Experimental Evaluation of Graph Classification with Hadamard Code Graph Kernels

Tetsuya Kataoka and Akihiro Inokuchi(✉)

School of Science and Technology, Kwansei Gakuin University,
2-1 Gakuen, Sanda, Hyogo, Japan
{TKataoka,inokuchi}@kwansei.ac.jp

Abstract. When mining information from a database comprising graphs, kernel methods are used to efficiently classify graphs that have similar structures into the same classes. Instances represented by graphs usually have similar properties if their graph representations have high structural similarity. The neighborhood hash kernel (NHK) and Weisfeiler–Lehman subtree kernel (WLSK) have previously been proposed as kernels that compute more quickly than the random-walk kernel; however, they each have drawbacks. NHK can produce hash collision and WLSK must sort vertex labels. We propose a novel graph kernel equivalent to NHK in terms of time and space complexities, and comparable to WLSK in terms of expressiveness. The proposed kernel is based on the Hadamard code. Labels assigned by our graph kernel follow a binomial distribution with zero mean. The expected value of a label is zero; thus, such labels do not require large memory. This allows the compression of vertex labels in graphs, as well as fast computation. This paper presents the Hadamard code kernel (HCK) and shortened HCK (SHCK), a version of HCK that compresses vertex labels in graphs. The performance and practicality of the proposed method are demonstrated in experiments that compare the computation time, scalability and classification accuracy of HCK and SHCK with those of NHK and WLSK for both artificial and real-world datasets. The effect of assigning initial labels is also investigated.

Keywords: Graph classification · Support vector machine · Graph kernel · Hadamard code

1 Introduction

A natural way of representing structured data is to use graphs [14]. As an example, the structural formula of a chemical compound is a graph, where each vertex corresponds to an atom in the compound and each edge corresponds to a bond between the two atoms therein. Using such graph representations, a new research field called graph mining has emerged from data mining, with the objective of mining information from a database consisting of graphs. With the potential to find meaningful information, graph mining has received much interest, and research in the field has grown rapidly in recent years. Furthermore, because the need for classifying graphs has strengthened in many real-world applications, such as the analysis of proteins in bioinformatics and chemical compounds in cheminformatics [11], graph classification has been widely researched worldwide. The main objective of graph classification is to classify graphs of similar structures into the same classes. This originates from the fact that instances represented by graphs usually have similar properties if their graph representations have high structural similarity.

Kernel methods such as the use of the support vector machine (SVM) are becoming increasingly popular because of their high performance in solving graph classification problems [8]. Most graph kernels are based on the decomposition of a graph into substructures and a feature vector containing counts of these substructures. Because the dimensionality of these feature vectors is typically high and this approach includes the subgraph isomorphism matching problem that is known to be NP-complete [6], kernels deliberately avoid the explicit computation of feature values and instead employ efficient procedures.

One representative graph kernel is the random-walk kernel (RWK) [8,10], which computes $k(g_i, g_j)$ in $O(|V(g)|^3)$ for graphs g_i and g_j, where $|V(g)|$ is the number of vertices in g_i and g_j. The kernel returns a high value if the random walk on the graph generates many sequences with the same labels for vertices and edges; i.e., the graphs are similar to each other. The neighborhood hash kernel (NHK) [7] and Weisfeiler–Lehman subtree kernel (WLSK) [9] are two other recently proposed kernels that compute $k(g_i, g_j)$ more quickly than RWK. NHK uses logical operations such as the exclusive OR on the label set of adjacent vertices, while WLSK uses a concatenation of label strings of the adjacent vertices to compute $k(g_i, g_j)$. The labels updated by repeating the hash or concatenation propagate the label information over the graph and uniquely represent the higher-order structures around the vertices beyond the vertex or edge level. An SVM with two graph kernels works well with benchmark data consisting of graphs.

