LETTER

Split-Jaccard Distance of Hierarchical Decompositions for Software Architecture

Ki-Seong LEE[†], **Byung-Woo HONG**[†], **Youngmin KIM**[†], **Jaeyeop AHN**[†], ***Nonmembers***, ***and*** **Chan-Gun LEE**[†a)], ***Member***

SUMMARY Most previous approaches on comparing the results for software architecture recovery are designed to handle only flat decompositions. In this paper, we propose a novel distance called Split-Jaccard Distance of Hierarchical Decompositions. It extends the Jaccard coefficient and incorporates the concept of the splits of leaves in a hierarchical decomposition. We analyze the proposed distance and derive its properties, including the lower-bound and the metric space.

key words: *split, clustering, hierarchical, decomposition, distance, metric*

1. Introduction

There has been much effort toward deriving design information from the source code of a software system. Such information is critical to software maintenance and provides abstract views to the software engineers [1]. Software clustering techniques are commonly used for recovering architecture information such as a module view of the system. Various approaches to software clustering have been proposed, and each of them has strong and weak points from different aspects [1], [2].

Therefore, evaluating various software clustering algorithms is an important task to pick the most appropriate one for the user's context. One of the fundamental functions needed for the evaluation is to compare the decompositions which are the results generated from the software clustering algorithms.

There have been many attempts to compare the decompositions by computing the similarities or differences of them [3]–[5]. Unfortunately, most of them are designed to handle only flat decompositions [6]; i.e., a cluster cannot contain another cluster, hence they cannot compare hierarchical decompositions although most software clustering algorithms produce hierarchical ones. It should be noted that tree edit distance algorithms cannot be directly applied to the problem of comparing software decompositions, which are typically represented as unordered leaf-labeled trees [7].

In this paper, we propose a novel distance called Split-Jaccard Distance of Hierarchical Decompositions for comparing hierarchical decompositions. The proposed distance is influenced by the split-order distance [7] but our approach does not suffer from an anomaly shown in Sect. 3. We designed the distance by incorporating Jaccard distance [8] with the concept of split nodes in a tree. The lower-bound analysis and the metric property of the distance are also presented.

The rest of our paper is composed as follows. Section 2 defines the Split-Jaccard Distance and proves that it is a metric. Section 3 discusses related work on comparisons of software decompositions. Section 4 summarizes and concludes the paper.

Manuscript received June 1, 2014.
Manuscript revised September 18, 2014.
Manuscript publicized November 20, 2014.

[†]The authors are with Chung-Ang University, Seoul, 156–756 Korea.
a) E-mail: cglee@cau.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2014EDL8113

2. Split-Jaccard Distance of Hierarchical Decompositions

2.1 Notations

In this paper, we model the decomposition of a software into a leaf-labeled tree. The label of a leaf represents a software entity such as a function, a class, or a file. For example, a file $F_1.c$ would be mapped to a leaf with the label "$F_1.c$".

A software clustering algorithm groups the similar entities into a cluster. Each internal node of the tree represents a cluster which can contain entities and nested clusters. Note that the trees reflecting different decompositions for a given software architecture consist of the same set of the leaves, but their nested structures may be different. Every leaf corresponds to an entity to be clustered and the label of a leaf identifies the entity.

More formally, let T be a rooted tree defined on a set of vertices V and edges E. A vertex set V consists of internal vertices I that have children and terminal vertices L that have no children. We consider tree T as being a labeled tree where each vertex $v \in V$ has its label $\mathcal{L}(v)$ that distinguishes one vertex from another in V. The path $P(v_1, v_2)$ from a vertex v_1 to a vertex v_2 in tree T is a sequence of vertices $(x_0 = v_1, x_1, \ldots, x_{n-1}, x_n = v_2)$ that traverse their associated edges. The level $\ell(v)$ of a vertex v in tree T is defined by the length of the unique shortest path $P(r, v)$ from the root r to the vertex v. For a vertex $v \in V$ in tree T, the subtree $S(v\,;T)$ of T with $v \in V$ as its root is given by a tree consisting of v and its descendants and all edges incident to these descendants.

