A New Dissimilarity Measure between Trees by Decomposition of Unit-Cost Edit Distance

Hisashi Koga, Hiroaki Saito, Toshinori Watanabe, and Takanori Yokoyama

Graduate Schools of Information Systems, University of Electro-Communications, Tokyo 182-8585, Japan

Abstract. Tree edit distance is a conventional dissimilarity measure between labeled trees. However, tree edit distance including unit-cost edit distance contains the similarity of label and that of tree structure simultaneously. Therefore, even if the label similarity between two trees that share many nodes with the same label is high, the high label similarity is hard to be recognized from their tree edit distance when their tree sizes or shapes are quite different. To overcome this flaw, we propose a novel method that obtains a label dissimilarity measure and a structural dissimilarity measure separately by decomposing unit-cost edit distance.

1 Introduction

Tree is useful for expressing various objects such as semi-structured data and genes [1]. For this reason, it is essential to compute tree similarity in the field of pattern recognition and information retrieval.

In this paper we focus on labeled ordered trees with the root. Let T be a rooted tree. T is called a *labeled tree* if each node is a assigned a symbol from a finite alphabet Σ. T is *ordered* if a left to right order among siblings in T is given. Tree edit distance [2] is one of the most common dissimilarity measures between two trees and defined as the minimum cost necessary to convert from one tree to another tree by repeating node edit operations (i.e, deletion, insertion and relabeling). Tree edit distance is easily implemented with dynamic programming for labeled ordered trees [3]. To compute a tree edit distance, users need to supply a cost function defined on each edit operation. Because it is difficult to tailor node edit costs for a specific application, *unit-cost edit distance* [4] in which all of node edit operations cost 1 equally is used frequently.

Tree edit distance including unit-cost edit distance mixes the similarity of node labels and that of tree structure, because not only label of nodes but also tree shape are matched in turning a tree T_1 to another tree T_2. Thus, even if T_1 and T_2 share many nodes with the same label, the high label similarity is hard to be recognized from their tree edit distance, if their tree sizes or shapes are quite different. To overcome this flaw, this paper newly proposes to decompose unit-cost edit distance into node edit operations to match node labels and into those to match tree structure and, then, to obtain a label dissimilarity measure from the former and a structural dissimilarity measure from the latter. Since both measures take a value between 0 and 1, users can easily understand the extent of dissimilarity. Furthermore, our label dissimilarity measure generalizes tree inclusion [2] which is the problem to decide if a tree T_1 includes another tree T_2. That is, it can measure the extent of the tree inclusion, even if T_2 is not completely contained in T_1. Since our dissimilarity measures are obtained from unit-cost edit distance, users may use unit-cost edit distance without additional overhead like development of a new program, in case they are not satisfied with our dissimilarity measures, which is a large advantage of our approach.

By applying our method to the noisy subsequence tree recognition problem [5] and to the classification of XML documents, we show that our method yields a better performance than the unit-cost edit distance.

The structure of this paper is as follows. Sect. 2 introduces the unit-cost edit distance and the tree inclusion as preliminaries. Sect. 3 exemplifies the flaw of the unit-cost edit distance. Sect. 4 presents our dissimilarity measures. Sections 5 and 6 report the experimental results. Sect. 7 is the conclusion.

2 Preliminaries

2.1 Unit-Cost Edit Distance

Here we define unit-cost edit distance between trees. Let T_1 and T_2 be labeled ordered trees with the root. A node with a label x is denoted by "node x". T_1 can be converted to T_2 by repeating deletion, insertion and relabeling of nodes.

All of the insertion, deletion and relabeling of nodes are named *node edit operations*. These operations are defined formally as follows.
insertion: Let $u_1, u_2, \ldots, u_l$ be the children of node y that are ordered, where l is the number of children nodes for y. Inserting a node x as the child of y between u_i and u_j $(1 \leq i < j \leq l)$ means that x becomes a child of y and the parent of nodes from u_{i+1} to u_{j-1}.
deletion: Deleting a node x means that the children of x become the children of the parent of x and then x is removed.
relabeling: Relabeling a node x to a node y means that the label of the node is modified from x to y. It has no influence on the tree shape.

By introducing the notation of a null node λ, all of these operations can be consistently described in the form of a node pair (x, y), where (x, y) indicates that node x is changed to node y. A relabeling operation corresponds to the case when $x \neq \lambda$ and $y \neq \lambda$. If $x = \lambda$ and $y \neq \lambda$, (x, y) becomes an insertion operation. If $x \neq \lambda$ and $y = \lambda$, (x, y) grows a deletion operation.