The computation of NHK is efficient because it is a logical operation between fixed-length bit strings and does not require string sorting. However, its drawback is hash collision, which occurs when different induced subgraphs have identical hash values. Meanwhile, WLSK must sort the vertex labels, but it has high expressiveness because each vertex v has a distribution of vertex labels within i steps from v. To overcome the drawbacks of NHK and WLSK, in this paper, we propose a novel graph kernel that is equivalent to NHK in terms of time and space complexities and comparable to WLSK in terms of expressiveness. The graph kernel proposed in this paper is based on the Hadamard code [13]. The Hadamard code is used in spread spectrum-based communication technologies such as Code Division Multiple Access to spread message signals. Because the probability of occurrences of values of 1 and -1 are equivalent in each column of the Hadamard matrix except for the first column, labels assigned by our graph kernel follow a binomial distribution with zero mean under a certain assumption. Therefore, the expected value of the label is zero, and for such labels, a large memory space is not required. This characteristic is used to compress vertex labels in graphs, allowing the proposed graph kernel to be computed quickly.

Note that large portions of this paper were covered in our previous work [13]. Within the current work we demonstrate the performance and practicality of the proposed method in experiments that compare the computation time, scalability and classification accuracy of HCK and SHCK with those of NHK and WLSK for various artificial datasets. The effect of assigning initial labels for the graph kernels is also investigated.

The rest of this paper is organized as follows. Section 2 defines the graph classification problem and explains the framework of the existing graph kernels. Section 3 proposes the Hadamard code kernel (HCK), based on the Hadamard code, and another graph kernel called the shortened HCK (SHCK), which is a version of HCK that compresses vertex labels in graphs. Section 4 demonstrates the fundamental performance and practicality of the proposed method through experiments. Finally, we conclude the paper in Sect. 5.

2 Graph Kernels

2.1 Framework of Representative Graph Kernels

This paper tackles the classification problem of graphs. A 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, Σ 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 denoted by $V(g)$. Although we assume that only the vertices in the graphs have labels in this paper, the methods used in this paper can be applied to graphs where both the vertices and edges have labels. The vertices adjacent to vertex v are represented as $N(v) = \{u \mid (v, u) \in E\}$. 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 described as being simple if and only if the path does not have repeating vertices. Paths in this paper are not always simple.

The graph classification problem is defined as follows. Given a set of n training examples $D = \{(g_i, y_i)\}_{i=1}^{n}$, 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 paper, graphs are classified by an SVM that uses graph kernels. Let Σ and $c(g, \sigma)$ be $\{\sigma_1, \sigma_2, \cdots, \sigma_{|\Sigma|}\}$ and $c(g, \sigma) = |\{v \in V(g) \mid \ell(v) = \sigma\}|$, respectively. A function ϕ that converts a graph g to a vector is defined as

$$\phi(g) = \big(c(g, \sigma_1), c(g, \sigma_2), \cdots, c(g, \sigma_{|\Sigma|})\big)^T.$$

Function $k'(g_i, g_j)$, defined as $\phi(g_i)^T \phi(g_j)$, is a semi-positive definite kernel. This function is calculated as

$$k'(g_i, g_j) = \phi(g_i)^T \phi(g_j) = \sum_{v_i \in V(g_i)} \sum_{v_j \in V(g_j)} \delta(\ell(v_i), \ell(v_j)), \tag{1}$$

where δ is the Kronecker delta. When $V(g_i)$ represents a set of vertices in g_i, $O(|V(g_i)| \times |V(g_j)|)$ is required to compute Eq. (1). However, Eq. (1) is solvable in $O(|V(g_i)| + |V(g_j)|)$ by using Algorithm 1 [7]. In Lines 1, a multiset of labels in the graph g_i is sorted in ascending order by using the radix sort. This requires $O(|V(g_i)|)$. Similarly, a multiset of labels in the graph g_i is also sorted in Line 2. In Lines 7 and 8, the a-th and b-th elements of the sorted labels are selected, respectively. Then, how many labels the graph have in common is counted in Line 10. This process is continued at most $|V(g_i)| + |V(g_j)|$ iterations.

Algorithm 1. Basic_Graph_Kernel.

```
    Data: two graphs $g_i = (V_i, E_i, \Sigma, \ell_i)$ and $g_j = (V_j, E_j, \Sigma, \ell_j)$
    Result: $k'(g_i, g_j)$
 1  $V_i^{sort} \leftarrow$Radix_Sort$(g_i, \ell_i)$;
 2  $V_j^{sort} \leftarrow$Radix_Sort$(g_j, \ell_j)$;
 3  $\kappa \leftarrow 0$;
 4  $a \leftarrow 1$;
 5  $b \leftarrow 1$;
 6  for $a \le |V(g_i)| \wedge b \le |V(g_j)|$ do
 7      $v_i \leftarrow V_i^{sort}[a]$;
 8      $v_j \leftarrow V_j^{sort}[b]$;
 9      if $\ell_i(v_i) = \ell_j(v_j)$ then
10          $\kappa \leftarrow \kappa + 1$;
11          $a \leftarrow a + 1$;
12          $b \leftarrow b + 1$;
        else
            if $\ell_i(v_i) < \ell_j(v_j)$ then
13              $a \leftarrow a + 1$;
            else
14              $b \leftarrow b + 1$;
15  return $\kappa$;
```

