Accurate and Fast Computation of Approximate Graph Edit Distance based on Graph Relabeling

Sousuke Takami and Akihiro Inokuchi
School of Science and Technology, Kwansei Gakuin University, 2-1 Gakuen, Sanda, Hyogo, Japan

Keywords: Graph Edit Distance, Graph Relabeling, Graph Classification.

Abstract: The graph edit distance, a well-known metric for determining the similarity between two graphs, is commonly used for analyzing large sets of structured data, such as those used in chemoinformatics, document analysis, and malware detection. As computing the exact graph edit distance is computationally expensive, and may be intractable for large-scale datasets, various approximation techniques have been developed. In this paper, we present a method based on graph relabeling that is both faster and more accurate than the conventional approach. We use unfolded subtrees to denote the potential relabeling of local structures around a given vertex. These subtree representations are concatenated as a vector, and the distance between different vectors is used to characterize the distance between the corresponding graphs. This avoids the need for multiple calculations of the exact graph edit distance between local structures. Simulation experiments on two real-world chemical datasets are reported. Compared with the conventional technique, the proposed method gives a more accurate approximation of the graph edit distance and is significantly faster on both datasets. This suggests the proposed method could be applicable in the analysis of larger and more complex graph-like datasets.

1 INTRODUCTION

Graphs are one of the most natural means of representing structured data. For instance, a chemical compound can be represented as a graph in which each vertex corresponds to an atom, each edge corresponds to a bond between two atoms, and the label of each vertex corresponds to the atom type. With recent improvements in system throughput, the need to analyze large numbers of graphs has arisen, and the topic of graph mining has received considerable interest because the knowledge present in structured data can be applied to various real-world datasets. For example, in cheminformatics, certain properties of chemical compounds (e.g., mutagenicity or toxicity) can be identified by analyzing their structural information, and in bioinformatics, the prediction of protein–protein interactions is beneficial for drug discovery.

When analyzing datasets of graphs, one of the most critical measures is the dissimilarity (or similarity) among the graphs. A representative measure of the dissimilarity is the graph edit distance. The graph edit distance $d(g_1, g_2)$ between graphs g_1 and g_2 is defined as the minimum length of the sequence of edit operations needed to transform g_1 into g_2, where one edit operation includes the insertion, deletion, or the substitution of a vertex and edge in the graphs. The method based on the $A^\star$ algorithm is a well-known technique for computing the *exact* graph edit distance (Hart et al., 1968). However, this method cannot be applied to large graphs, because the problem of obtaining the exact graph edit distance between two graphs is known to be NP-complete.

To overcome this difficulty, a method that uses the minimum matching problem of a complete bipartite graph (V_1, V_2, E, w) has been proposed to compute the *approximate* graph edit distance between the graphs (Riesen, 2015). In a bipartite graph, V_1 and V_2 correspond to the vertices of g_1 and g_2, respectively, and w is a function assigning values of the dissimilarities between local structures around the vertices to edges in the bipartite. Given the complete bipartite graph, the matching problem returns a mapping from the vertices in g_1 to vertices in g_2 that is solvable in $O(b^3)$, where $b = \max\{|V_1|, |V_2|\}$. Computing the approximate graph edit distance using this method is computationally simpler than computing the exact graph edit distance. In this method, the local structures are important in obtaining as accurate an approximate distance as possible. The local structure around each vertex v is represented by star structures (Zeng et al., 2009), random walks

Takami, S. and Inokuchi, A.
Accurate and Fast Computation of Approximate Graph Edit Distance based on Graph Relabeling.
DOI: 10.5220/0006540000170026
In *Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2018)*, pages 17-26
ISBN: 978-989-758-276-9

from v (Gaüzère et al., 2014), and subgraphs induced by vertices reachable within h steps from v, called limited-size subgraphs (Carletti et al., 2015). If complex and/or large structures such as limited-size subgraphs are used as local structures, the computation time required to assign values to edges in the bipartite graph can be significant. In contrast, if simple structures such as stars or walks are used, the accuracy of the approximated graph edit distance decreases. A recently proposed method (Carletti et al., 2015) measures the dissimilarity between local structures around v_1 in g_1 and v_2 in g_2 with small limited-size subgraphs. The dissimilarity in this approach is the exact graph edit distance. Thus, although this method gives a relatively accurate approximate graph edit distance between g_1 and g_2, it requires considerable computation time to measure multiple exact graph edit distances for the small subgraphs.

In this paper, we tackle the problem of measuring the approximate graph edit distance between graphs and propose an accurate and fast method. In the proposed method, each of the complex local structures is represented by a vector, which enables this method to be applied to complex and large local structures and ensures fast computation. The proposed method requires $O(b^2h(|\Sigma|+b)$ time to compute the approximate graph edit distance, where Σ is the set of vertex labels in the graphs. In addition, the method requires $O(|\Sigma|b+b^2)$ memory.

The remainder of this paper is organized as follows. Section 2 formalizes the problem of measuring the graph edit distance and explains the existing method based on the minimum matching problem of a complete bipartite graph. In Section 3, we propose an accurate and fast method for measuring the graph edit distance by representing local structures as vectors. In Section 4, we verify the computational efficiency of the proposed method and compare it with the conventional method in terms of accuracy using real-world datasets. Finally, we conclude the paper in Section 5.

2 PRELIMINARIES

This paper tackles the problem of computing the approximate graph edit distance between graphs. First, we define the terminology used to solve the problem. An undirected graph is represented as $g = (V, E, \Sigma, \ell)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_\Sigma\}$ is a set of vertex labels, and $\ell : V \to \Sigma$ is a function that assigns a label to each vertex in the graph. Additionally, the set of vertices in graph g is represented as $V(g)$. Although we assume that only the vertices in the graphs have labels, the methods in this paper can be applied to graphs where both the vertices and edges have labels (Hido and Kashima, 2009). The vertices adjacent to vertex v are represented as $N(v) = \{u \mid (v,u) \in E\}$. The average number of adjacent vertices is represented as $\overline{d}$. A sequence of vertices from v to u is called a path, and its step refers to the number of edges on that path. A path is said to be simple if and only if it does not have repeating vertices. The paths discussed in this paper are not always simple.