When T_1 is converted to T_2, we denote this conversion by $T_1 \rightarrow T_2$. The tree conversion is determined uniquely by the set of the performed node edit operations. This set is expresses as $M(T_1, T_2)$ and M is called a *tree mapping*. Fig. 1 illustrates the tree mapping. Let $T[i]$ be the i-th node in T. A dotted line from node $T_1[i]$ to node $T_2[j]$ indicates that $T_1[i]$ is relabeled to $T_2[j]$. The nodes in T_1 not touched by a dotted line are deleted and those in T_2 are inserted.

A node edit operation (x, y) is associated with its cost $c(x, y)$. For computing unit-cost edit distance, we assume that $c(x, y) = 1$ for any x, y satisfying $x \neq y$

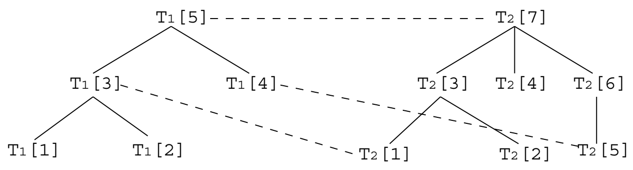

Fig. 1. A tree mapping

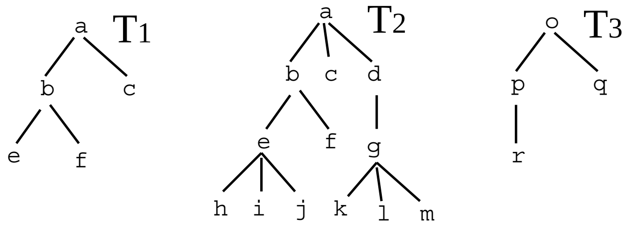

Fig. 2. 3 trees

and that $c(x,x) = 0$ for any x. This means that any insertion, deletion and relabeling to a different label costs 1 evenly. Relabeling to a different label is referred to as *non-free relabeling*.

Let D_M (I_M) be the set of nodes deleted from T_1 (respectively inserted to T_2) in M. Let S_M be the set of relabeled node pairs in M. Then, the cost of M is defined as (1) that is the total cost incurred in deletion, insertion and relabeling.

$$\mathrm{cost}(M) = \sum_{(v,w)\in S_M} c(v,w) + \sum_{v\in D_M} c(v,\lambda) + \sum_{w\in I_M} c(\lambda,w). \tag{1}$$

$D(T_1,T_2) = \min_M\{\mathrm{cost}(M)\}$ where the minimum is taken over M is called the unit-cost edit distance between T_1 and T_2.

2.2 Tree Inclusion

When T_1 can be converted to T_2 only by node delete operations, T_2 is said to be included in T_1. Tree inclusion problem is to determine if T_2 is included in T_1.

3 Flaw of Unit-Cost Edit Distance

Unit-cost edit distance contains the similarity of node labels and that of tree structure simultaneously. Hence, even if a pair of trees share many nodes with the same label, the high label similarity between them is hard to be recognized from the unit-cost edit distance, if their tree sizes or shapes are much different.

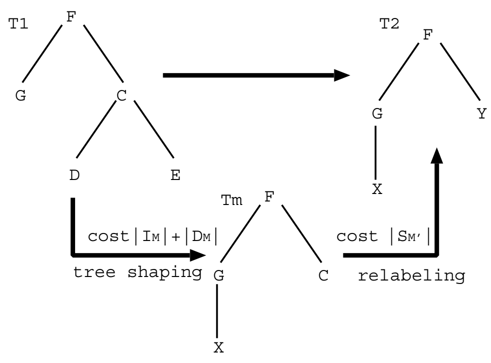

Fig. 3. Decomposition of tree conversion

Fig. 2 illustrates the above claim. In this figure, T_1 is a subtree of T_2 and every node label that appears in T_1 also emerges in T_2. By contrast, T_3 do not include the same label as T_1 at all. Here, $D(T_1, T_2)$, the unit-cost edit distance between T_1 and T_2, becomes 8, where the optimal tree mapping is to insert 8 nodes into T_1. On the other hand, $D(T_1, T_3) = 5$, where the optimal tree mapping is to delete node f from T_1 and relabel the remaining 4 nodes in T_1. Thus, T_1 becomes closer to T_3 than T_2 under unit-cost edit distance, despite the label similarity is higher between T_1 and T_2 than between T_1 and T_3.