2.2 Structural Discrepancy

In the computation of a similarity between two sampled sets

A and B, we use the Jaccard coefficient $J(A, B)$ as a similarity measure defined by the size of the intersection of the sets A and B divided by the size of their union as follows [8]:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}, \tag{1}$$

where $|\cdot|$ denotes the cardinality of a set. The value of the Jaccard coefficient ranges from 0 to 1 for a pair of finite sets, where 0 represents no match and 1 represents perfect match. Note that we set $0/0 = 0$ in the definition.

We now introduce a discrepancy measure $d(T_1, T_2)$ between a pair of trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ based on the Jaccard coefficient, where V denotes the vertex set and E denotes the edge set. In our application, we assume that the number of terminal vertices $L_1 \subset V_1$ in tree T_1 is the same as the number of terminal vertices $L_2 \subset V_2$ in tree T_2. We assign a unique label l to each vertex v in terminal vertex set L using the following mapping $\mathcal{L}(v) = l$:

$$\mathcal{L} : L \to N, \tag{2}$$

where N is a set of labels such as $\{l_1, l_2, l_3, \cdots\}$, and each element in the label set $\mathcal{L}(L)$ of terminal vertex set L is assumed to be distinct:

$$\mathcal{L}(u) \neq \mathcal{L}(v) \text{ if } u \neq v, \tag{3}$$

where $u, v \in L$.

In the computation of similarity between T_1 and T_2, we also assume that the label set $\mathcal{L}(L_1)$ of terminal vertex set L_1 in tree T_1 is identical to the label set $\mathcal{L}(L_2)$ of terminal vertex set L_2 in tree T_2 so that we can build one-to-one correspondences between $\mathcal{L}(L_1)$ and $\mathcal{L}(L_2)$.

In our proposed discrepancy measure between a pair of trees T_1 and T_2, we consider a combination of pairwise structural similarity between their corresponding substructures as defined by:

$$S(u\,; T_1) \sim S(v\,; T_2) = J(N_1, N_2), \tag{4}$$

where $\sim$ denotes a similarity measure between two subtrees $S(u\,; T_1)$ and $S(v\,; T_2)$, and $S(u\,; T)$ denotes the subtree of $T = (V, E)$ with $u \in V$ being the root vertex of $S(u\,; T)$. For the identification of vertices, N_1 and N_2 denote the label sets of the terminal vertex sets in $S(u\,; T_1)$ and $S(v\,; T_2)$, respectively.

For a pair of labels l_1, l_2 that are included in both $\mathcal{L}(L_1)$ and $\mathcal{L}(L_2)$ where $\mathcal{L}(L_1)$ is constrained to be identical to $\mathcal{L}(L_2)$ and we denote the common label set by $\mathcal{L}(L) = \mathcal{L}(L_1) = \mathcal{L}(L_2)$, we define a structural discrepancy $d(l_1, l_2\,; T_1, T_2)$ between T_1 and T_2 given a pair of labels $l_1, l_2 \in \mathcal{L}(L)$ as follows:

$$d(l_1, l_2\,; T_1, T_2) = S(u\,; T_1) \sim S(v\,; T_2), \tag{5}$$

where the correspondence between subtrees $S(u\,; T_1)$ and $S(v\,; T_2)$ is established by assigning

$$u = \arg\min_p \{\ell(p) \,|\, p \in P(u_1, u_2)\}, u_1, u_2 \in L_1, \tag{6}$$

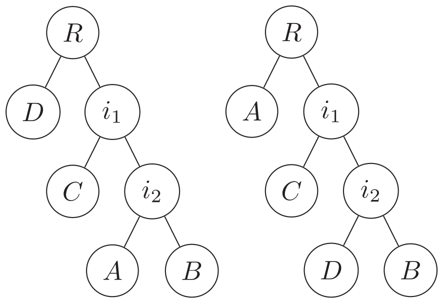

Fig. 1 Decompositions T_1 (left) and T_2 (right).