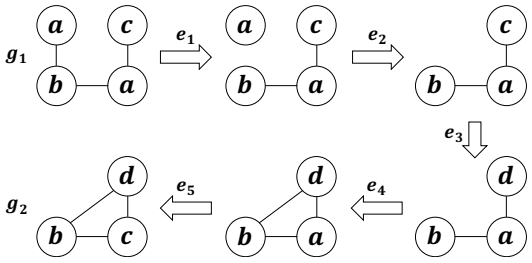

Figure 1: Sequence of edit operations for transforming g_1 into g_2.

The graph edit distance is one of the most representative metrics to measure the dissimilarity between graphs. The graph edit distance $d(g_1, g_2)$ between graphs g_1 and g_2 is defined as the minimum length of the sequence of edit operations needed to transform g_1 into g_2, where one edit operation includes the insertion or deletion of a vertex/edge and the substitution of a vertex label. Although the edit distance was originally proposed for measuring the dissimilarity between two strings, the metric was extended to graphs by introducing graph edit operations.

Figure 1 shows a certain sequence of edit operations that consists of one deletion of vertex (e_2), one insertion of edge (e_4), one deletion of edge (e_1), and two substitutions of labels (e_3, e_5). Computing the edit distance between g_1 and g_2 is equivalent to searching for the minimum length of the sequence of edit operations needed to transform g_1 into g_2. The method based on the $A^\star$ algorithm is a well-known technique for computing the exact graph edit distance (Hart et al., 1968). However, this method cannot be applied to large graphs, because the problem of obtaining the exact graph edit distance between two graphs is known to be NP-complete. To address this drawback, various methods for computing the approximate graph edit distance have been proposed.

The graph edit distance is an important measure for analyzing graphs, and is applicable to a wide range of practical applications such as fingerprint authentication (Choi and Kim, 2010), malware detection (Kinable and Kostakis, 2011), chemoinformatics (Kashima et al., 2003), bioinformatics, and doc-

ument analysis (Wang et al., 2014). We explain three typical fundamental problems that use the analysis of graphs. First, given a set of graphs, we can apply methods for clustering the graphs to detect some latent groups within them (Robles-Kelly and Hancock, 2003). Second, given a set of graphs and a query graph g_q, we can apply a similarity search to the graphs (Yan et al., 2005). Third, given a set of graphs with class labels, we can apply a Support Vector Machine (SVM) and kernel functions to forecast the class of a graph whose label is unknown (Kashima et al., 2003). The graph edit distance is converted to similarity using the Gaussian kernel, which is defined as

$$k(g_1,g_2) = \exp\left(-\frac{d(g_1,g_2)^2}{2\sigma^2}\right). \quad (1)$$

Most machine learning and data mining algorithms that are designed to analyze d-dimensional vectors can be applied to graphs using this kernel.

3 APPROXIMATE GRAPH EDIT DISTANCE

This section surveys a framework for computing the approximate graph edit distance between two graphs using the linear sum assignment problem (LSAP). The problem of computing the exact graph edit distance between graphs $g_1 = (V_1, E_1, \Sigma, \ell_1)$ and $g_2 = (V_2, E_2, \Sigma, \ell_2)$ is formalized as follows (Riesen and Bunke, 2009; Riesen, 2015): First, the set of vertices V_1 is extended to

$$V_1^+ = V_1 \cup \overbrace{\{\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_{b-a}\}}^{b-a \text{ empty vertices}}$$

where $|V_1| = a$, $|V_2| = b$, and we assume that $|V_1| \leq |V_2|$. The graph edit distance computation is eventually performed on graphs $g_1 = (V_1^+, E_1, \Sigma, \ell_1)$ and $g_2 = (V_2, E_2, \Sigma, \ell_2)$. Additionally, a cost matrix for editing g_1 to g_2 is defined as

$$C = \begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ a \\ 1 \\ \vdots \\ b-a \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \cdots & b \end{array} \\ \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1b} \\ c_{21} & c_{22} & \cdots & c_{2b} \\ \vdots & \vdots & \ddots & \vdots \\ c_{a1} & c_{a2} & \cdots & c_{ab} \\ c_{\varepsilon 1} & c_{\varepsilon 2} & \cdots & c_{\varepsilon b} \\ \vdots & \vdots & \ddots & \vdots \\ c_{\varepsilon 1} & c_{\varepsilon 2} & \cdots & c_{\varepsilon b} \end{pmatrix} \end{array}, \quad (2)$$

where

- c_{ij} denotes the cost of replacing a label of vertex $v_i \in V_1$ with a label of vertex $v_j \in V_2$ and
- $c_{\varepsilon i}$ denotes the cost of inserting a vertex v_i into g_1 to edit g_1 to g_2.

Additionally, we denote the cost of editing an edge (v_i, v_j) in g_1 into an edge $(v_{i'}, v_{j'})$ in g_2 by $c((v_i, v_j) \to (v_{i'}, v_{j'}))$. In this paper, we assume that only vertices in graphs have labels, and so $c((v_i, v_j) \to (v_{i'}, v_{j'}))$ is the cost of inserting or deleting an edge. When g_1 is edited to g_2, the mapping between the two sets of vertices is denoted by a bijective function $\varphi : V_1^+ \to V_2$. Then, the total cost of editing g_1 to g_2 via φ is

$$dist(g_1,g_2,\varphi) = \sum_{i=1}^{b} c_{i\varphi(i)} + \sum_{i=1}^{b-1}\sum_{j=i+1}^{b} c\left((v_i,v_j) \to \left(v_{\varphi(i)}, v_{\varphi(j)}\right)\right). \quad (3)$$

Therefore, the exact edit distance between g_1 and g_2 is

$$d(g_1,g_2) = \min_{\varphi \in \Phi} dist(g_1,g_2,\varphi), \quad (4)$$

where Φ is a set of all possible permutations of integers $1, 2, \cdots, b$. Equation (4) is a type of quadratic assignment problem that is known to be NP-complete (Koopmans and Beckmann, 1957), although it reduces to the LSAP if Eq. (3) does not contain its second term.