This implies that unit-cost edit distance does not suit for applications for which label similarity should be paid much attention.

4 Our Dissimilarity Measures

After mentioning the decomposition of unit-cost edit distance in Sect. 4.1, our measures are defined in Sect. 4.2.

4.1 Decomposition of Unit-Cost Edit Distance

Especially, given two trees T_1 and T_2, we decompose the optimal tree mapping from a tree with more nodes to a tree with less nodes into node edit operations to match node labels and those to match tree structure. Let $|T_1| \geq |T_2|$ in the subsequence, where $|T|$ indicates the number of nodes in T and called the *tree size* of T. When $|T_1| < |T_2|$, they are permuted. Let $M(T_1, T_2)$ be the optimal tree mapping from T_1 to T_2 which corresponds to the unit-cost edit distance. We use Fig. 3 for explanation. Among the three types of node edit operations, only insertion and deletion are related to the change of the tree shape. Hence, we may suppose that M follows the next two steps in order.
Step 1: The tree shape is matched to T_2 by insertion and deletion operations.
Step 2: After Step 1, the node labels are made consistent with T_2 by relabeling.

Step 1 is further divided into two substeps in the next way.

Step 1a: The tree size is matched to T_2 by deleting some nodes from T_1.

Step 1b: The tree shape is matched between two trees of the same size by insertion and deletion operations.

Step 1 becomes $T_1 \to T_m$ and Step 2 becomes $T_m \to T_2$ in Fig. 3. Let S'_M is the set of non-free relabeling operations performed in M. Then the cost in Step 1 is $|I_M| + |D_M|$ and that in Step 2 is $|S'_M|$. In the example of Fig. 3, as two nodes D and E are removed and node X is inserted in Step 1, $|I_M| + |D_M| = 3$. Then, as Step 2 includes only one non-free relabeling operation (C, Y), $|S'_M| = 1$.

Since the cost in Step 1a is obviously $|T_1| - |T_2|$, the cost in Step 1b becomes $|I_M| + |D_M| - (|T_1| - |T_2|)$. Note that the number of insertion operations and that of deletion operations are the same in Step 1b, because the tree size does not change in Step 1b. In addition, insertion operations are performed only in Step 1b. Hence, we have $|I_M| + |D_M| - (|T_1| - |T_2|) = 2|I_M|$.

Because the nodes inserted in Step 1b (e.g. node X in Fig 3) must have labels included in T_2, the matching of labels is realized by $|I_M|$ insertion operations in Step 1b and $|S'_M|$ relabeling operations in Step 2. On the other hand, the matching of tree structure is realized by deletion operations in Step 1a and insertion and deletion operations in Step 1b.

4.2 Definitions of Our Dissimilarity Measures

Label Dissimilarity Measure: In the optimal tree mapping M, the number of node edit operations for matching labels is exactly $|I_M| + |S'_M|$. The label dissimilarity measure is defined as (2) in which $|I_M| + |S'_M|$ is normalized by the tree size $|T_2|$. The term "label dissimilarity measure" is abbreviated as LDM.

$$LDM(T_1, T_2) = \frac{|I_M| + |S'_M|}{|T_2|} \tag{2}$$

Since the matched labels remain in T_2, $0 \le |I_M| + |S'_M| \le |T_2|$. Hence, LDM takes a value between 0 and 1. Note that LDM takes the preservation of the order of nodes common to T_1 and T_2 into account. LDM has the next features.

- If and only if T_1 includes T_2, the LDM between them equals 0, because the optimal tree mapping consists of only deletion operations.
- When T_1 and T_2 do not have any common label at all, the LDM becomes 1, since any node in T_2 must be prepared by means of insertion or relabeling.

For the trees in Fig. 3, the LDM grows $\frac{1+1}{4} = 0.5$. In this way, LDM generalizes tree inclusion and measures the extent that T_1 includes T_2, even if T_2 is not completely included in T_1.

Structural Dissimilarity Measure (SDM) is defined as Formula (3).

$$SDM(T_1, T_2) = \frac{1}{2}\left(\frac{|T_1| - |T_2|}{|T_1|} + \frac{|I_M|}{|T_2|}\right) \tag{3}$$

The first term in (3) divides the cost for Step 1a by T_1 to exclude the influence of the tree size. Obviously, $0 \leq \frac{|T_1|-|T_2|}{|T_1|} \leq 1$. The second term in (3) corresponds to the cost of matching the shapes of the two trees of the same size in Step 1b, which equals $2|I_M|$. Note that the size of the two trees in Step 1b is $|T_2|$. As a tree mapping which deletes all nodes from one tree and then inserts all nodes contained in T_2 is feasible, the cost for Step 1b is at most $2|T_2|$. Thus, we have $0 \leq \frac{|I_M|}{|T_2|} \leq 1$. Hence, SDM takes a value between 0 and 1.