Given a $g^{(h)} = (V, E, \Sigma, \ell^{(h)})$, a procedure that converts $g^{(h)}$ to another graph $g^{(h+1)} = (V, E, \Sigma', \ell^{(h+1)})$ is called a relabel. Although relabel function $\ell^{(h+1)}$ is defined later in detail, the label of a v in $g^{(h+1)}$ is defined using the labels of v and $N(v)$ in $g^{(h)}$, and is denoted as $\ell^{(h+1)}(v) = r(v, N(v), \ell^{(h)})$. Let $\{g^{(0)}, g^{(1)}, \cdots, g^{(h)}\}$ be a series of graphs obtained by iteratively applying a relabel h times, where $g^{(0)}$ is a graph contained in D. Given two graphs g_i and g_j, a graph kernel is defined using k' as

$$k(g_i, g_j) = k'(g_i^{(0)}, g_j^{(0)}) + k'(g_i^{(1)}, g_j^{(1)}) + \cdots + k'(g_i^{(h)}, g_j^{(h)}). \tag{2}$$

Because k is a summation of semi-positive definite kernels, k is also semi-positive definite [3]. In addition, Eq. (2) is solvable in $O(h(|V(g_i)| + |V(g_j)|))$ according Algorithm 1.

Recently, various graph kernels have been applied to the graph classification problem. Representative graph kernels such as NHK and WLSK follow the above framework, where graphs contained in D are iteratively relabeled. In these kernels, $\ell^{(h)}(v) = r(v, N(v), \ell^{(h-1)})$ characterizes a subgraph induced by the vertices that are reachable from v within h steps in $g^{(0)}$. Therefore, given $v_i \in V(g_i)$ and $v_j \in V(g_j)$, if subgraphs of the graphs induced by the vertices reachable from vertices v_i and v_j within h steps are identical, the relabel assigns identical labels to them. Additionally, it is desirable for a graph kernel to fulfill the converse of this condition. However, it is not an easy task to design such a graph kernel.

We now review the representative graph kernels, NHK and WLSK.

NHK: Given a fixed-length bit string $\ell_1^{(0)}(v)$ of length L, $\ell_1^{(h)}(v)$ is defined as

$$\ell_1^{(h)}(v) = ROT(\ell_1^{(h-1)}(v)) \oplus \left(\bigoplus_{u \in N(v)} \ell_1^{(h-1)}(u) \right),$$

where ROT is bit rotation to the left and $\oplus$ is the exclusive OR of the bit strings. NHK is efficient in terms of computation and space complexities because the relabel of NHK is computable in $O(L|N(v)|)$ for each vertex and its space complexity is $O(L)$.

Figure 1 shows an example of an NHK relabel and its detailed calculation for a vertex v_2, assuming that $L = 3$. First, $\ell_1^{(0)}(v_2) = \#011$ is rotated to return #110. We then obtain #001 using the exclusive OR of #110, $\ell_1^{(0)}(v_1) = \#011$, $\ell_1^{(0)}(v_3) = \#001$, $\ell_1^{(0)}(v_4) = \#001$, and $\ell_1^{(0)}(v_5) = \#100$. In this computation, we do not require sorted bit strings because the exclusive OR is commutative. Three bits are required for $\ell_1^{(0)}(v_2)$ in this example, and $\ell_1^{(h)}(v_2)$ also requires three bits, even if h is increased.

NHK has a drawback with respect to accidental hash collisions. For example, vertices v_1, v_3, and v_4 in $g^{(1)}$ in Fig. 1 have identical labels after the relabel. This is because v_3 and v_4 in $g^{(0)}$ have identical labels and the same number of adjacent vertices. However, despite the different labels and numbers of adjacent vertices of v_1 and v_3, these vertices have the same vertex labels in $g^{(1)}$, leading to low graph expressiveness and low classification accuracy.