$$v = \arg\min_p \{\ell(p) \,|\, p \in P(v_1, v_2)\}, v_1, v_2 \in L_2, \tag{7}$$

$$l_1 = \mathcal{L}(v_1) = \mathcal{L}(u_1), \tag{8}$$

$$l_2 = \mathcal{L}(v_2) = \mathcal{L}(u_2), \tag{9}$$

where $\ell(p)$ denotes the level of vertex $p \in V$ in tree $T = (V, E)$ and the level of a vertex is defined by the length from the root vertex to the given vertex. $P(u_1, u_2)$ denotes a unique path that consists of a sequence of consecutive vertices connecting u_1 and u_2. In the above, u and v are typically called *splits* [7] because each of them corresponds to the nearest common ancestor of the leaves with the labels l_1 and l_2.

Given trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$, their structural similarity with a given pair of common labels $l_1 \in \mathcal{L}(L_1)$ and $l_2 \in \mathcal{L}(L_2)$ for the vertices in the terminal vertex sets $L_1 \subset V_1$ and $L_2 \subset V_2$ is computed by comparing subtrees $S(u\,; T_1)$ and $S(v\,; T_2)$ where u is a common ancestor of u_1 and u_2 with the maximum level and v is a common ancestor of v_1 and v_2 with the maximum level.

Then, we define a discrepancy measure $D(T_1, T_2)$ between T_1 and T_2 by considering the combination of structural comparisons with respect to all possible pairs of labels as follows:

$$D(T_1, T_2) = 1 - \frac{1}{\binom{n}{2}} \sum_{i=l_1}^{l_n} \sum_{j=i}^{l_n} d(i, j\,; T_1, T_2), \tag{10}$$

where n denotes the cardinality of the label set $\mathcal{L}(L) = \{l_1, l_2, \ldots, l_n\}$ and $\binom{n}{2} = \frac{n(n-1)}{2}$. Then, the value of the discrepancy $D(T_1, T_2)$ for any pair of trees T_1, T_2 ranges from 0 to 1, where 0 indicates perfect match and 1 indicates no match.

Before presenting the properties of the proposed distance, we provide an example of calculating the distance of two decompositions of a software in the following. Note that we assign labels to some internal nodes for the illustration purpose. As mentioned in Sect. 3, every leaf should be tagged with a label, but internal nodes do not have such a requirement.

Example 1. *Figure 1 shows two hierarchical decompositions T_1 and T_2 for a software architecture. The structural similarity between substructures for the pair of labels (A, B) is calculated as follows:*

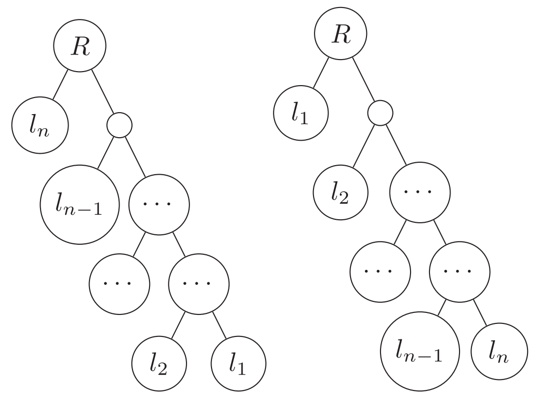

Fig. 2 Examples of trees that yield the maximum distance.

$$\begin{aligned} d(A,B;T_1,T_2) &= S(i_2;T_1) \sim S(R;T_2) \\ &= J(\{A,B\};T_1) \sim J(\{A,B,C,D\};T_2) \\ &= \frac{|\{A,B\} \cap \{A,B,C,D\}|}{|\{A,B\} \cup \{A,B,C,D\}|} = \frac{1}{2} \end{aligned}$$

Similarly, the remaining combinations of labels are computed as follows: $d(A,C,;T_1,T_2) = 0.75$*,* $d(A,D,;T_1,T_2) = 1.0$*,* $d(B,C,;T_1,T_2) = 0.5$*,* $d(B,D,;T_1,T_2) = 0.5$*,* $d(C,D,;T_1,T_2) = 0.75$*. Hence,* $D(T_1,T_2) = 0.67$*.*

We now analyze the range of our distance measure and the necessary condition that yields the maximal distance.