4.3 Related Works

LDM works as a measure for approximate tree inclusion. With respect to approximate tree inclusion, Schlieder and Naumann [6] measures the quality of a tree inclusion by the number of nodes skipped in the tree mapping. Pinter et al. [7] allows the inexact matching of node labels in subtree homeomorphism, a special case of tree inclusion. They rank subtree homeomorphisms by label similarity. These two works disallow inexact tree inclusion in terms of tree structure unlike our approach. Sanz et al. [8] studies approximate subtree identification which admits an inexact matching of tree structure like our paper. Although their algorithm is very fast, it cannot recognize the exact tree inclusion, as the ancestor relationship is weakened. Bunke and Shearer [9] proposes a dissimilarity measure in which the size of the maximum common embedded subtree is divided by $\max\{|T_1|, |T_2|\}$. Their measure generalizes not tree inclusion but graph isomorphism.

5 Application to Noisy Subsequence Tree Recognition

Our LDM is especially suitable for the noisy subsequence tree recognition [5] that is formulated as follows:

Suppose we have a database DB of labeled ordered trees. Let T be any tree from DB. U_T is an arbitrary subtree of T obtained by randomly deleting nodes from T. A noisy subsequence tree Y_T of T is constructed by garbling U_T by insertion, deletion and relabeling further. Fig. 4 illustrates an example. Here, the nodes surrounded by a circle in T constitutes U_T. The nodes surrounded by a rectangle in U_T and Y_T correspond to noises in the tree conversion $U_T \rightarrow Y_T$. The task of the noisy subsequence tree recognition problem is to identify the original tree T from the trees in DB, given Y_T. One major application of this problem is the comparison of RNA secondary structures.

For this problem, Oommen and Loke [5] computed the constrained tree edit distances between Y_T and every tree in DB and judged the tree in DB that is the least dissimilar to Y_T as the original tree T. The constrained tree edit distance is a special tree edit distance under the condition that the number of relabeling operations executed in the tree mapping is fixed. In particular, they assume that the number of relabeling operations denoted by L executed in $U_T \rightarrow Y_T$ can be obtained by some means. The computational complexity of the constrained tree edit distance between two trees T_1 and T_2 be-

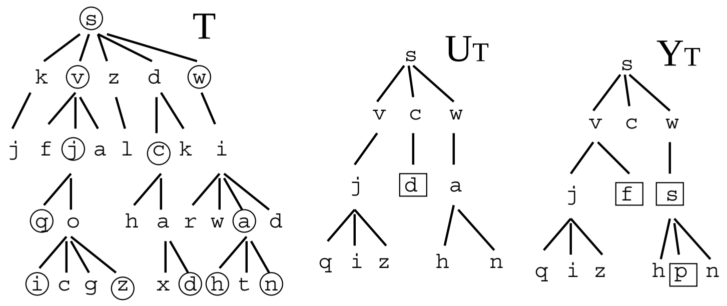

Fig. 4. Tree T, Subtree U_T and Noisy Subsequence Tree Y_T

comes $O(|T_1||T_2| * \min\{|T_1|, |T_2|\}^2 * span(T_1) * span(T_2))$, where $span(T) = \min\{\text{No. of leaves in } T, \text{No. of depths in } T\}$.

Our method utilizes the LDM instead of the constrained tree edit distance. The computational complexity of LDM is $O(|T_1||T_2| * span(T_1) * span(T_2))$ which inherits from tree edit distance. Our method is splendid, as it does not need L.

Our method is compared with the one by Oommen and Loke. We perform the same experiment as their paper [5]: We prepare 25 labeled ordered tree as DB which vary in sizes from 25 to 35 nodes. A label of a node is chosen uniformly randomly from the English alphabet. For a tree T in DB, a corresponding noisy subsequence tree is constructed in the following manner.

1. 60% of the nodes in T are randomly selected and removed to produce U_T.
2. In making Y_T, each node in U_T is deleted with a probability of 5% and relabeled with a probability that follows the QWERTY confusion matrix in [5] which models the errors in stroking a keyboard. Also, several nodes are inserted to randomly chosen places in U_T such that the number of inserted nodes follows the geometric distribution with an expectation value of 2.