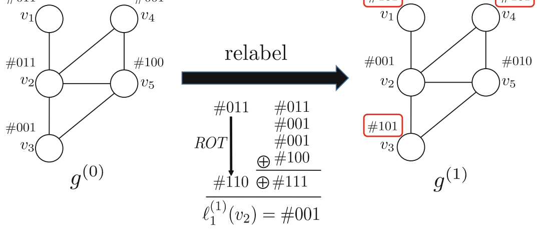


Fig. 1. Relabeling $g^{(0)}$ to $g^{(1)}$ in NHK.

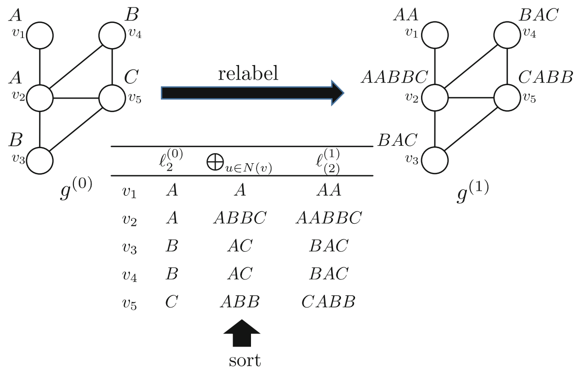
We next describe WLSK, which is based on the Weisfeiler–Lehman algorithm, an algorithm that determines graph isomorphism.

WLSK: When $\ell_2^{(0)}(v)$ returns a string of characters, $\ell_2^{(h)}(v)$ is defined as

$$\ell_2^{(h)}(v) = \ell_2^{(h-1)}(v) \cdot \left(\bigodot_{u \in N(v)} \ell_2^{(h-1)}(u) \right),$$

where $\cdot$ and $\bigodot$ are string concatenation operators. Because concatenation is not commutative, u is an iterator that obtains the vertices $N(v)$ adjacent to v in alphabetical order. Because $\ell_2^{(h)}(v)$ has information on the distribution of labels for h steps from v, it has high graph expressiveness.[1] If the labels are sorted using a bucket sort, the time complexity of WLSK is $O(|\Sigma||N(v)|)$ for each vertex.

Figure 2 shows an example of a relabel using WLSK. Vertices v_1, v_2, v_3, v_4, and v_5 in $g^{(0)}$ have labels A, A, B, B, and C, respectively. For each vertex, WLSK sorts the labels of the vertices adjacent to the vertex, and then concatenates these labels. In $g^{(1)}$, v_3 has the label BAC, meaning that v_3 has the label B in $g^{(0)}$ and two adjacent vertices whose labels are A and C.



	$\ell_2^{(0)}$	$\bigoplus_{u \in N(v)}$	$\ell_{(2)}^{(1)}$
v_1	A	A	AA
v_2	A	$ABBC$	$AABBC$
v_3	B	AC	BAC
v_4	B	AC	BAC
v_5	C	ABB	$CABB$

Fig. 2. Relabeling $g^{(0)}$ to $g^{(1)}$ in WLSK.

In addition to NHK and WLSK, we define the label aggregate kernel (LAK) to facilitate the understanding of the other kernels proposed in this paper.

LAK: In this kernel, $\boldsymbol{\ell}_3^{(0)}(v)$ is a vector in $|\Sigma|$-dimensional space. In concrete terms, if a vertex in a graph has a label σ_i among $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_{|\Sigma|}\}$, the i-th element in the vector is 1. Otherwise, it is zero. In LAK, $\boldsymbol{\ell}_3^{(h)}(v)$ is defined as

$$\boldsymbol{\ell}_3^{(h)}(v) = \boldsymbol{\ell}_3^{(h-1)}(v) + \sum_{u \in N(v)} \boldsymbol{\ell}_3^{(h-1)}(u).$$

[1] When $\ell_2^{(0)}(v)$ is a string of length 1, $\ell_2^{(1)}(v)$ is a string of length $|N(v)| + 1$. By replacing the later string with a new string of length 1, both the computation time and memory space required by WLSK are reduced.