Lemma 1. *Let $T_1 = (V_1,E_1)$ and $T_2 = (V_2,E_2)$ be trees with vertex sets $V_1 = I_1 \cup L_1$ and $V_2 = I_2 \cup L_2$ where I denotes a set of internal vertices and L denotes a set of terminal vertices. A necessary condition for the structural distance $D(T_1,T_2)$ to be maximal is that each vertex in L_1 and L_2 has different level from 1 to n when $|L_1| = |L_2| = n$ with an assumption that $|I_1| = |L_1|$ and $|I_2| = |L_2|$.*

Proof. Our structural distance is measured based on the ratio of the cardinality of the intersection and the union of terminal vertices in the corresponding subtrees for all the combination of label pairs. Let $L_1^{(i,j)}$ and $L_2^{(i,j)}$ be sets of terminal vertices of the subtrees $S(u;T_1)$ and $S(v;T_2)$ which are considered in the computation of $d(l_1,l_2;T_1,T_2)$. Thus, the structural distance increases when the following quantity decreases:

$$\sum_{i=l_1}^{l_n} \sum_{j=i}^{l_n} \left(|L_1^{(i,j)}| + |L_2^{(i,j)}| \right), \tag{11}$$

A pair of trees that are shaped in such a way that the above quantity is minimized yield the minimal set of intersection in the computation of the distance. A graphical illustration of trees that satisfy this condition is presented in Fig. 2. □

Lemma 2. *A necessary condition for the structure distance $D(T_1,T_2)$ to be maximal is to satisfy the following condition:*

$$\ell(u) = n - \ell(v) + 1, \quad \mathcal{L}(u) = \mathcal{L}(v) \tag{12}$$

where $u \in L_1$, $v \in L_2$, and n is the number of terminal vertices in each tree.

Proof. The cardinality of the union sets in the computation of the structural distance is maximized when the label of the terminal vertices in one tree is reversely ordered to the label of the terminal vertices in the other tree with the shape of one tree being the same as the shape of the other tree as shown in Fig. 2. □

Theorem 1. *Let $T_1 = (V_1,E_1)$ and $T_2 = (V_2,E_2)$ be trees with vertex sets $V_1 = I_1 \cup L_1$ and $V_2 = I_2 \cup L_2$ where I denotes a set of internal vertices and L denotes a set of terminal vertices. The structural distance $D(T_1,T_2)$ between T_1 and T_2 is bounded by $1 - \frac{(n+4)(n-1)}{3n(n+1)}$ where a constraint on the cardinality of the terminal vertex sets is imposed by $|L_1| = |L_2|$.*

Proof. In the computation of the structural distance between trees T_1 and T_2, it is assumed that $|L_1| = |L_2|$ and $\mathcal{L}(L_1) = \mathcal{L}(L_2)$. Let us denote $N = \{l_1, l_2, \cdots, l_n\}$ a set of common label sets $N = \mathcal{L}(L_1) = \mathcal{L}(L_2)$. The structural distance $D(T_1,T_2)$ between T_1 and T_2 is maximized when the following quantity is maximized:

$$\sum_{i=1}^{n} |\ell(u_i) - \ell(v_i)|^2, \ \mathcal{L}(u_i) = \mathcal{L}(v_i) = l_i, \ l_i \in N \tag{13}$$

The visual illustration for an example of trees yielding the maximal structural distance between them in Fig. 2. The sum of structural discrepancy between corresponding subtrees for all the combination of label pairs is obtained by

$$\begin{aligned} &\sum_{i=l_1}^{l_n} \sum_{j=i}^{l_n} d(i,j;T_1,T_2) \\ &= \left(\frac{2}{n} + \frac{3}{n} + \cdots + \frac{n}{n} \right) + \left(\frac{2}{n} + \frac{3}{n} + \cdots + \frac{n-1}{n} \right) \\ &\quad + \cdots + \frac{2}{n} \\ &= \frac{2(n-1) + 3(n-2) + \cdots + (n-1)2 + n}{n} \\ &= \frac{1}{n} \sum_{k=2}^{n} k(n-(k-1)) = \frac{n^2+3n-4}{6} \end{aligned}$$

