Mapping Distance Graph Kernels using Bipartite Matching

Tetsuya Kataoka, Eimi Shiotsuki and Akihiro Inokuchi
School of Science and Technology, Kwansei Gakuin University, 2-1 Gakuen, Sanda, Hyogo, Japan
{*tkataoka, inokuchi*}*@kwansei.ac.jp*

Keywords: Machine Learning, Graph Kernel, Graph Mining, Graph Classification.

Abstract: The objective of graph classification is to classify graphs of similar structures into the same class. This problem is of key importance in areas such as cheminformatics and bioinformatics. Support Vector Machines can efficiently classify graphs if graph kernels are used instead of feature vectors. In this paper, we propose two novel and efficient graph kernels called Mapping Distance Kernel with Stars (MDKS) and Mapping Distance Kernel with Vectors (MDKV). MDKS approximately measures the graph edit distance using star structures of height one. The method runs in $O(\upsilon^3)$, where υ is the maximum number of vertices in the graphs. However, when the height of the star structures is increased to avoid structural information loss, this graph kernel is no longer efficient. Hence, MDKV represents star structures of height greater than one as vectors and sums their Euclidean distances. It runs in $O(h(\upsilon^3 + |\Sigma|\upsilon^2))$, where Σ is a set of vertex labels and graphs are iteratively relabeled h times. We verify the computational efficiency of the proposed graph kernels on artificially generated datasets. Further, results on three real-world datasets show that the classification accuracy of the proposed graph kernels is higher than three conventional graph kernel methods.

1 INTRODUCTION

A graph is one of the most natural data structures for representing structured data. For instance, a chemical compound can be represented as a graph, where each vertex corresponds to an atom, each edge corresponds to a bond between two atoms therein, and the label of the vertex corresponds to an atom type. With the recent improvement in system throughput, the need for the analysis of a large number of graphs have risen, and the topic of graph mining has raised great interest because knowledge discovery from 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 an analysis of their structural information. In bioinformatics, the prediction of protein-protein interactions is beneficial for drug discovery. One of methods for this application is graph classification.

According to the principle of Johnson and Maggiora, structurally similar chemical compounds have common properties. Virtual screening methods in cheminformatics assume that chemical compounds in a database that are structurally similar to a query have the same physiological activities. Therefore, the objective of the graph classification problem is to classify graphs of similar structures into the same class.

Kernel methods such as Support Vector Machines (SVMs) are becoming increasingly popular because of their high performance (Bernhard, et. al, 1992). In an SVM, a hyperplane for classifying samples is computed from the inner products of the samples. An inner product between two samples has a high value if the two samples are similar. In general, it is hard to represent graphs as feature vectors without losing some of their structural information. However, the application of SVM to graphs becomes possible by replacing the inner products of all pairs of vectorized graphs with specifically designed graph kernels $k(g_i, g_j)$. Furthermore, most methods using graph kernels are efficient because they deliberately avoid the explicit generation of feature vectors. The performance of a graph kernel is evaluated in terms of computational complexity and expressiveness. Here, expressiveness means that the more and larger subgraphs that g_i and g_j contain in common, the higher $k(g_i, g_j)$ will be in value.

There are various frameworks for defining $k(g_i, g_j)$. Two representative frameworks are based on graph edit distance (Neuhaus and Bunke, 2007) and graph relabeling (Kataoka and Inokuchi, 2016). The graph edit distance between graphs g_i and g_j is defined as the minimum length of the sequence of edit operations needed to transform g_i into g_j, where one edit operation includes the insertion or deletion

Kataoka, T., Shiotsuki, E. and Inokuchi, A.
Mapping Distance Graph Kernels using Bipartite Matching.
DOI: 10.5220/0006112900610070
In *Proceedings of the 6th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2017)*, pages 61-70
ISBN: 978-989-758-222-6

of a vertex/edge or substitution of a vertex label. The problem of obtaining the exact graph edit distance between graphs is known to be NP-hard. The other framework iteratively relabels vertex labels in graphs using the adjacent vertices of each vertex, and then measures the similarity between sets of vertices in the graphs using the Jaccard index.

In this paper, we motivate to propose two more accurate graph kernels by incorporating characteristics of the aforementioned frameworks than existing graph kernels. The proposed graph kernels are called the Mapping Distance Kernel with Stars (MDKS) and Mapping Distance Kernel with Vectors (MDKV). One of them is based on a method for approximately measuring the graph edit distance between two graphs. The method runs in $O(\upsilon^3)$ for two graphs, where υ is the maximum number of vertices in the graphs. The kernel sums up the edit distances among star structures of height one obtained from the graphs. When the height of the star structures is increased to avoid loss of structural information, the number of vertices in each star structure exponentially increases, which prevents the efficient computation of this graph kernel. To overcome this difficulty, in the other proposed graph kernel, each of the star structures of height higher than one is represented as a vector, and the graph kernel is computed by summing up the Euclidean distances on these vectors. The graph kernel between two graphs is computed in $O(h(\upsilon^3+|\Sigma|\upsilon^2))$, where Σ is a set of vertex labels and graphs are iteratively relabeled h times.

The rest of this paper is organized as follows. Section 2 formalizes the graph classification problem that this paper tackles and explains the kernel function used in SVM. In Section 3, we propose MDKS and MDKV after we explain two graph kernel frameworks. In Section 4, we verify the computational efficiency of the proposed graph kernels on artificially generated dataset and compare the proposed graph kernels with conventional graph kernels in terms of classification accuracy using real-world datasets. Finally, we conclude the paper in Section 5.