Riesen et al. proposed an efficient algorithmic framework that enables us to obtain the approximate graph edit distance by omitting the second term of Eq. (3). The framework first solves

$$\hat{\varphi} = \arg\min_{\varphi \in \Phi} \sum_{i=1}^{b} c_{i\varphi(i)} \quad (5)$$

and then obtains the approximate edit distance between g_1 and g_2 by substituting $\hat{\varphi}$ into Eq. (3). Equation (5) implies that each vertex v_i in g_1 should, as far as possible, be mapped to a vertex in g_2 that has the same label as v_i. Solving Eq. (5) is equivalent to the minimum matching problem of a bipartite graph whose vertices are V_1^+ and V_2, and whose edge weights are c_{ij} in Eq. (2). Therefore, this problem is tractable in $O(b^3)$. However, because the structural information of the two graphs is ignored and only vertex labels are taken into account in the optimization problem shown in Eq. (3), we do not always obtain an adequate mapping from V_1^+ to V_2. To overcome this difficulty, the cost matrix in Eq. (2) are redefined to take account of the structural information as the matrix C^* with elements

$$c^*_{ij} = c(local(v_i) \to local(v_j)),$$

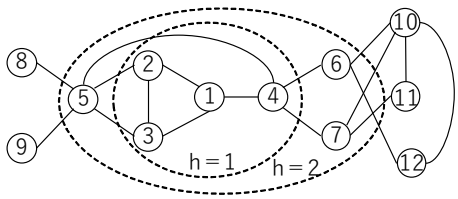

Figure 2: Limited-size subgraphs induced by vertices within h steps from vertex v_1.

Algorithm 1: Approximate_Edit_Distance.

Data: graphs g_1 and g_2, and h
Result: approximate graph edit distance $\hat{d}$
1 **while** $|V(g_1)| < |V(g_2)|$ **do**
2 $\quad V(g_1) \leftarrow V(g_1) \cup \{\varepsilon\}$;
3 **for** $(v_i, v_j) \in V(g_1) \times V(g_2)$ **do**
4 $\quad c^*_{ij} \leftarrow d(g^i_h, g^j_h)$;
5 $\hat{\varphi} \leftarrow LSAP(C^*)$;
6 $\hat{d} \leftarrow dist(g_1, g_2, \hat{\varphi})$;
7 **return** $\hat{d}$;

where $local(v_i)$ is the local structure around a vertex v_i, and $c(local(v_i) \rightarrow local(v_j))$ is the cost of editing $local(v_i)$ to $local(v_j)$. Recently, various methods within this framework have been developed by representing local structures as stars (Zeng et al., 2009), walks (Gaüzère et al., 2014), or limited-size subgraphs (Carletti et al., 2015). This enables a more accurate mapping between sets of vertices than that of Eq. (5).

Carletti et al. (2015) introduced $local(v_i)$, a limited-size subgraph induced by vertices reachable within h steps from vertex v_1, as shown in Fig. 2. The subgraph $local(v_i)$ is denoted by g^i_h, and c^*_{ij} is the exact edit distance between g^i_h and g^j_h, that is, $c^*_{ij} = d(g^i_h, g^j_h)$. $d(g^i_h, g^j_h)$ is tractable for sufficiently small h, although applying the problem to large graphs is intractable, because the problem of computing the graph edit distance is NP-complete. Additionally, because the structural information contained in limited-size subgraphs is greater than that of walks and stars, the graph edit distance based on limited-size subgraphs provides an accurate approximate graph edit distance.

Algorithm 1 shows the pseudo-code for computing an approximate graph edit distance between graphs g_1 and g_2. In Lines 1–2, the numbers of vertices in g_1 and g_2 are equalized. For every pair of vertices in $V(g_1) \times V(g_2)$, the exact edit distance between limited-size subgraphs g^i_h and g^j_h is measured and set as the (i, j)-th element in the cost matrix C^*. *LSAP* in Line 5 returns the optimum mapping according to the optimal bipartite graph matching. Finally, in Line 7, Algorithm 1 returns the approximate graph edit distance between graphs g_1 and g_2.

This algorithm has a drawback in terms of computational efficiency, because it computes the exact edit distance multiple times. To overcome this, we propose a novel method for computing the approximate graph edit distance more efficiently by comparing structural information in g^i_h and g^j_h with a computation time that is proportional to $|V(g^i_h)|$ and $|V(g^j_h)|$.

4 APPROXIMATE GRAPH EDIT DISTANCE BASED ON RELABELING GRAPHS

Given a graph $g^{(h)} = (V, E, \Sigma, \ell^{(h)})$, all labels of vertices in $g^{(h)}$ are updated to obtain another graph $g^{(h+1)} = (V, E, \Sigma', \ell^{(h+1)})$. We call this operation "relabeling," and define it as $\ell^{(h+1)}(v) = r(v, N(v), \ell^{(h)})$. Representative methods based on relabeling include the Weisfeiler–Lehman Subtree Kernel (WLSK) (Shervashidze et al., 2011), Neighborhood Hash Kernel (NHK) (Hido and Kashima, 2009), and Hadamard Code Kernel (HCK) (Kataoka and Inokuchi, 2016). The vertex labels of WLSK are represented as strings and the relabeling of vertex v is defined as a string concatenation of the labels of $N(v)$. In NHK, the vertex labels are represented as fixed-length bit strings and relabeling v is defined in terms of logical operations such as XOR on the labels of $N(v)$. The labels of HCK are based on the Hadamard code, which is used in spread spectrum-based communication technologies, and relabeling v is defined as a summation on the labels of $N(v)$.

Figure 3 shows an example of the framework based on graph relabeling. Let $g^{(0)}$ be the original graph whose vertices have labels a, b, and c. Each of the labels is relabeled to obtain $g^{(1)}$. Although the actual calculation depends on the method of relabeling (e.g., NHK, WLSK, or HCK), the relabeling of v is commonly applied using v, $N(v)$, and $\ell^{(0)}(v)$. In the center of Figure 3, $\ell^{(0)}(v_1) = b$ is relabeled into d using adjacent vertices v_2 and v_4. Therefore, $\ell^{(1)}(v_1) = d$ represents the characteristics of $st(v_1, 1)$, where $st(v, 1)$ is a tree of height 1 whose root and leaves are v and $N(v)$, respectively. The labels of v_1 and v_3 in $g^{(1)}$ are identical because $st(v_1, 1) = st(v_3, 1)$ in $g^{(0)}$.