The i-th element in $\boldsymbol{\ell}_3^{(h)}(v)$ is the frequency of occurrence of character σ_i in the string $\ell_2^{(h)}(v)$ concatenated by WLSK. Therefore, $\boldsymbol{\ell}_3^{(h)}(v)$ has information on the distribution of labels within h steps from v. Hence, LAK has high graph expressiveness. However, when h is increased, the number of paths from v that reach vertices labeled σ_i increases exponentially. Thus, elements in $\boldsymbol{\ell}_3^{(h)}(v)$ also increase exponentially. For example, if the average degree of vertices is d, there are $(d+1)^h$ vertices reachable from v within h steps. LAK thus requires a large amount of memory.

Figures 3 and 4 show an example of a relabel using LAK, assuming that $|\Sigma| = 3$. The vertex label of v_5 in $g^{(1)}$ is (1, 2, 1), which means that there are one, two, and one vertices reachable from v within one step that have labels σ_1, σ_2, and σ_3, respectively. Compared with relabeling $g^{(0)}$ to $g^{(1)}$, the additional number of values in $\boldsymbol{\ell}_3^{(h)}(v)$ when relabeling $g^{(3)}$ to $g^{(4)}$ is large.

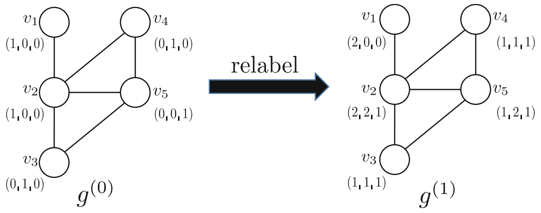


Fig. 3. Relabeling $g^{(0)}$ to $g^{(1)}$ in LAK.

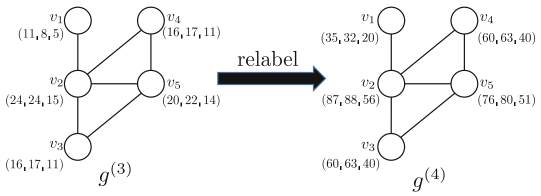


Fig. 4. Relabeling $g^{(3)}$ to $g^{(4)}$ in LAK.

2.2 Drawbacks of Existing Graph Kernels

We here summarize the characteristics of the above three graph kernels. NHK is efficient because its computation is a logical operation between fixed-length bit strings and does not require string sorting. However, its drawback is a tendency for hash collision, where different induced subgraphs have identical hash values. Meanwhile, WLSK requires vertex label sorting, but it has high expressiveness because $\ell_2^{(h)}(v)$ contains the distribution of the vertex labels within h' steps

$(0 \le h' \le h)$ from v. LAK requires a large amount of memory to store vectors for high h although it does not require label sorting. To overcome the drawbacks of NHK, WLSK and LAK, in this paper, we propose a novel graph kernel that is equivalent to NHK in terms of time and space complexities and equivalent to LAK in terms of expressiveness.

3 Graph Kernels Based on the Hadamard Code

In this section, we propose a novel graph kernel with the Hadamard code to overcome the aforementioned drawbacks. A Hadamard matrix is a square $(-1, 1)$ matrix in which any two row vectors are orthogonal, defined as

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad H_{2^k} = \begin{pmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{pmatrix}.$$

A Hadamard code is a row vector of the Hadamard matrix. Given a Hadamard matrix of order 2^k, 2^k Hadamard codes having 2^k elements are generated from this matrix. Using the Hadamard codes, we propose HCK as follows.

HCK: Let H be a Hadamard matrix of order $2^{\lceil \log_2 |\Sigma| \rceil}$ and $\boldsymbol{\ell}_4^{(0)}(v)$ be a Hadamard code of order $|H|$. If a vertex v has label σ_i, the i-th row in the Hadamard matrix of order $|H|$ is assigned to the vertex. $\boldsymbol{\ell}_4^{(h)}(v)$ is then defined as

$$\boldsymbol{\ell}_4^{(h)}(v) = \boldsymbol{\ell}_4^{(h-1)}(v) + \sum_{u \in N(v)} \boldsymbol{\ell}_4^{(h-1)}(u).$$

When $\boldsymbol{\ell}_{\sigma_i}$ is a Hadamard code for a vertex label σ_i, $\boldsymbol{\ell}_{\sigma_i}^T \boldsymbol{\ell}_4^{(h)}(v)/|H|$ is the occurrence of σ_i in a string $\ell_2^{(h)}(v)$ generated by WLSK. HCK therefore has the same expressiveness as LAK.