2 PRELIMINARIES

This paper tackles the classification problem of graphs. First, we define some terminologies used for solving 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

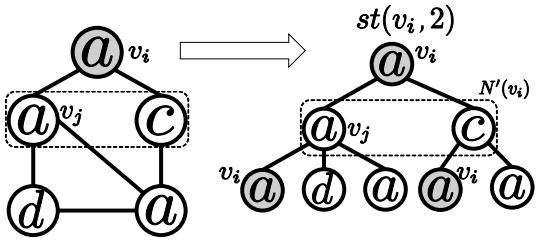

Figure 1: Subtree for v in a graph ($h = 2$).

$V(g)$. 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 $N(v) = \{u \mid (v,u) \in E\}$. Further, $L(N(v)) = \{\ell(u) \mid u \in N(v)\}$ is a multiset of labels adjacent to v. 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 called simple if and only if the path does not have repeating vertices. Paths in this paper are not always simple. Given $v \in V(g)$, $st(v,h)$ is a subtree of height h, where v is the root and u is child of w if u and w are adjacent in g. Here, the height of the subtree is the length of a path from the rooted vertex to a leaf vertex, and $N'(v')$ is a set of children of v' in $st(v,h)$. Figure 1 shows an example of a subtree of height two in a graph. As shown in Fig. 1, when a vertex v_j belongs to $N'(v_i)$ and $h > 1$, v_i also belongs to $N'(v_j)$. That is, v' is a grandchild of v' in $st(v,h)$.

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 classifies the classes of the test examples.

We can classify graphs using SVM and a Gaussian kernel. Given two examples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ as feature vectors, the Gaussian kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is defined as

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right),$$

where σ^2 is a parameter that adjusts the variance. Because it is hard to represent graphs as feature vectors without a loss of their structural information, we design a dissimilarity $d(g_i, g_j)$ between g_i and g_j to replace $||\boldsymbol{x}_i - \boldsymbol{x}_j||$. A kernel function for graphs is called a graph kernel, denoted as $k(g_i, g_j)$ and defined as

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

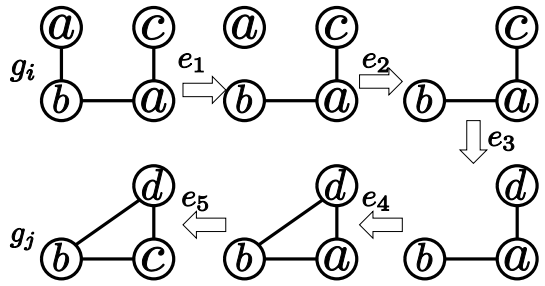

Figure 2: Sequence of edit operations for transforming g_i into g_j.

3 PROPOSED GRAPH KERNELS

The definition of $d(g_i, g_j)$ is vital for the performance of the classification model. There are various frameworks for designing graph kernels. Two representative frameworks among them are based on graph edit distance and graph relabeling. First, we propose a novel graph kernel based on the former framework, and then we propose another novel graph kernel based on both frameworks.

3.1 Graph Kernels based on Graph Edit Distance

Graph edit distance is one of the most representative metrics for defining $d(g_i, g_j)$, and a number of graph kernels based on the graph edit distance have been proposed (Neuhaus and Bunke, 2007). The graph edit distance between graphs g_i and g_j is defined as the minimum length of the sequence of edit operations needed to transform g_i into g_j, where one edit operation includes the insertion or deletion of a vertex/edge and substitution of a vertex label. Although edit distance was originally proposed for measuring the dissimilarities between two strings, the metric was extended to graphs because edit operations were introduced for graphs.

Figure 2 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). The computation needed to obtain the edit distance between g_i and g_j is equivalent to searching for the minimum length of the sequence of edit operations needed to transform g_i into g_j. The method based on the $A^\star$ algorithm is a well-known method for computing the exact graph edit distance (P. Hart, et. al, 1968). However, this method cannot be applied to graphs of large size, because the problem of obtaining the exact graph edit distance between two graphs is known to be NP-hard and consequently, the graph kernels based on the graph edit distance have drawback in terms of computational efficiency. To address this drawback, we propose a graph kernel based on the mapping distance (Zhiping, et. al, 2009) (Riesen and Bunke, 2009), which is the suboptimal graph edit distance between graphs.

Here, we explain the mapping distance between graphs. The distance is one method for approximately measuring the graph edit distance, and this metric is obtained in $O(\upsilon^3)$, where $\upsilon = \max\{|V(g_i)|, |V(g_j)|\}$. To obtain the mapping distance, we use star structures in the graph. A star structure $s(v)$ for v in a graph g is a subtree whose root is v and leaves consist of $N(v)$. That is, $s(v)$ is equivalent to $st(v,1)$. Given a graph g, $|V(g)|$ star structures can be generated from g. The multiset of star structures generated from g is denoted as $S(g) = \{s(v_1), s(v_2), \cdots, s(v_{|V(g)|})\}$. The star edit distance between $s(v_i)$ and $s(v_j)$ is the minimum length of the sequence of edit operations needed to transform $s(v_i)$ into $s(v_j)$ and is denoted as $\lambda(s(v_i), s(v_j))$, where $\lambda(s(v_i), s(v_j))$ is defined as

$$\begin{aligned}\lambda(s(v_i), s(v_j)) = \lambda_1(v_i, v_j) &+ \lambda_2(N(v_i), N(v_j)) \\ &+ \lambda_3(N(v_i), N(v_j)),\end{aligned}$$

where

$$\begin{aligned}\lambda_1(v_i, v_j) &= \delta(\ell(v_i), \ell(v_j)), \\ \lambda_2(N(v_i), N(v_j)) &= \big||N(v_i)| - |N(v_j)|\big|, \text{ and} \\ \lambda_3(N(v_i), N(v_j)) &= \max\{|N(v_i)|, |N(v_j)|\} \\ &\quad -|L(N(v_i)) \cap L(N(v_j))|.\end{aligned}$$

Star edit distance $\lambda_1(v_i, v_j)$ returns 1 if the roots of the star structures have identical labels and 0 otherwise, which is equivalent to a substitution for the labels of roots. Distance $\lambda_2(N(v_i), N(v_j))$ equals the required number of insertions and/or deletions of edges in $s(v_i)$ and $s(v_j)$. Distance $\lambda_3(N(v_i), N(v_j))$ equals the required number of substitutions for labels of leaves in $s(v_i)$ and $s(v_j)$. From the above, $\lambda(s(v_i), s(v_j))$ represents the star edit distance between $s(v_i)$ and $s(v_j)$.