Given a graph g and its vertex v, an unfolded subtree $st(v, h)$ of height h is defined recursively from the root of the tree:

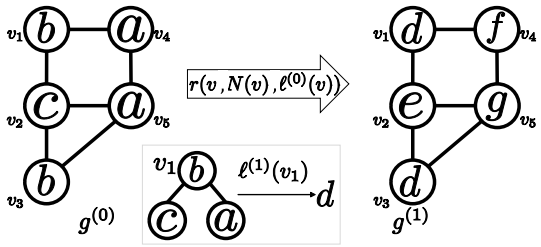

Figure 3: Example of relabeling ($g^{(0)} \to g^{(1)}$).

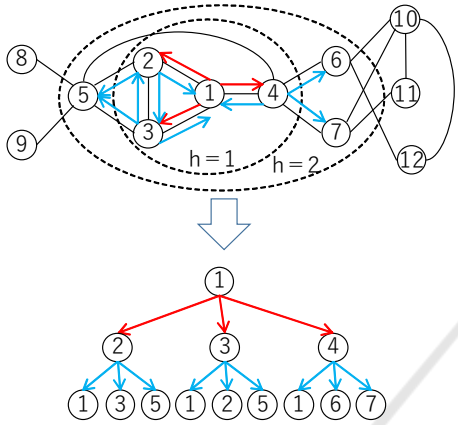

Figure 4: Derivation of an unfolded subtree of height 2.

- the root of $st(v,h)$ is the vertex v in g,
- each node in $st(v,h)$, except for leaves, has children that are $N(u)$ of the vertex u corresponding to the node, and
- the height of $st(v,h)$ is h.

Figure 4 shows an example of an unfolded subtree $st(v_1,2)$ derived from the graph shown in Figure 2. A vertex in the graph may appear in the unfolded subtree (such as v_1, v_2, and v_3), because multiple paths reach to the vertices from the root; this causes the size of the tree to grow exponentially as h increases.

The Label Aggregate Kernel (LAK) (Kataoka and Inokuchi, 2016) is another method based on this framework. Next, we present a concrete definition of the relabeling operation in LAK. In LAK, $\boldsymbol{\ell}_L^{(0)}(v)$ is a vector in $|\Sigma|$-dimensional space. If a vertex in a graph has a label σ_i from the set $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_{|\Sigma|}\}$, the i-th element in the vector is 1 and the other elements are 0. In LAK, $\boldsymbol{\ell}_L^{(h)}(v)$ is defined as

$$\boldsymbol{\ell}_L^{(h)}(v) = \boldsymbol{\ell}_L^{(h-1)}(v) + \sum_{u \in N(v)} \boldsymbol{\ell}_L^{(h-1)}(u).$$

The i-th element in $\boldsymbol{\ell}_L^{(h)}(v)$ equals the frequency of occurrence of σ_i in $st(v,h)$. Label $\boldsymbol{\ell}_L^{(h)}(v)$, obtained by iteratively relabeling h times, has a distribution of labels that is reachable within h steps from v. Therefore,

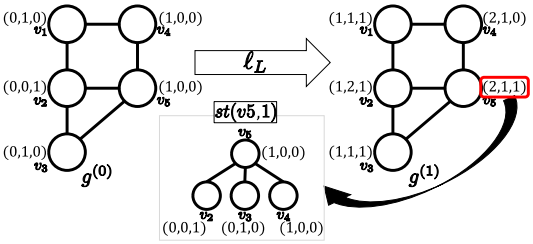

Figure 5: Example of relabeling in LAK.

$\boldsymbol{\ell}_L^{(h)}(v)$ represents the characteristics of $st(v,h)$. In addition, although the size of $st(v,h)$ increases exponentially as h increases, the size of $\boldsymbol{\ell}_L^{(h)}(v)$ remains $|\Sigma|$. Therefore, retaining $\boldsymbol{\ell}_L^{(h)}(v)$ in memory is tractable, although retaining $st(v,h)$ is intractable.

We show an example of relabeling in LAK in Figure 5, assuming that $|\Sigma| = 3$ and relabeling is applied only once. Consider the graph $g^{(0)}$, whose vertices have labels $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$. We next relabel the graphs to obtain $g^{(1)}$. The label of vertex v in $g^{(1)}$ represents the distribution of labels contained in $st(v,1)$. For instance, the label of v_5 in $g^{(1)}$ is $\boldsymbol{\ell}_L^{(1)}(v_5) = (2,1,1)$, which indicates that there are two vertices labeled $(1,0,0)$, one vertex labeled $(0,1,0)$, and one vertex labeled $(0,0,1)$. This distribution is equivalent to that of the labels contained in $st(v_5,1)$.

By relabeling the graphs h times, each vertex v has $h+1$ $|\Sigma|$-dimensional vectors $\boldsymbol{\ell}_L^{(0)}(v), \boldsymbol{\ell}_L^{(1)}(v), \cdots, \boldsymbol{\ell}_L^{(h)}(v)$. We concatenate these vectors to obtain an $(h+1) \times |\Sigma|$-dimensional vector as

$$\boldsymbol{\ell}_L(v) = (\overbrace{\ell_1^{(0)}, \cdots, \ell_{|\Sigma|}^{(0)}}^{\boldsymbol{\ell}_L^{(0)}(v)}, \overbrace{\ell_1^{(1)}, \cdots, \ell_{|\Sigma|}^{(1)}}^{\boldsymbol{\ell}_L^{(1)}(v)}, \cdots, \overbrace{\ell_1^{(h)}, \cdots, \ell_{|\Sigma|}^{(h)}}^{\boldsymbol{\ell}_L^{(h)}(v)}). \quad (6)$$

Given two graphs g_1 and g_2, if g_h^i is isomorphic to g_h^j, where $v_i \in V(g_1)$ and $v_j \in V(g_2)$, then $\boldsymbol{\ell}_L(v_i)$ is the same as $\boldsymbol{\ell}_L(v_j)$.