10 noisy subsequence trees are made per a tree in DB. Thus, 250 noisy subsequence trees are generated in total. The average number of noises to deform U_T is 3.67 that consists of 1.98 insertion, 0.53 deletion and 1.16 relabeling operations. The average size of 25 trees in DB is 30.7 and that of the 250 noisy subsequence trees is 13.8.

For each noisy subsequence subtree, its original tree is searched from DB both with our method and Oommen's method [5]. As the result, 99.6 % out of the 250 noisy sybsequence trees are correctly recognized by our method, which is superior to the success ratio of 92.8% by Oommen's method reported in [5]. Though we also implemented the unit-cost constrained tree edit distance, we could not attain a success ratio higher than 90%. The execution time of our method is 21.8s, whereas that of Oommen's method is 114.4s. Each execution time contains the time to compute a dissimilarity measure $250 \times 25 = 6250$ times. Our method is faster than Oommen's method, since unit-cost edit distance is lighter to compute than unit-cost constrained edit distance.

We remark here that even if 5 relabeling operations and 5 insertion operations are performed on each U_T to create each Y_T, the success ratio still grows about 98%. Roughly speaking, LDM is not affected by the gap of the tree sizes between T and U_T, so LDM is robust.

6 Application to Classification of XML documents

This section demonstrates that our measures yield a more natural clustering result (that is, a dendrogram) when combined with hierarchical clustering algorithms than the unit-cost edit distance in classifying XML documents. Especially, when a set of XML documents from multiple different XML databases are given, our measures are good at bundling the XML documents from the identical database into the same cluster. Our method works in two phases as follows.

Step 1: The hierarchical clustering is executed by using the LDM only. From the clustering result, clusters $C_1, C_2, \ldots, C_k$ are determined.
Step 2: The hierarchical clustering is performed once more. This time, we use a weighted sum $L_{ij} + \alpha S_{ij}$ as the dissimilarity measure between a tree i and a tree j. Here, L_{ij} and S_{ij} are the LDM and the SDM between tree i and tree j. The weighting parameter α is determined from the constraint that the membership of clusters $C_1, C_2, \ldots, C_k$ remains unchanged. As the result, α is not so large.

Our method aims to categorize the XML documents from the same database into the same cluster in Step 1, because LDM can equate them without regard to the number of repeatable tags or elements that are specified with the '*' regular expression in the schema. Note that these repeatable tags/elements cause tree structural difference among the XML documents from the same database. After Step 1, Step 2 attempts to classify the XML documents inside each cluster, considering their structural dissimilarities.

Step 1 need to determine the number of clusters k. k can be estimated from the dendrogram such that a sudden increase of the LDM value between a pair of clusters to be merged in the agglomeration signifies that two heterogeneous clus-

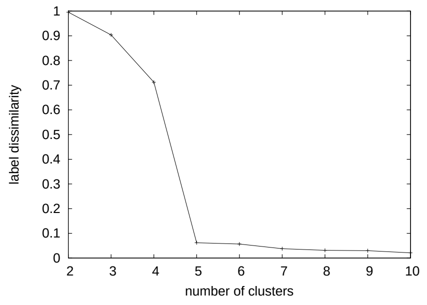

Fig. 5. Estimation of the number of clusters

Table 1. Performance comparison to the unit-cost edit distance

	Actors	Car	IndexTerms	OrdinaryIssue	No. of Misses
Our Method	20	20	20	20	0
Unit-Cost Edit Distance	9	20	20	19	12

ters that should not be united are merged. Step 2 produces a single dendrogram over the whole data by using a weighted sum of the LDM and the SDM.

6.1 Experimental Results

80 XML documents are sampled from the next 4 different XML databases (that is, 20 documents per a database): (1) XML-Actors, (2) the database of car catalogs from Edmunds.com, (3) ACM SIGMOD RECORD IndexTermsPage and (4) ACM SIGMOD RECORD OrdinaryIssuePage.

These documents are classified with our method, where the group averaging method is adopted as a hierarchical clustering algorithm. Fig. 5 displays the LDM values of the merged clusters in Step 1. Since the LDM rises greatly when the number of clusters is reduced from 5 to 4, the number of clusters is determined as 4, which is the correct answer. Each of the 4 clusters contains exactly 20 XML documents that come from the same database. Table 1 compares our method with the unit-cost edit distance. It shows how many XML documents from the same database appear as a single clump on the dendrogram. Inferior to our method, the unit-cost edit distance fails for 12 XML documents because it is annoyed by the difference in tree sizes. The final dendrogram by our method after Step 2 is published on our web page [10].