Given two multisets of star structures $S(g_i)$ and $S(g_j)$, the mapping distance between g_i and g_i is denoted as $md_1(g_i, g_j)$ and defined as

$$md_1(g_i, g_j) = \min_P \sum_{s(u) \in S(g_i)} \lambda(s(u), P(s(u))), \qquad (2)$$

where $P : S(g_i) \to S(g_j)$ is a bijective function. The computation of $md_1(g_i, g_j)$ is equal to solving the minimum weight matching on the complete bipartite graph $g' = (V_i, V_j, E')$ such that for every two vertices $(v_i, v_j) \in V_i \times V_j$, there is an edge whose weight is the star edit distance $\lambda(s(v_i), s(v_j))$ between $s(v_i)$ and $s(v_j)$. Given a square matrix in

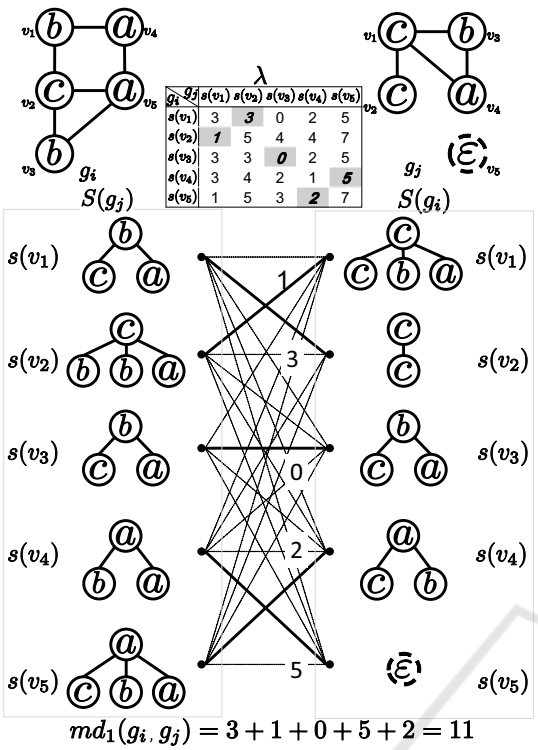

Figure 3: Minimum weight matching to find the mapping distance between g_i and g_j.

which the (i, j)-element represents the star edit distance $\lambda(s(v_i), s(v_j))$, this matching problem is solved by means of the Hungarian algorithm, which runs in $O(\upsilon^3)$ (Kuhn, 1955), where $\upsilon = \max\{|V_i|, |V_j|\}$.

Figure 3 shows an example of mapping $S(g_i)$ to $S(g_j)$ to obtain $md_1(g_i, g_j)$. Given two graphs g_i and g_j, five star structures are generated from g_i and four star structures are generated from g_j. The table between g_i and g_j represents the star edit distance between every pair of star structures in $S(g_i)$ and $S(g_j)$. If $|V(g_i)|$ does not equal $|V(g_j)|$, the matrix that represents star edit distances among star structures is not square and cannot be used as an input for the Hungarian algorithm. In order to obtain a square matrix, a dummy vertex (denoted as v_5 in g_j) whose label is ε is inserted in g_j to equalize the numbers of vertices in g_i and g_j. By applying the Hungarian algorithm, the optimal bipartite graph matching (indicated by solid lines) is output and the final answer $md_1(g_i, g_j) = 3+1+0+5+2 = 11$ is obtained.

Using the mapping distance, we propose a novel graph kernel called MDKS.

MDKS: We adopt the mapping distance defined in Eq. (2) to the graph kernel defined in Eq. (1). Given two graphs g_i and g_j, the graph kernel in MDKS is defined as follows:

Algorithm 1: Mapping_Distance_Kernel1.

Data: a set of graphs D for training and variance σ^2
Result: kernel matrix K

1 **for** $g_i, g_j \in D$ **do**
2 **while** $|V(g_i)| < |V(g_j)|$ **do**
3 $V(g_i) \leftarrow V(g_i) \cup \{dummy\_vertex\}$;
4 **while** $|V(g_i)| > |V(g_j)|$ **do**
5 $V(g_j) \leftarrow V(g_j) \cup \{dummy\_vertex\}$;
6 **for** $(v_a, v_b) \in V(g_i) \times V(g_j)$ **do**
7 $\lambda \leftarrow 0$;
8 **if** $\ell_i(v_a) \neq \ell_j(v_b)$ **then**
9 $\lambda \leftarrow 1$;
10 $\lambda \leftarrow \lambda + ||N(v_a)| - |N(v_b)||$;
11 $\lambda \leftarrow \lambda + \max\{|N(v_a)|, |N(v_b)|\} - |L(N(v_a)) \cap L(N(v_b))|$;
12 $T_{ab} \leftarrow \lambda$;
13 $md_1 \leftarrow Hungarian(T)$;
14 $K_{ij} \leftarrow \exp\left(-\frac{md_1^2}{2\sigma^2}\right)$;
15 **return** K;

$$k_{MDKS}(g_i, g_j) = \exp\left(-\frac{md_1(g_i, g_j)^2}{2\sigma^2}\right). \quad (3)$$

Here, $k_{MDKS}(g_i, g_j)$ is obtained in $O(\upsilon^3)$, which is faster than graph kernels based on the exact graph edit distance.