Based on this observation, we propose a novel method for computing the difference between local structures around v_i and v_j. Given two concatenated vectors $\boldsymbol{\ell}_L(v_i)$ and $\boldsymbol{\ell}_L(v_j)$, the distance between $\boldsymbol{\ell}_L(v_i)$ and $\boldsymbol{\ell}_L(v_j)$ is defined as

$$c_{ij}^* = \left\| \boldsymbol{\ell}_L(v_i) - \boldsymbol{\ell}_L(v_j) \right\|^2 \quad (7)$$

$$= \sum_{t=0}^{h} \left\| \boldsymbol{\ell}_L^{(t)}(v_i) - \boldsymbol{\ell}_L^{(t)}(v_j) \right\|^2. \quad (8)$$

Because Eq. (7) is the square of the Euclidean distance between $\boldsymbol{\ell}_L(v_i)$ and $\boldsymbol{\ell}_L(v_j)$, the four axioms about this metric hold (non-negativity, identity of indiscernibles, symmetry, and triangle inequality). However, we need to discuss the identity of indis-

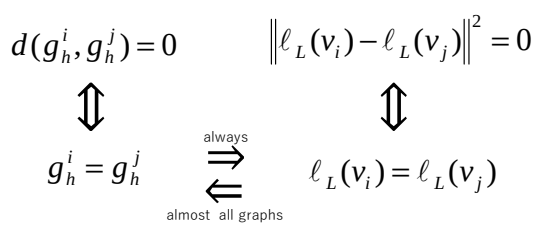

Figure 6: Identity of indiscernibles.

Algorithm 2: Proposed_Method.

Data: graphs g_1 and g_2, and H
Result: approximate graph edit distance $\hat{d}$

1 **while** $|V(g_1)| < |V(g_2)|$ **do**
2 $\quad V(g_1) \leftarrow V(g_1) \cup \{\varepsilon\}$;
3 $C^* \leftarrow 0$;
4 **for** $h \in [0, H]$ **do**
5 $\quad$ **for** $(v_i, v_j) \in V(g_1) \times V(g_2)$ **do**
6 $\quad\quad c^*_{ij} \leftarrow c^*_{ij} + \gamma^{2h} \left\| \boldsymbol{\ell}_L^{(h)}(v_i) - \boldsymbol{\ell}_L^{(h)}(v_j) \right\|^2$;
7 $\quad g_1 \leftarrow (V(g_1), E(g_1), \mathcal{Z}^{|\Sigma|}, \ell^{(h+1)})$;
8 $\quad g_2 \leftarrow (V(g_2), E(g_2), \mathcal{Z}^{|\Sigma|}, \ell^{(h+1)})$;
9 $\quad \hat{\varphi} \leftarrow LSAP(C^*)$;
10 $\quad \hat{d}_h \leftarrow dist(g_1, g_2, \hat{\varphi})$;
11 **return** $\min_{h \in [0,H]} \hat{d}_h$;

cernibles in more detail. As shown in Fig. 6, the identity of indiscernibles for the Euclidean distance implies that $\boldsymbol{\ell}_L(v_i) = \boldsymbol{\ell}_L(v_j)$ if and only if $\|\boldsymbol{\ell}_L(v_i) - \boldsymbol{\ell}_L(v_j)\|^2 = 0$. The above framework for relabeling graphs is based on the 1-dimensional Weisfeiler–Lehman algorithm, which checks the isomorphism between two graphs. This algorithm is known to be a valid isomorphism test for *almost* all graphs (see (Cai et al., 1992) for examples of graphs that cannot be distinguished by the algorithm) (Shervashidze et al., 2011). Therefore, if $g_h^i = g_h^j$, $\|\boldsymbol{\ell}_L(v_i) - \boldsymbol{\ell}_L(v_j)\|^2 = 0$ always holds. In contrast, its converse holds for almost all graphs.

To control the effect of steps from a central vertex of the local structure, we add to Eq. (6) a coefficient that changes exponentially as h increases:

$$\boldsymbol{\ell}_L(v, \gamma) = (\overbrace{\ell_1^{(0)}, \cdots, \ell_{|\Sigma|}^{(0)}}^{\boldsymbol{\ell}_L^{(0)}(v)}, \cdots, \overbrace{\gamma^h \ell_1^{(h)}, \cdots, \gamma^h \ell_{|\Sigma|}^{(h)}}^{\gamma^h \boldsymbol{\ell}_L^{(h)}(v)}). \quad (9)$$

Algorithm 2 shows the pseudo-code of the proposed method for computing the approximate graph edit distance between graphs g_1 and g_2. In Lines 1–2, the numbers of vertices in g_1 and g_2 are equalized. This is repeated at most b times. For each pair of vertices in $V(g_1) \times V(g_2)$, the Euclidean distance between two vectors $\boldsymbol{\ell}_L^{(h)}(v_i)$ and $\boldsymbol{\ell}_L^{(h)}(v_j)$ is measured and set as the (i, j)-th element in a square matrix C^* in Line 6. In Lines 7–8, using the set of nonnegative integers $\mathcal{Z}$, $g_1^{(h)}$ and $g_2^{(h)}$ are relabeled to obtain $g_1^{(h+1)}$ and $g_2^{(h+1)}$, respectively. In Line 9, LSAP returns the mapping $\hat{\varphi}$ from $V(g_1)$ to $V(g_2)$ according to the optimal bipartite graph matching. The processes in Lines 5–10 are repeated $H+1$ times. Finally, the algorithm returns the minimum approximate graph edit distance among $\hat{d}_h$. This algorithm runs in $O(H(|\Sigma|b^2 + |\Sigma|b\bar{d} + b^3))$ time, because the computational complexities of Lines 6, 7, 9, and 10 are $O(|\Sigma|)$, $O(|\Sigma|b\bar{d})$, $O(b^3)$, and $O(b^2)$, respectively. Because $\bar{d}$ is bounded by b, the computational complexity of Algorithm 2 becomes $O(b^2 H(|\Sigma| + b))$. If we compute Eq. (7) in a straightforward manner, we require $O(|\Sigma|H)$ memory for each vertex of graphs g_1 and g_2. However, when we compute Lines 6, 7, and 8 in Algorithm 2, we do not require $\boldsymbol{\ell}_L^{(\tau)}(v)$ for $\tau < h$. Therefore, Algorithm 2 requires $O(|\Sigma|b + b\bar{d})$ memory.

The notable difference between Algorithms 1 and 2 is that the former requires the exact graph edit distance to be computed, which is known to be an NP-complete problem. Although the computation is conducted for small graphs, Algorithm 1 runs the computation b^2 times, which entails a significant computational cost. In contrast, the proposed method does not require the exact graph edit distance to be computed, replacing this with computations of the Euclidean distance. The complexity of this operation is independent of the size of the graphs, which enables us to use LSAP multiple times in our proposed method. The final output of the proposed method is selected from $\hat{d}_h$ for $0 \leq h \leq H$, with our method returning the most accurate graph edit distance.