Instead, the final dendrogram by our method for a smaller dataset is presented here. The dataset consists of 4 documents from the XML-actors (i.e., A1,A2,A3,A4), 4 documents from the IndexTermsPage (i.e., I1, I2, I3, I4) and 8 documents from the OrdinaryIssuePage (i.e., from O1 to O7). The dendrogram is described in Fig. 6 where $\alpha = 2.8$. The three clusters are separated clearly.

Table 2 shows the number of 'articlesTuple' elements in the XML documents from the OrdinaryIssuePage database. This element is defined as a repeatable element in the schema. The cluster for the OrdinaryIssuePage database classifies the members according to the number of the elements, reflecting the structural dissimilarity among the members. Since each articlesTuple element corresponds to a technical paper in one journal issue, our method is to categorize several journal issues according to the number of papers published in them.

Table 2. the number of the 'articlesTuple' elements in XML documents

XML document	O1	O2	O3	O4	O5	O6	O7	08
No. of appearances	8	7	7	7	4	10	9	8

7 Conclusion

This paper proposes a novel method to extract a label dissimilarity measure and a structural dissimilarity measure between two trees separately by decomposing their unit-cost edit distance. As our dissimilarity measures are derived from unit-cost edit distance with a little overhead, they are expected to complement unit-cost edit distance for applications for which unit-cost edit distance do not perform well for the reason that unit-cost edit distance mixes the similarity of node labels and that of tree structure. Furthermore, our label dissimilarity measure works as a measure for approximate tree inclusion and can evaluate the extent of tree inclusion, if a tree is not completely included in another tree. We verify the effectiveness of our dissimilarity measures with two experiments.

References

1. Moulton, V., Zuker, M., Steel. M., Pointon, R., Penny, D.: Metrics on RNA Secondary Structures. J. of Computational Biology, Vol. 7 (2000) 277-292
2. Bille, P.: A Survey on Tree Edit Distance and Related Problems. Theoretical Computer Science, Vol. 337 (2005) 217–239
3. Zhang, K., Shasha, D.: Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. SIAM J. on Computing, Vol. 18 (1989) 1245–1262
4. Shasha, D., Zhang K.: Fast Algorithms for the Unit Cost Editing Distance between Trees. J. of Algorithms, Vol. 11 (1990) 581–621
5. Oommen, B. J., Loke, R. K. S.: On the Pattern Recognition of Noisy Subsequence Trees. IEEE Trans. on PAMI, Vol. 23, No. 9 (2001) 929–946
6. Schlieder, T., Naumann, F.: Approximate Tree Embedding for Querying XML Data. In Proc. of ACM SIGIR Workshop on XML and Information Retrieval (2000)
7. Pinter, R. Y., Rokhlenko, O., Tsur, D., Ziv-Ukelson, M.: Approximate Labelled Subtree Homeomorphism, In Proc. of CPM2004, (2004) 59-73
8. Sanz, I., Mesiti. M., Guerrini, G., Llavori, R. B.: Approximate Subtree Identification in Heterogeneous XML Documents Collections, In Proc. of 3rd International XML Database Symposium (XSym 2005), (2005) 192–206
9. Bunke. H., Shearer, K.: A Graph Distance Metric based on the Maximal Common Subgraph. Pattern Recognition Letters, Vol. 19, (1998) 255–259
10. http://sd.is.uec.ac.jp/˜koga/IDEALdata.html

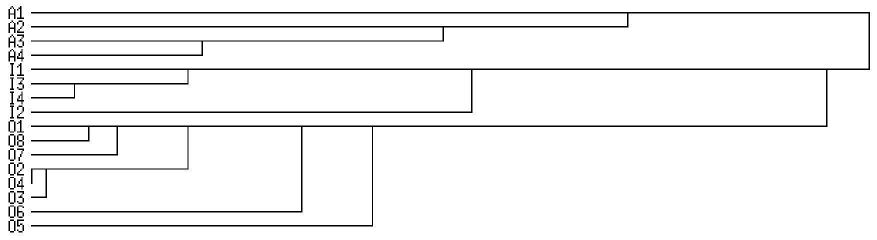

Fig. 6. Clustering result by our measures