Algorithm 1 shows the pseudo-code for computing an MDKS kernel matrix for a set of graphs D. In Lines 2 to 5, the numbers of vertices in g_i and g_j are equalized. For each pair of vertices in $V(g_i) \times V(g_j)$, the star edit distance between star structures $s(v_a)$ and $s(v_b)$ is measured and set as the (a, b)-th element in square matrix T, which is given to the Hungarian algorithm. The Hungarian algorithm returns the mapping distance according to the optimal bipartite graph matching in Line 13. In Line 14, Eq. (3) is computed. These procedures are repeated for every pair of graphs in D, and Algorithm 1 finally returns a kernel matrix for D. This algorithm runs in $O(n^2(\upsilon^3 + \overline{d}\upsilon^2))$, where n, υ, and $\overline{d}$ are the number of graphs in D, the maximum number of vertices in the graphs, and the average degree of the vertices, respectively. Because $\overline{d}$ is bounded by υ, the computational complexity becomes $O(n^2\upsilon^3)$.

MDKS has a drawback in terms of graph expressiveness. The height of the subtrees between which we measure the mapping distance is limited, and this causes a leveling off of the graph expressiveness. It is desirable to measure the edit distance between high order subtrees, as the edit distance between trees with m vertices is computed in $O(m^3)$ (E. D. Demaine,

et. al, 2009). However, because the paths from the root to leaves in a subtree are not simple in the graph from which the subtree is generated, the number of vertices in the subtrees increases exponentially for h, which makes measuring the edit distance between $s(v_i,h)$ and $s(v_j,h)$ intractable. Another way (Carletti, et. al, 2015) is to use subgraphs of g each of which consists of vertices within h step from v_i instead of star structures of G. However, we require exact distances between subgraphs of g_i and subgraphs of g_j, which need computation time. In the next subsection, we propose another efficient graph kernel that compares the characteristics of two subtrees $st(v_i,h)$ and $st(v_j,h)$ for $h > 1$.

3.2 Graph Kernels based on Relabeling

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 the operation a relabel, and it is defined as $\ell^{(h+1)}(v) = r(v,N(v),\ell^{(h)})$. 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) are representative graph kernels based on this relabeling framework. The vertex label of WLSK is represented as a string and a relabel for vertex v is defined as a string concatenation of the labels of $N(v)$. In NHK, the vertex label is represented as a fixed-length bit string and relabeling v is defined as logical operations such as an exclusive-or on the labels of $N(v)$. The label of HCK is based on the Hadamard code, which is used in spread spectrum-based communication technologies, and a relabel for v is defined as a summation on the labels of $N(v)$.

Figure 4 shows an example of the framework based on graph relabeling. Let $g^{(0)}$ be original graph whose vertices have labels a, b, and c. Each of the labels is relabeled to obtain $g^{(1)}$. Although the concrete calculation depends on the method of relabeling such as NHK, WLSK, or HCK , it is common that a relabel for v is applied using v, $N(v)$ and $\ell^{(0)}(v)$. In the center of Fig. 4, $\ell^{(0)}(v_1) = b$ is relabeled into d using adjacent vertices v_2, v_4, and its original label b. Therefore, $\ell^{(1)}(v_1) = d$ represents the characteristics of $st(v_1,1)$. The labels of v_1 and v_3 in $g^{(1)}$ are identical labels because $st(v_1,1) = st(v_3,1)$ in $g^{(0)}$. It is desirable to define labels as identical if and only if both their own labels and labels of adjacent nodes are also identical. However, realizing this condition is hard, and it is important to design a relabel method that satisfies this condition as much as possible.

The label of each vertex is relabeled iteratively.

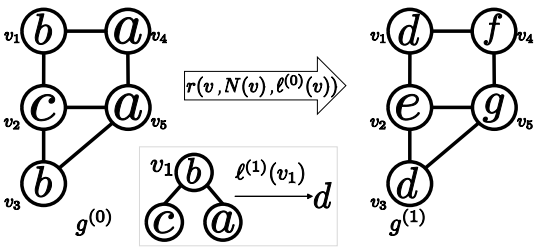

Figure 4: Example of relabeling ($g^{(0)} \to g^{(1)}$).

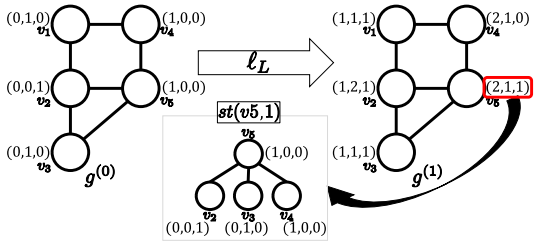

Figure 5: Example of relabeling in LAK.

Labeling $\ell^{(h)}(v)$, obtained by iteratively relabeling h times, has a distribution of labels that is reachable within h steps from v. Therefore, $\ell^{(h)}(v)$ represents the characteristics of $st(v,h)$. Let $\{g^{(0)},g^{(1)},\cdots,g^{(h)}\}$ be a series of graphs obtained by iteratively applying a relabeling h times, where $g^{(0)}$ is an original graph contained in D. Kernel $k(g_i,g_j)$ is defined as

$$k(g_i,g_j) = k(g_i^{(0)},g_j^{(0)}) + k(g_i^{(1)},g_j^{(1)}) + \cdots + k(g_i^{(h)},g_j^{(h)}). \quad (4)$$

The Label Aggregate Kernel (LAK) (Kataoka and Inokuchi, 2016) is another graph kernel based on this framework. Next, we present a concrete definition of a relabel 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 $st(v,h)$. Therefore, $\boldsymbol{\ell}_L^{(h)}(v)$ has information on the distribution of labels in $st(v,h)$, which means that $\boldsymbol{\ell}_L^{(h)}(v)$ is more expressive than the star structure $s(v)$ used to measure the mapping distance.