In the proposed method, we used LAK to characterize the local structure around each vertex. If we use WLSK, HCK, or NHK, the difference between g_h^i and g_h^j is represented as only a binary value, rather than in a quantitative form. The binary value represents whether g_h^i and g_h^j are isomorphic or not. This is why we use LAK in our proposed method.

5 EXPERIMENTAL EVALUATION

The proposed method was implemented in Java. All experiments were conducted on an Intel Xeon E5-2609 2.50 GHz computer with 32 GB memory running Microsoft Windows 7. We used two real-world datasets. The first dataset, MUTAG (Debnath et al., 1991), contains information on 188 chemical compounds and their class labels. The class labels are binary values that indicate the mutagenicity of chem-

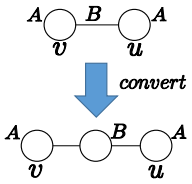

Figure 7: Conversion of a graph.

Table 1: Summary of evaluation datasets.

	MUTAG	ENZYMES
Number of graphs $\|D\|$	188	600
Maximum graph size	84	126
Average graph size	53.9	32.6
Number of labels $\|\Sigma\|$	12	3
Number of classes (class distribution)	2 (126,63)	6 (100,100,100,) 100,100,100)
Avg. degree of vertices	2.1	3.8

ical compounds. Each chemical compound is represented as an undirected graph where each vertex, edge, vertex label, and edge label corresponds to an atom, chemical bond, atom type, and bond type, respectively. Because we assume that only the vertices in the graphs have labels, the chemical graphs are converted using a previous method (Hido and Kashima, 2009), that is, an edge labeled with ℓ that is adjacent to vertices v and u in a chemical graph is replaced with a vertex labeled with ℓ that is adjacent to v and u with unlabeled edges, as shown in Figure 7. The second dataset, ENZYMES, contains information on 600 proteins and their class labels. These are one of six labels denoting the six EC top-level classes (Schomburg et al., 2004). Table 1 summarizes the datasets.

5.1 Computational Efficiency

Figures 8 and 9 show the average computation time required to compute the approximate graph edit distance for each pair of graphs in the MUTAG and ENZYMES datasets for various H, respectively. These figures do not contain results of the exact graph edit distance, because we could not compute the exact graph edit distance between most of the graphs with more vertices than 30. The horizontal axes are logarithmic. To plot the computation times of both methods for $H = 0$, note that the horizontal axes refer to $H + 1$ rather than H. The result of the proposed method for $H = 0$ is almost the same as that of the conventional method for $H = 0$, because they use the same local structures around vertices. As H increases, the computation time of the conventional method aforementioned in Algorithm 1 (Carletti et al., 2015) increases dramatically. We could compute up to $H = 2$ within a total computation time of 3 hours,

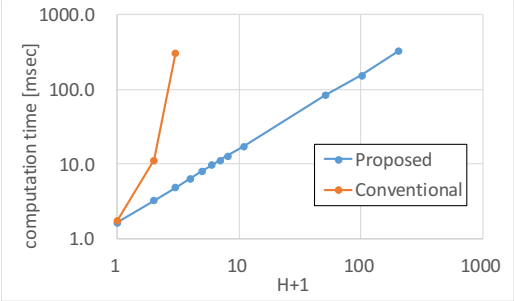

Figure 8: Average computation time for various H (MUTAG).

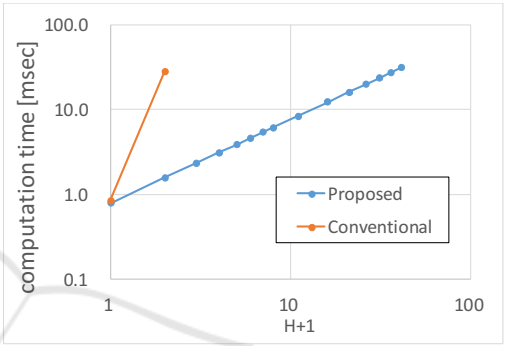

Figure 9: Average computation time for various H (ENZYMES).

with more than 95% of the computation time in the conventional method taken up by computing the exact graph edit distance between local structures. Therefore, the conventional method can only take account of small local structures around vertices to compute the approximate graph edit distance. In contrast, the computation time of the proposed method is proportional to H, as discussed in the final part of Section 4. As the proposed method does not require the exact graph edit distance to be computed, we obtained results for up to $H = 200$ within a total computation time of 3 hours, which indicates that the proposed method takes account of large local structures around vertices. Although the proposed method computes LSAP multiple times, the computational complexity of LSAP is $O(b^3)$, which is sufficiently tractable for the size of the graphs in the experimental datasets. From these results, we can confirm that the proposed method is much faster than the conventional method for computing approximate graph edit distances.

5.2 Accuracy of the Graph Edit Distance

Figures 10 and 11 show the average approximate graph edit distance measured by the proposed method

for each pair of graphs in the MUTAG and ENZYMES datasets, respectively. The approximate graph edit distance measured by the proposed method is not less than the exact graph edit distance, because the exact graph edit distance is defined as the "minimum length" of the sequence of edit operations needed to transform g_1 into g_2 and the approximate graph edit distance measured by the proposed method is computed by Eq. (3). As H increases, the average approximate graph edit distance decreases in both datasets, because large local structures around the vertices are used to map the vertices in one graph to vertices in another graph. In addition, by tuning γ, we can obtain more accurate edit distances than with $\gamma = 1$. In Figure 12, each point $(\hat{d}_c, \hat{d}_p)$ shows the approximate graph edit distances $\hat{d}_c$ and $\hat{d}_p$ measured by the conventional and proposed methods, respectively, for a pair of graphs in the MUTAG dataset. There are many points above the red line, which suggests that most of the approximate graph edit distances measured by the proposed method are more accurate than those given by the conventional method, because $\hat{d}_c > \hat{d}_p$. For the MUTAG dataset, the average approximate graph edit distances of the conventional and proposed methods are 78.5 and 82.4, respectively. Figure 13 for ENZYMES indicates the same tendency as Figure 12. For the ENZYMES dataset, the average approximate graph edit distances of the conventional and proposed methods are 115.8 and 129.0, respectively. One of the reasons why the proposed method computes accurate approximate graph edit distances is that it very efficiently takes account of large local structures around vertices, which enables the minimum approximate graph edit distance to be selected from among $\hat{d}_0, \hat{d}_1, \cdots, \hat{d}_H$.

