005)

[Demaine07] Demaine, E.D., Mozes, S., Rossman, B., Weimann, O.: An Optimal Decomposition Algorithm for Tree Edit Distance. Proc. of the 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)

[Estabrook85] Eastabrook, G.F., McMorris, F.R., Meacham, C.A.: Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units. Syst. Zool. (1985)

[Felsenstein78] Felsenstein, J.: Cases in which parsimony and compatibility methods will be positively misleading. Syst. Zool., 27:401-410 (1978)

[Iyer96] Iyer V.R. *et al.*: Transcriptional program in the response of human fibroblasts to serum. Science, 283:83-87 (1996)

[Jiang03] Jiang, D., Pei, J., Zhang, A.: DHC: a density-based hierarchical clustering method for time series gene expression data. Proc. of the The Third Symposium on Bioinformatics and Bioengineering, 393-400 (2003)

[Karypis02] Karypis, G.: CLUTO - A Clustering Toolkit. Tech Report, Dept. of Computer Science, University of Minnesota, 2002

[Klein98] Klein, P.N.: Computing the edit-distance between unrooted ordered trees. Proc. of the 6th annual European Symposium on Algorithms (ESA) 1998

[Lu79] Lu, S.Y.: A tree-to-tree distance and its application to cluster analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 1979

[Robinson79] Robinson, D.F., Foulds, L.R.: Comparison of weighted labeled trees. Combinatorial mathematics, VI 119-126 (1979)

[Robinson81] Robinson, D.F., Foulds, L.: Comparison of phylogenetic trees. Math. Biosci., 53(1-2):131-147 (1981)

[Saitou87] Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. The Mol. Biol. Evol., 4(4):406425 (1987)

[Shasha04] Shasha, D., Wang, J.T.L., Zhang, S.: Unordered Tree Mining with Applications to Phylogeny. Proc. IEEE International Conference on Data Engineering (ICDE'04)

[Sneath73] Sneath, P.H.A., Sokal, R.R.: Numerical Taxonomy. WH Freeman and Company, 230-234 (1973)

[Touzet03] Touzet, H.: Tree edit distance with gaps. Information Processing Letters, 85(3): 123-129 (2003)

[Wang94] Wang, J.T., Zhang, K., Jeong, K., Shasha, D.: A system for approximate tree matching. IEEE Transactions on Knowledge and Data Engineering, 6(4): 559-571 (1994)

[Wang03] Eastabrook, G.F., McMorris, F.R., Meacham, C.A.: TreeRank: A similarity measure for nearest neighbor searching in phylogenetic databases. Proc. of the 15th International Conference on Scientific and Statistical Database Management (1985)

[Waterman78] Waterman, M.S., Smith, T.F.: On the similarity of dendrograms. Journal of Theoretical Biology, 73:789-800 (1978)

[Zhang89] Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal of Computing (1989)