We show an example of relabeling in LAK in Fig. 5, assuming that $|\Sigma| = 3$ and relabeling is applied only once. Consider graph $g^{(0)}$, whose vertices have labels $(1,0,0)$, $(0,1,0)$, and $(0,0,1)$. We next apply the relabeling to 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 $st(v_5,1)$. In LAK, the kernel function is defined as

$$k(g_i^{(h)}, g_j^{(h)}) = \sum_{(v_i,v_j)\in V(g_i^{(h)})\times V(g_j^{(h)})} \delta(\boldsymbol{\ell}_L^{(h)}(v_i), \boldsymbol{\ell}_L^{(h)}(v_j)).$$

Using the labels used in LAK, we propose another novel graph kernel called MDKV.

MDKV: Given two labels $\boldsymbol{\ell}_L^{(h)}(v_i)$ and $\boldsymbol{\ell}_L^{(h)}(v_j)$, we denote the distance between $\boldsymbol{\ell}_L^{(h)}(v_i)$ and $\boldsymbol{\ell}_L^{(h)}(v_j)$ as $\tau(\boldsymbol{\ell}_L^{(h)}(v_i), \boldsymbol{\ell}_L^{(h)}(v_j))$, defined as

$$\tau(\boldsymbol{\ell}_L^{(h)}(v_i), \boldsymbol{\ell}_L^{(h)}(v_j)) = ||\boldsymbol{\ell}_L^{(h)}(v_i) - \boldsymbol{\ell}_L^{(h)}(v_j))||^2. \quad (5)$$

Given two graphs $g_i^{(h)}$ and $g_j^{(h)}$ relabeled iteratively h times, the distance between $g_i^{(h)}$ and $g_j^{(h)}$ is denoted as $md_2(g_i^{(h)}, g_j^{(h)})$ and defined as

$$md_2(g_i^{(h)}, g_j^{(h)}) = \min_Q \sum_{u\in V(g_i^{(h)})} \tau(\boldsymbol{\ell}_L^{(h)}(u), \boldsymbol{\ell}_L^{(h)}(Q(u))), \quad (6)$$

where $Q : V(g_i^{(h)}) \to V(g_j^{(h)})$ is a bijective function. The computation of $md_2(g_i^{(h)}, g_j^{(h)})$ is also equal to solving the minimum weight matching on a complete bipartite graph and is obtained by means of the Hungarian algorithm. By combining Eqs. (1), (4), and md_2, $k_{MDKV}(g_i, g_j)$ is defined as follows:

$$\begin{aligned} k_{MDKV}(g_i, g_j) &= \sum_{t=0}^{h} k(g_i^{(t)}, g_j^{(t)}) \\ &= \sum_{t=0}^{h} \exp\left(-\frac{md_2(g_i^{(t)}, g_j^{(t)})^2}{2\sigma^2}\right) \end{aligned}$$

The notable difference between MDKS and MDKV is that while the inputs for md_1 are multisets of $st(v,1)$, the ones for md_2 are multisets of vectors obtained from higher order subtrees $st(v,h)$. That is, the computation of MDKV between two graphs contains larger subgraphs in the two graphs. If we directly measure the edit distance between $s(v_i,h)$ and $s(v_j,h)$, the number of vertices in $s(v,h)$ exponentially increases when h increases. In this case, MDKV needs a huge amount of computation time to compute the edit distance. However, by using a vector representation for the vertices and their relabeling, our proposed kernel computes a mapping distance between $s(v_i,h)$ and $s(v_j,h)$ efficiently.

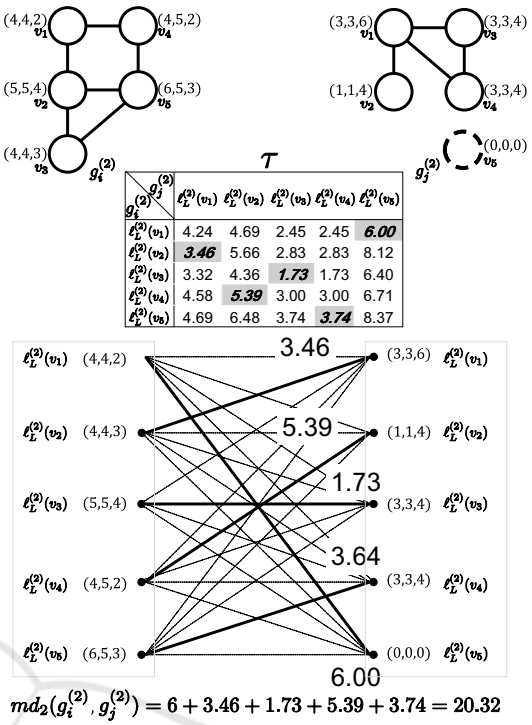

Figure 6: Computation for obtaining $md_2(g_i^{(2)}, g_j^{(2)})$.

Figure 6 shows an example of the procedure to obtain $md_2(g_i^{(2)}, g_j^{(2)})$, assuming that $|\Sigma| = 3$. Graphs $g_i^{(2)}$ and $g_j^{(2)}$ are obtained by relabeling given graphs $g_i^{(0)}$ and $g_j^{(0)}$ iteratively twice. After relabeling, the Euclidean distance between every pair of labels in $g_i^{(2)}$ and $g_j^{(2)}$ are measured. The table between $g_i^{(2)}$ and $g_j^{(2)}$ represents the Euclidean distance of every pair of labels. To equalize the number of vertices in $g_i^{(2)}$ and $g_j^{(2)}$, a dummy vertex whose label is $(0,0,0)$ is inserted to $g_j^{(2)}$. The minimum weight matching is solved by means of the Hungarian algorithm, and the final answer of $md_2(g_i^{(2)}, g_j^{(2)}) = 6 + 3.46 + 1.73 + 5.39 + 3.74 = 20.32$ is obtained.