In general, there is a trade-off between fast computation and accuracy. However, as shown in the previous subsection and here, the proposed method is much faster and more accurate than the conventional method.

5.3 Application of Graph Edit Distance

To validate the applicability of the graph edit distance, we compared the accuracy of the predictions given by the conventional and proposed methods. The graph classification problem is defined as follows. Given a set of n training examples $D = \{(g_i, y_i)\}$ $(i = 1, 2, \cdots, n)$, where each example is a pair consisting of a labeled graph g_i and the class $y_i \in \{+1, -1\}$ to which it belongs, the objective is to learn a function f that correctly predicts the classes of the test examples. In this experiment, graphs are classified by an SVM using a graph kernel. The graph kernel used in

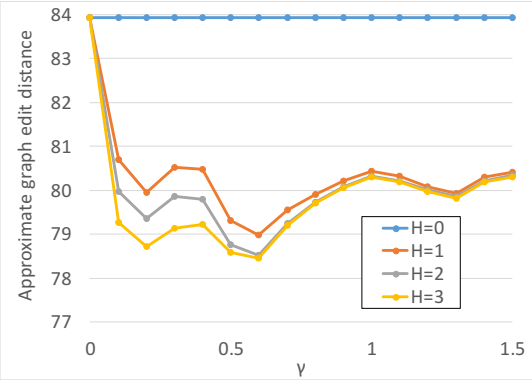

Figure 10: Average approximate graph edit distance for various H and γ (MUTAG).

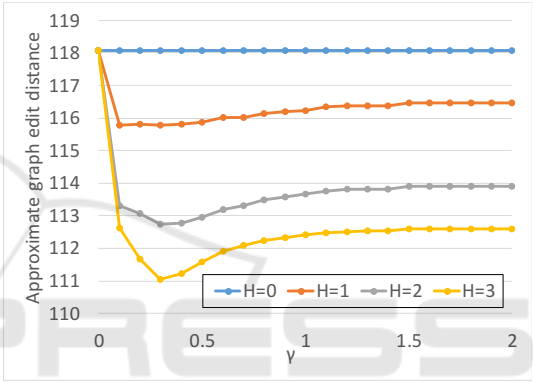

Figure 11: Average approximate graph edit distance for various H and γ (ENZYMES).

this experiment is defined as

$$k(g_1, g_2) = \exp\left(-\frac{\hat{d}(g_1, g_2)^2}{2\sigma^2}\right), \qquad (10)$$

where $\hat{d}(g_1, g_2)$ is the approximate graph edit distance between g_1 and g_2 measured by the conventional and proposed methods[1]. To learn from the kernel matrices generated by the above graph kernel, we used the LIBSVM package with 10-fold cross-validation (Chang and Lin, 2001).

Figures 14 and 15 show the classification accuracy of the conventional and proposed methods, as well as that of WLSK (Shervashidze et al., 2011), for various H using the MUTAG and ENZYME datasets. The maximum accuracy of the proposed method is greater than that of the conventional method, because the proposed method computes accurate approximate graph edit distances, as shown in the previous section. In addition, the maximum accuracy of the proposed

[1]Their experimental results are represented with "Proposed" and "Conventional" in Figs. 14 and 15.

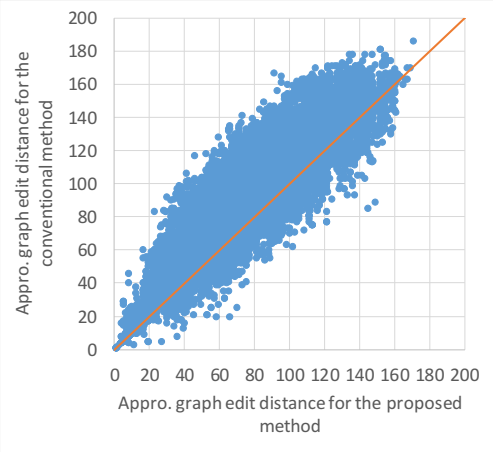

Figure 12: Approximate graph edit distance for $H = 2$ (MUTAG).

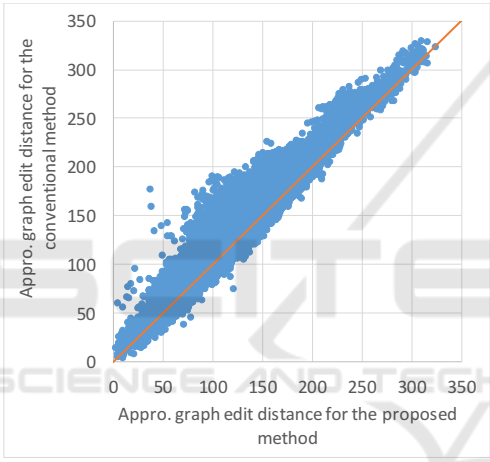

Figure 13: Approximate graph edit distance for $H = 2$ (ENZYMES).

method is comparative with that of WLSK, which is one of the most representative graph kernels.

6 CONCLUSION

As computing the exact graph edit distance is computationally expensive, and may be intractable for large-scale datasets, in this paper, we proposed a method based on graph relabeling that is both faster and more accurate than the conventional approach. We used unfolded subtrees to denote the potential relabeling of local structures around a given vertex. These subtree representations are concatenated as a vector, and the distance between different vectors is used to characterize the distance between the corresponding graphs. This avoids the need for multiple calculations of the exact graph edit distance between local structures.

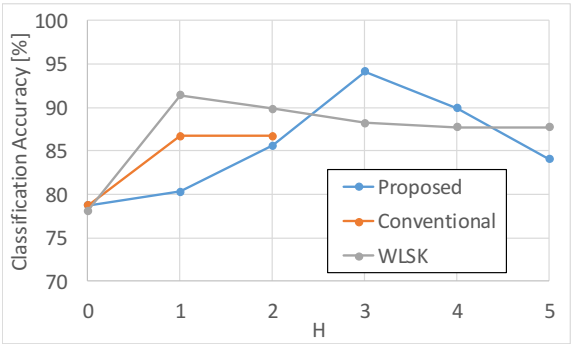

Figure 14: Prediction accuracy for various H (MUTAG).