Then, the structural distance $D(T_1,T_2)$ of T_1 and T_2 which are maximally different in terms of our discrepancy measure becomes:

$$\begin{aligned} D(T_1,T_2) &= 1 - \frac{2}{n(n+1)} \frac{n^2+3n-4}{6} \\ &= 1 - \frac{(n+4)(n-1)}{3n(n+1)} \end{aligned} \tag{14}$$

□

2.3 Metric Space

An order pair (T,D) where T denotes a set representing a tree and D denotes the discrepancy measure on T forms a metric space, which indicates that a function $D : T \times T \to \mathbb{R}$ for any $T_1, T_2, T_3 \in T$ satisfies the following properties:

(a) $D(T_1, T_2) \geq 0$

(b) $D(T_1, T_2) = 0$ if and only if $T_1 = T_2$

(c) $D(T_1, T_2) = D(T_2, T_1)$

(d) $D(T_1, T_3) \leq D(T_1, T_2) + D(T_2, T_3)$

The condition (a) follows from the other three properties:

$$2D(T_1, T_2) = D(T_1, T_2) + D(T_2, T_1) \geq D(T_1, T_1) = 0. \tag{15}$$

The condition (b) can be proved as follows:

$$\begin{aligned} T_1 = T_2 &\iff d(l_1, l_2\,; T_1, T_2) = 1,\ \forall l_1, l_2 \in \mathcal{L}(L) \\ &\iff D(T_1, T_2) = 0 \end{aligned} \tag{16}$$

where $\mathcal{L}(L)$ denotes the label set of L_1 and L_2. The condition (c) holds due to the symmetric property of the Jaccard coefficient as follows:

$$\begin{aligned} &J(N_1, N_2) = J(N_2, N_1) \\ \implies\ &d(l_1, l_2,\ T_1, T_2) = d(l_1, l_2,\ T_2, T_1) \\ \implies\ &D(T_1, T_2) = D(T_2, T_1) \end{aligned} \tag{17}$$

where N_1 denotes the label set of the terminal vertices in $S(v\,; T_1)$ and N_2 denotes the label set of the terminal vertices in $S(u\,; T_2)$ following the definition given in Eq. (4). The condition (d) is satisfied by the following relations:

$$\begin{aligned} &J(N_1, N_3) \leq J(N_1, N_2) + J(N_2, N_3) \implies \\ &d(l_1, l_2,\ T_1, T_3) \leq d(l_1, l_2,\ T_1, T_2) + d(l_1, l_2,\ T_2, T_3), \\ &\forall l_1, l_2 \in \mathcal{L}(L) \implies \\ &d(l_1, l_2,\ T_1, T_3) \leq d(l_1, l_2,\ T_1, T_2) + d(l_1, l_2,\ T_2, T_3), \\ &\forall l_1, l_2 \in \mathcal{L}(L) \implies D(T_1, T_3) \leq D(T_1, T_2) + D(T_2, T_3) \end{aligned} \tag{18}$$

2.4 Complexity

Let us derive the complexity of computing $d(T_1, T_2)$ for the trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ where V denotes the vertex set and E denotes the edge set. n represents the number of terminal vertices $L_1 \subset V_1$ and it is the same as the number of terminal vertices $L_2 \subset V_2$.

As shown in Eq. (10), computing the proposed distance requires the cardinality of the intersection and the union of terminal vertices in the subtrees of the splits for all the combination of label pairs.