Algorithm 2 shows the pseudo-code for computing an MDKV kernel matrix for a set of graphs D. In Lines 5 to 8, the numbers of vertices in g_i and g_j are equalized. For each pair of vertices in $V(g_i) \times V(g_j)$, the Euclidean distance between two vectors $\boldsymbol{\ell}_L^{(t)}(v_a)$ and $\boldsymbol{\ell}_L^{(t)}(v_b)$ is measured and set as the (a,b)-th element in T. The Hungarian algorithm returns the mapping distance according to the optimal bipartite graph matching in Line 11. Its output using the Gaussian kernel is added to K_{ij}. In Lines 14 to 16, where $\mathcal{Z}$ is a set of non-negative integers, g is relabeled to obtain $g^{(t+1)}$. These processes in Lines 9 to 15 are repeated

Algorithm 2: Mapping_Distance_Kernel2.

Data: a set of graphs D for training and variance σ^2
Result: kernel matrix K

1 $K \leftarrow 0$;
2 $D^{(0)} \leftarrow D$;
3 **for** $t \in [0,h]$ **do**
4 **for** $g_i, g_j \in D^{(t)}$ **do**
5 **while** $|V(g_i)| < |V(g_j)|$ **do**
6 $V(g_i) \leftarrow V(g_i) \cup \{dummy\_vertex\}$;
7 **while** $|V(g_i)| > |V(g_j)|$ **do**
8 $V(g_j) \leftarrow V(g_j) \cup \{dummy\_vertex\}$;
9 **for** $(v_a, v_b) \in V(g_i) \times V(g_j)$ **do**
10 $T_{ab} \leftarrow \tau(\boldsymbol{\ell}_L^{(t)}(v_a), \boldsymbol{\ell}_L^{(t)}(v_b))$;
11 $md_2 \leftarrow Hungarian(T)$;
12 $K_{ij} \leftarrow K_{ij} + \exp\left(-\frac{md_2^2}{2\sigma^2}\right)$;
13 $D^{(t+1)} \leftarrow \emptyset$;
14 **for** $g \in D^{(t+1)}$ **do**
15 $g^{(t+1)} \leftarrow (V(g), E(g), \mathcal{Z}^{|\Sigma|}, \ell^{(t+1)})$;
16 $D^{(t+1)} \leftarrow D^{(t+1)} \cup \{g^{(t+1)}\}$;
17 **return** K;

$h+1$ times. This algorithm runs in $O(h(n^2\upsilon^3 + n^2|\Sigma|\upsilon^2 + n\overline{d}|\Sigma|\upsilon))$, because the computational complexities of Lines 11, 10, and 15 are $O(\upsilon^3)$, $O(|\Sigma|\upsilon^2)$, and $O(\overline{d}|\Sigma|\upsilon)$, respectively. Because $\overline{d}$ is bounded by υ, the computational complexity of Algorithm 2 becomes $O(hn^2(\upsilon^3 + |\Sigma|\upsilon^2))$.

4 EVALUATION EXPERIMENTS

In this section, we compare the performance of our graph kernels MDKS and MDKV through numerical experiments. We implemented the proposed graph kernels MDKS and MDKV in Java. All experiments were done on an Intel Xeon E5-2609 2.50 GHz computer with 32 GB memory running Microsoft Windows 7. To learn from the kernel matrices generated by the above graph kernels, we used the LIBSVM package[1] using 10-fold cross validation.

4.1 Evaluation using Synthetic Datasets

We examine the computational performance of the proposed graph kernels by means of synthetic graph datasets to confirm that the proposed graph kernels run in $O(n^2(\upsilon^3 + \overline{d}\upsilon))$ and $O(h(n^2\upsilon^3 + n^2|\Sigma|\upsilon^2 + n\overline{d}|\Sigma|\upsilon))$, respectively. We generated graphs with a set of four parameters . Their default values are listed in Table 1.

[1] http://www.csie.ntu.edu.tw/∼cjlin/libsvm/

Table 1: Parameters of the artificial datasets.

Parameters	Defaults
Number of graphs in a dataset	n =100
Average number of vertices in a graph	$\overline{\upsilon}$ =50
Average degrees of a graph	$\overline{d}$ =2
Number of distinct labels in a dataset	$\|\Sigma\|$ =10

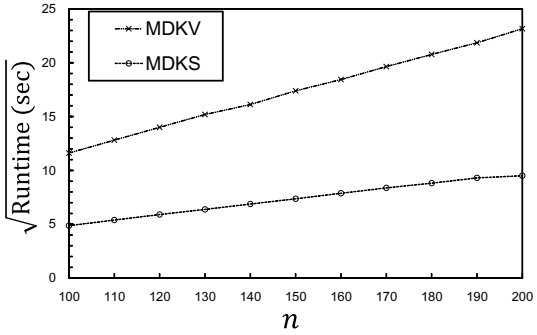

Figure 7: Computation time for various n.

For each dataset, n graphs, each with an average of $\overline{\upsilon}$ vertices, were generated. Two vertices in a graph were connected with probability $\frac{\overline{d}}{\overline{\upsilon}-1}$, and one label from $|\Sigma|$ was assigned to each vertex in the graph. The computation times shown in this subsection are the average of ten trials.