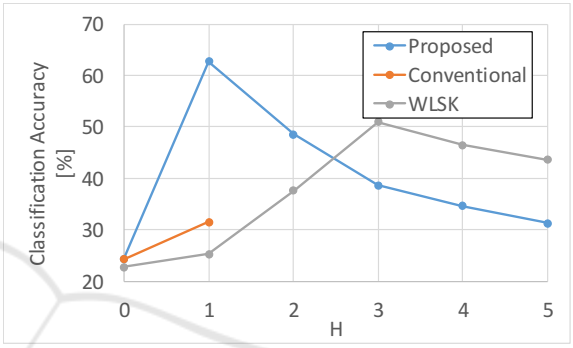

Figure 15: Prediction accuracy for various H (ENZYMES).

Simulation experiments on two real-world chemical datasets were reported. Compared with the conventional technique, the proposed method gave a more accurate approximation of the graph edit distance and is significantly faster on both datasets. This suggested the proposed method could be applicable in the analysis of larger and more complex graph-like datasets.

REFERENCES

Cai, Jin-yi, Fürer, Martin, and Immerman, Neil. 1992. An Optimal Lower Bound on the Number of Variables for Graph Identification. *Combinatorica*, 12(4), 389–410.

Carletti, Vincenzo, Gaüzère, Benoit, Brun, Luc, and Vento, Mario. 2015. Approximate Graph Edit Distance Computation Combining Bipartite Matching and Exact Neighborhood Substructure Distance. In *International Workshop on Graph Based Representations in Pattern Recognition* (GbRPR), 188–197.

Chang, Chih-Chung, and Lin, Chih-Jen. 2001. LIBSVM: A library for Support Vector Machines. Available online at http://www.csie.ntu.edu.tw/cjlin/libsvm.

Choi, Yeonjoo, and Kim, Gyeonghwan. 2010. Graph-based Fingerprint Classification using Orientation Field in Core Area. *IEICE Electronic Express*, 7(17), 1303–1309.

Debnath, Asim Kumar, Lopez de Compadre, Rosa L., Debnath, Gargi, Shusterman, Alan J., and Hansch, Cor-

win. 1991. Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with Molecular Orbital Energies and Hydrophobicity. *Journal of Medicinal Chemistry*, 34, 786–797.

Gaüzère, Benoit, Bougleux, Sébastien, Riesen, Kaspar, and Brun, Luc. 2014. Approximate Graph Edit Distance Guided by Bipartite Matching of Bags of Walks. In *Proc. of International Workshop on Structural and Syntactic Pattern Recognition* (SSPR), 73–82.

Hart, Peter E., Nilsson, Nils J., and Raphael, Bertram. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Journal of IEEE Transaction on Systems Science and Cybernetics*, 4(2), 100–107.

Hido, Shohei and Kashima, Hisashi. 2009. A Linear-Time Graph Kernel. In *Proc. of International Conference on Data Mining* (ICDM), 179–188.

Kashima, Hisashi, Tsuda, Koji, and Inokuchi, Akihiro. 2003. Marginalized Kernels Between Labeled Graphs. In *Proc. of International Conference on Machine Learning* (ICML), 321–328.

Kataoka, Tetsuya and Inokuchi, Akihiro. 2016. Hadamard Code Graph Kernels for Classifying Graphs. In *Proc. of International Conference on Pattern Recognition Applications and Methods* (ICPRAM), 24–32.

Kinable, Joris, and Kostakis, Orestis. 2011. Malware Classification based on Call Graph Clustering. *Journal in Computer Virology*, 7(4), 233–245.

Koopmans, Tjalling C. and Beckmann, Martin. 1957. Assignment Problems and the Location of Economic Activities. *Econometrica*, 25(1), 53–76.

Riesen, Kaspar and Bunkle, Horst. 2009. Approximate Graph Edit Distance Computation by Means of Bipartite Graph Matching. *Image Vision Computing*, 27(7), 950–959.

Riesen, Kaspar. 2015. *Structural Pattern Recognition with Graph Edit Distance: Approximation Algorithms and Applications*. Advances in Computer Vision and Pattern Recognition, Springer.

Robles-Kelly, Antonio, and Hancock, Edwin R. 2003. Edit Distance From Graph Spectra. In *Proc. of International Conference on Computer Vision* (ICCV), 234–241.

Schomburg, Ida, Chang, Antje, Ebeling, Christian, Gremse, Marion, Heldt, Christian, Huhn, Gregor, and Schomburg, Dietmar. 2004. BRENDA, the Enzyme Database: Updates and Major New Developments. *Nucleic Acids Research*, 32D, 431–433.

Shervashidze, Nino, Schweitzer, Pascal, Jan van Leeuwen, Erik, Mehlhorn, Kurt, and Borgwardt, Karsten M.. 2011. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research* (JMLR), 2539–2561.

Wang, Peng, Eglin, Véronique, Garcia, Christophe, Largeron, Christine, Lladós, Josep, and Fornés, Alicia. 2014. A Coarse-to-Fine Word Spotting Approach for Historical Handwritten Documents Based on Graph Embedding and Graph Edit Distance. In *Proc of International Conference on Pattern Recognition* (ICPR), 3074–3079.

Yan, Xifeng, Yu, Philip S., and Han, Jiawei. 2005 Substructure Similarity Search in Graph Databases. In *Proc. of the ACM SIGMOD International Conference on Management of Data* (SIGMOD), 766–777.

Zeng, Zhiping, Tung, Anthony K. H. Tung, Wang, Jianyong, Feng, Jianhua, and Zhou, Lizhu. 2009. Comparing Stars: On Approximating Graph Edit Distance. In *Proc. of International Conference on Very Large Databases* (PVLDB), 2(1), 25–36.

APPENDIX

The Label Aggregate Kernel (LAK) was designed to measure the similarity between two graphs g_1 and g_2 to enable the application of a Support Vector Machine (SVM). The kernel is defined as

$$k(g_1,g_2)=\sum_{t=0}^{h}\sum_{v_i\in V(g_1)}\sum_{v_j\in V(g_2)}\delta(\boldsymbol{\ell}_L^{(t)}(v_i),\boldsymbol{\ell}_L^{(t)}(v_j)),$$

where δ is the Kronecker delta. In contrast, in this paper, LAK has been used to measure the approximate graph edit distance corresponding to the dissimilarity between two graphs g_1 and g_2.