For a pair of labels (i, j), let $L_1^{(i,j)}$ and $L_2^{(i,j)}$ be sets of terminal vertices of the subtrees of $S(u; T_1)$ and $S(v; T_2)$, respectively where u and v are the splits of (i, j) in each tree. $L_1^{(i,j)}$ can be derived as a bit vector of length n. Each bit maps to a unique label and indicates whether the corresponding terminal vertex with the label is a member of $S(u; T_1)$. $L_2^{(i,j)}$ can be derived similarly. We assume that the access time to each bit in the vector mapped to a label is $O(1)$. Locating two terminal vertices corresponding to i and j and finding their split u in T_1 can be done in $O(|V_1|)$. Collecting the terminal vertices rooted at u and constructing the bit vector representing $L_1^{(i,j)}$ can be done in $O(|V_1|)$. $L_2^{(i,j)}$ can be constructed similarly in $O(|V_2|)$. Computing the intersection and the union of two bit vectors takes $O(n)$.

Therefore, the time complexity of computing the metric is $O(n^2(|V_1| + |V_2|))$ because we need to consider all the combinations of label pairs. The space complexity is the same as the size of the trees $O(|V_1| + |V_2|)$.

3. Related Work

There have been many approaches to compare tree structures. The tree edit distance is a well-known metric to compare two trees, where their internal and leaf nodes are all labeled. The tree edit distance is considered intuitive because it represents the number of items to be edited to convert a given tree to the corresponding tree. The typical edit operations include insert, delete, and rename. It is known that the tree edit distance computation for unordered trees is NP-complete [9]. There are a few algorithms [9]–[13] with polynomial complexities for ordered trees. An extensive survey on tree edit distance algorithms can be found in [14].

It is worth noting that the tree edit distance does not fit to the problem addressed in our paper. For the decomposition results obtained for software architecture recovery, there is no ordering among the siblings; which means that the result is an unordered leaf-labeled tree, as mentioned earlier in Sect. 1. As indicated in [7], the general tree edit distance algorithms may generate answers contradicting our understanding of the hierarchies, when we consider unordered leaf-labeled trees as ordered fully-labeled trees. Figure 3 illustrates such examples. The tree edit distance between T_1 and T_2 is 2 because we can make them identical by editing two labels. Note that these trees in fact represent the same decompositions in our purpose. On the other hand, The tree edit distance between T_1 and T_3 is also 2, where we indeed have different decompositions.

For this reason, in the following we restrict ourselves to the approaches to compare flat decompositions or unordered leaf-labeled trees. Most previous work on comparing software clustering results has focused on only flat decompositions [6]. Our work most closely relates to recent work by Shtern et al. [6], [15] and Zhang et al. [7].

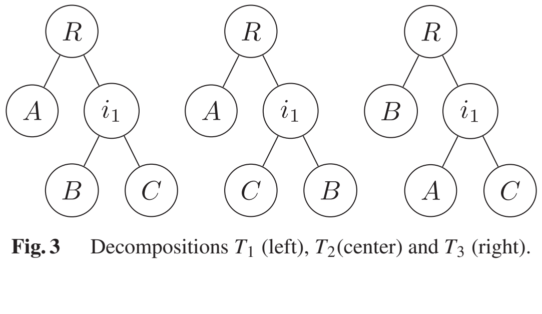

Fig. 3 Decompositions T_1 (left), T_2(center) and T_3 (right).

The END framework [15] is designed to apply the evaluation methods, which are originally capable of processing only flat decompositions, for hierarchical decompositions. The framework enables this by incrementally reducing the hierarchical decompositions into flat ones level by level. The END framework produces a vector, which needs to be transformed to a scalar value by a user function. The framework does not provide a specific guideline for the user functions [6].

UpMoJo [6] extends MoJo so that it can compute the distance between hierarchical decompositions. As well as classical Join and Move operations of MoJo, it adds Up operation so that the entities can move to upper levels in the decomposition. However, UpMoJo is not a metric because its range is not confined.

Zhang et al. [7] proposed a metric called Split-Order distance. For each triple of leaves (l_1, l_2, l_3) where $l_1 \neq l_2 \neq l_3$, the algorithm computes the order of the split nodes for (l_1, l_2) and (l_1, l_3) in each tree. The order of the nodes i and j is defined as "<" when i is closer to the root than j. In case j is closer, then the result is "<". Otherwise the result is "=". If the orders from the trees are different, then the metric increases by one. For example, the splits of (A, B) and (A, C) are i_2 and i_1 respectively in T_1. In T_2, the splits of the corresponding pairs are both R. Thus, the order results from the trees are ">" and "=", hence this increases the metric by one for this particular triple.