We first varied only n to generate various datasets in which the other parameters were set to their default values. The number of graphs in each dataset was varied from 10 to 100. Figure 7 shows the computation time needed to generate a kernel matrix for each dataset for the proposed graph kernels. In this experiment, h was set to 3. As shown in Fig. 7, the square root of the computation time for the graph kernels is proportional to the number of graphs in the dataset. That is, the computation time is proportional to the square of the number of graphs in the dataset. This is because the proposed graph kernels are computed for two graphs, and the kernels runs for all pairs of graphs in the dataset. We next varied only $\overline{\upsilon}$ to generate various datasets with the other parameters set to their default values. Figure 8 shows the computation time required to generate a kernel matrix in each dataset when the number of vertices in each dataset was varied from 50 to 120. The cubic root of the computation time for the proposed graph kernels is almost proportional to the average number of vertices in the datasets. The parts that need a large amount of computation time in Algorithms 1 and 2 are those that include the Hungarian algorithm. The computa-

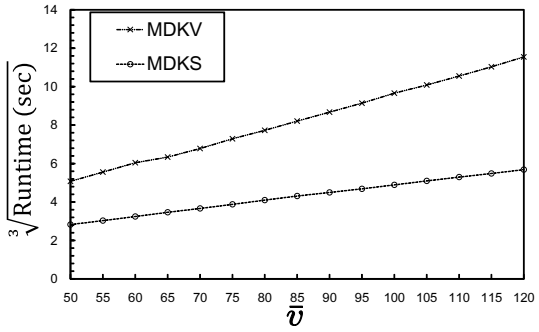

Figure 8: Computation time for various $\overline{v}$.

tion time of the algorithm is proportional to the cube of the number of vertices of the bipartite graph that is given as input. In MDKV, $\tau(\ell_L^{(h)}(v_i), \ell_L^{(h)}(v_j))$ represents the dissimilarity between $s(v_i,h)$ and $s(v_j,h)$. The number of vertices in $s(v,h)$ exponentially increases as h increases. If we directly measure the edit distance between $s(v_i,h)$ and $s(v_j,h)$, MDKV needs a huge amount of computation time. However, by using a vector representation of vertices and the relabeling for the vertices, our proposed MDKV kernel generates the kernel matrix efficiently.

In Figs. 9 and 10, we respectively varied $|\Sigma|$ and $\overline{d}$ to generate various datasets. MDKV needs a computation time that is proportional to $|\Sigma|$ in order to compute the Euclidean distance $\tau(\ell_L^{(h)}(v_i), \ell_L^{(h)}(v_j))$ and relabel the vertices for the $|\Sigma|$ dimensional vectors. The computation times for the propose graph kernels are almost proportional to the average number of degree of each vertex and the number of vertex labels. MDKS needs a computation time that is proportional to $\overline{d}$ in order to measure the edit distance of the substitutions for the leaf labels in the star structures. In contrast, MDKV needs a computation time that is proportional to $\overline{d}$ and $|\Sigma|$ in order to relabel graphs.

Finally, we varied only h for a dataset generated with all other parameters set to their default values. Figure 11 shows the computation time required to generate a kernel matrix in each dataset when h was

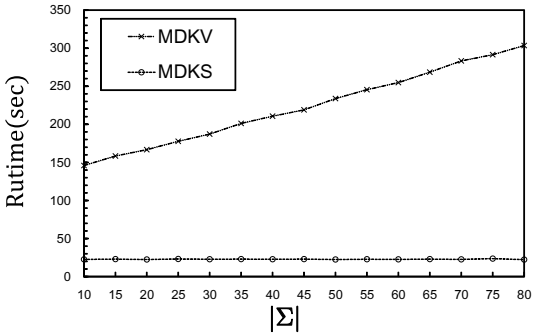

Figure 9: Computation time for various $|\Sigma|$.

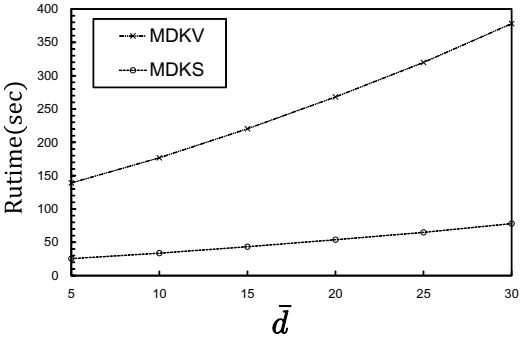

Figure 10: Computation time for various $\overline{d}$.

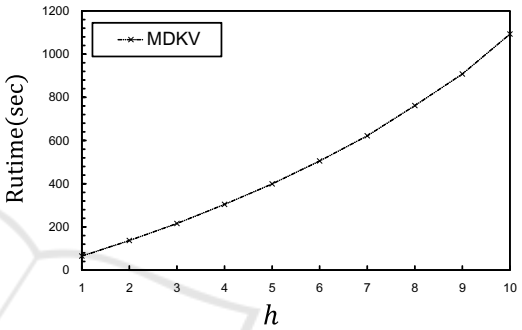

Figure 11: Computation time for various h.

varied from 0 to 15. The computation time is proportional to h.

4.2 Classification Accuracy

We compare the classification accuracies of the proposed graph kernels with those of conventional graph kernels based on the relabeling framework, WLSK, NHK, and HCK, on three real world datasets, MUTAG (Debnath, et. al, 1991), PTC (Helma and Kramer, 2003), and ENZYMES (Schomburg, et. al, 2004). Since HCK theoretically returns the same values as LAK returns, classification accuracies of HCK are equivalent with those of LAK. The first dataset MUTAG consists of 188 chemical compounds and their classes are binary values representing whether each compound is mutagenic. The second dataset PTC consists of 344 chemical compounds, and their classes are binary values representing whether each compound is toxic. Generally, a chemical compound is represented as a graph with labeled edges, which is not a graph that is treated in this paper. We treated the graphs with edge labels in the following two ways: 1) we ignore the edge labels or 2) an edge labeled ℓ that is adjacent to vertices u and v in a graph is converted into a vertex labeled ℓ that is adjacent to u and v, as explained in (Hido and Kashima, 2009). After con-

Table 2: Description of evaluation datasets.