Note that Fig. 1, which was used in Example 1, depicts the case where the split-order distance [7] determines that the two decompositions are totally different. However, the decompositions are slightly different in fact; only the nodes with the labels A and D are swapped and the configurations of the rest nodes are the same. This phenomenon is referred to as the anomaly of split-order distance. Our proposed distance does not suffer from this issue.

4. Conclusions

We proposed Split-Jaccard Distance of Hierarchical Decompositions, which can be used for comparing the results obtained for software architecture recovery. It is based on the Jaccard coefficient and utilizes the information regarding the splits of leaves in the hierarchical decompositions. We proposed use of this distance as a metric and analyzed its properties, including the lower-bound.

Acknowledgements

This research was supported by the Chung-Ang University Excellent Student Scholarship, the National Research Foundation (NRF-2011-0013924, NRF-2014-005519) and the MSIP (Ministry of Science, ICT and Future Planning) under the ITRC (Information Technology Research Center) support Program (NIPA-2014-H0301-14-1023) supervised by the NIPA.

References

[1] O. Maqbool and H.A. Babri, "Hierarchical clustering for software architecture recovery," IEEE Trans. Softw. Eng., vol.33, no.11, pp.759–780, 2007.
[2] M. Shtern and V. Tzerpos, "Methods for selecting and improving software clustering algorithms," Software: Practice and Experience, vol.44, no.1, pp.33–46, 2014.
[3] V. Tzerpos and R.C. Holt, "Mojo: A distance metric for software clusterings," Proc. Sixth Working Conference on Reverse Engineering, 1999, pp.187–193, 1999.
[4] N. Anquetil and T.C. Lethbridge, "Experiments with clustering as a software remodularization method," Proc. Sixth Working Conference on Reverse Engineering, 1999, pp.235–255, 1999.
[5] B.S. Mitchell and S. Mancoridis, "Comparing the decompositions produced by software clustering algorithms using similarity measurements," Proc. IEEE International Conference on Software Maintenance, 2001, pp.744–753, 2001.
[6] M. Shtern and V. Tzerpos, "Lossless comparison of nested software decompositions," 14th Working Conference on Reverse Engineering, 2007. WCRE 2007, pp.249–258, 2007.
[7] Q. Zhang, E.Y. Liu, A. Sarkar, and W. Wang, "Split-order distance for clustering and classification hierarchies," in Scientific and Statistical Database Management, pp.517–534, Springer, 2009.
[8] P. Jaccard, "The distribution of the flora in the alpine zone," New Phytologist, vol.11, no.2, pp.37–60, 1912.
[9] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," SIAM J. Comput., vol.18, no.6, pp.1245–1262, 1989.
[10] H. Touzet, "Tree edit distance with gaps," Inf. Process. Lett., vol.85, no.3, pp.123–129, 2003.
[11] E.D. Demaine, S. Mozes, B. Rossman, and O. Weimann, "An optimal decomposition algorithm for tree edit distance," ACM Trans. Algorithms, vol.6, no.1, pp.2:1–2:19, 2009.
[12] M. Pawlik and N. Augsten, "Rted: A robust algorithm for the tree edit distance," Proc. VLDB Endow., vol.5, no.4, pp.334–345, Dec. 2011.
[13] M. Pawlik and N. Augsten, "A memory-efficient tree edit distance algorithm," in Database and Expert Systems Applications, ed. H. Decker, L. Lhotska, S. Link, M. Spies, and R.R. Wagner, Lect. Notes Comput. Sci., vol.8644, pp.196–210, 2014.
[14] P. Bille, "A survey on tree edit distance and related problems," Theor. Comput. Sci., vol.337, no.1, pp.217–239, 2005.
[15] M. Shtern and V. Tzerpos, "A framework for the comparison of nested software decompositions," Proc. 11th Working Conference on Reverse Engineering, 2004, pp.284–292, 2004.