	MUTAG		PTC		ENZYMES
	edge labels	no edge labels	edge labels	no edge labels	
The number of graphs n	188		344		600
The number of classes (class distribution)	2 (125,63)		2 (152,192)		6 (100 per class)
Maximum number of vertices	84	40	325	109	126
Average number of vertices	53.9	26.0	77.5	25.6	32.6
Number of labels	12	8	67	19	3
Average Degree	2.1	2.1	2.7	4.0	3.9
σ_{min}	10		10		10^2
σ_{max}	10^4		10^5		10^4

Table 3: Classification accuracies.

	MUTAG		PTC		ENZYMES
	edge labels	no edge labels	edge labels	no edge labels	
MDKS	92.6%	91.0%	64.2%	63.1%	61.2%
MDKV	**94.1%** ($h=3$)	**93.6%** ($h=2$)	64.0% ($h=0$)	**66.9%** ($h=3$)	**65.3%** ($h=2$)
St-MDKV	91.5% ($h=7,8$)	90.4% ($h=3$)	**64.9%** ($h=1,8$)	64.0% ($h=1$)	63.0% ($h=4$)
NHK	92.6% ($h=3,4$)	90.4% ($h=2$)	60.8% ($h=3,5$)	55.8% ($h=1,2,\cdots,15$)	45.0% ($h=8$)
WLSK	92.0% ($h=3$)	90.4% ($h=1$)	62.8% ($h=15$)	64.2% ($h=10$)	58.5% ($h=1$)
HCK	92.0% ($h=3$)	91.0% ($h=1$)	63.1% ($h=15$)	65.4% ($h=12$)	57.2% ($h=4$)

verting edges in graphs, labels are assigned to only the vertices. The third dataset, ENZYMES consists 600 proteins and their classes represent Enzyme Commission numbers from 1 to 6. Table 2 shows a summary of each dataset.

Before classifying a dataset that does not contain graphs but consists of points in a p-dimensional feature space, we usually normalize the dataset using the mean μ_q and standard deviation σ_q in the q-th feature ($1 \leq q \leq p$). By normalizing the dataset, we often obtain an accurate model for classifying the dataset. Similarly, we apply this procedure in MDKV. To do so, we use the mean $\mu_q^{(t)}$ and standard deviation $\sigma_q^{(t)}$ for the $|\Sigma|$ dimensional vectors to represent the vertex labels for each t ($1 \leq t \leq h$) and q ($1 \leq q \leq |\Sigma|$). Using this procedure, we avoid the exponential increase in the elements in the vectors representing vertex labels when h is increased. We call the MDKV method that uses this procedure St-MDKV.

Table 3 shows the classification accuracies of the proposed and conventional graph kernels . We examined the highest accuracy for each kernel and each dataset varying σ of the Gaussian kernel and h. We varied the σ from σ_{min} to σ_{max} in intervals of 10 (see Table 2) and h from 0 to 15 in intervals of 1. As shown in Table 3, the classification accuracies of the proposed graph kernels outperform those of the conventional graph kernels. The values of h for MDKV are relatively low, which indicates that the elements in the vectors representing vertex labels exponentially increase and distance $\tau(\boldsymbol{\ell}_L^{(h)}(v_i), \boldsymbol{\ell}_L^{(h)}(v_j))$ becomes inadequate when h increases. However, the values of h for St-MDKV are high. By normalizing the vectors representing vertex labels, we adequately measure the (dis)similarity between $s(v_i,h)$ and $s(v_j,h)$, which results in high classification accuracy for various datasets.

5 CONCLUSION

In this paper, we proposed two novel and efficient graph kernels called Mapping Distance Kernel with Stars (MDKS) and Mapping Distance Kernel with Vectors (MDKV). MDKS approximately measures the graph edit distance using star structures of height one. The method runs in $O(\upsilon^3)$, where υ is the maximum number of vertices in the graphs. However, when the height of the star structures is in-

creased to avoid structural information loss, this graph kernel is no longer efficient. Hence, MDKV represents star structures of height greater than one as vectors and sums their Euclidean distances. It runs in $O(h(\upsilon^3+|\Sigma|\upsilon^2))$, where Σ is a set of vertex labels and graphs are iteratively relabeled h times. We verified the computational efficiency of the proposed graph kernels on artificially generated datasets. Further, results on three real-world datasets showed that the classification accuracy of the proposed graph kernels is higher than three conventional graph kernel methods.

REFERENCES

Schölkopf, Bernhard, and Smola, Alexander J.. 2002. *Learning with Kernels*. MIT Press.

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

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

Riesen, Kaspar, and Bunkle, Horst. 2009. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Computing*. 27(7): 950–959.

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

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.

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

Schölkopf, Bernhard, Tsuda, Koji, and Vert, Jean-Philippe. 2004. *Kernel Methods in Computational Biology*. MIT Press.

Kuhn, Harold W.. 1955. The Hungarian Method for the Assignment Problem. *Naval Research Logistics*. 2: 83–97.

Bernhard E. Boser, Isabelle, Guyon, and Vladimir, Vapnik. 1992. A Training Algorithm for Optimal Margin Classifiers. In *Proc. of the Conference on Learning Theory (COLT)*. 144–152.

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

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

Helma, Christoph, and Kramer, Stefan. 2003. A Survey of the Predictive Toxicology Challenge *Bioinformatics* 19(10): 1179–1182.

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.

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.

Neuhaus, Michel, and Bunke, Horst. 2007. Bridging the Gap Between Graph Edit Distance and Kernel Machines. *World Scientific*.

H. W. Kuhn. 1955. The Hungarian Method for the Assignment Problem. *Naval Research Logistics*, 2: 83–97.

E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann. 2009. An Optimal Decomposition Algorithm for Tree Edit Distance. *ACM Transaction on Algorithm*, 6 (1).

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