Figure 5 shows an example of a relabel using HCK. Each vertex v in $g^{(1)}$ is represented as a vector produced by the summation of vectors for vertices adjacent to v in $g^{(0)}$. Additionally, after the relabel, we obtain the distribution of the vertex labels within one step of v according to

$$\frac{1}{|H|} H \boldsymbol{\ell}_4^{(1)}(v_5) = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 4 \\ 0 \\ 2 \\ -2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \end{pmatrix}.$$

In other words, there are one σ_1, two σ_2, and one σ_3 labels within one step of v_5. Furthermore, the result is equivalent to $\boldsymbol{\ell}_3^{(1)}(v_5)$, as shown in Fig. 3. The reason why we divide $H\boldsymbol{\ell}_4^{(h)}(v)$ by 4 is that the order of the Hadamard matrix is $|H| = 4$.

If each element in $\boldsymbol{\ell}_4^{(h)}(v)$ is stored in four bytes (the commonly used size of integers in C, Java, and other languages), the space complexity of HCK is

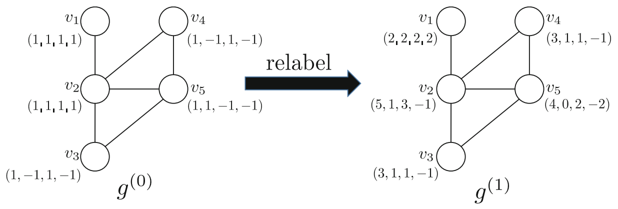


Fig. 5. Relabeling $g^{(0)}$ to $g^{(1)}$ in HCK.

equivalent to that of LAK. We therefore have not yet overcome the drawback of LAK. In this paper, we assume that each vertex label is assigned to a vertex with equal probability. The probabilities of occurrence of 1 and -1 are equivalent in each column of the Hadamard matrix except for the first column, and the i-th element $(1 < i \leq |\Sigma|)$ in $\boldsymbol{\ell}_4^{(h)}(v)$ follows a binomial distribution with zero mean under this assumption. Therefore, the expected value of the element in $\boldsymbol{\ell}_4^{(h)}(v)$ is zero, and for the elements, a large memory space is not required. For example, Tables 1 and 2 present values of the i-th elements in $\boldsymbol{\ell}_3^{(h)}(v_2)$ and $\boldsymbol{\ell}_4^{(h)}(v_2)$, respectively, in a graph $g^{(h)}$, when $g^{(0)}$ (shown in Fig. 6) is relabeled iteratively h times. Under the assumption of the vertex label probability, the expected value of all elements in $\boldsymbol{\ell}_4^{(h)}(v_2)$ except for the first element becomes zero. The first element represents the number of paths from v_2 to the vertices reachable within h steps. On the basis of this observation, we assign bit arrays of length ρ in the L-bit array to the elements as follows.

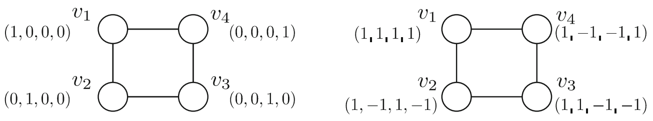
SHCK: Similar to NHK, $\ell_5^{(0)}(v)$ is a fixed-length bit array of length L. The bit array is divided into $|H|$ fragments, one of which is a bit array of length $L - \rho(|H| - 1)$ and the rest are bit arrays of length ρ. The first fragment of length $L - \rho(|H| - 1)$ is assigned to store the first element of $\boldsymbol{\ell}_4^{(0)}(v)$, the next

Table 1. Elements in a label in LAK.

h	Label
0	$\boldsymbol{\ell}_3^{(0)}(v_2) = (0\ 1\ 0\ 0)$
1	$\boldsymbol{\ell}_3^{(1)}(v_2) = (1\ 1\ 1\ 0)$
2	$\boldsymbol{\ell}_3^{(2)}(v_2) = (2\ 3\ 2\ 2)$
3	$\boldsymbol{\ell}_3^{(3)}(v_2) = (7\ 7\ 7\ 6)$
4	$\boldsymbol{\ell}_3^{(4)}(v_2) = (20\ 21\ 20\ 20)$
5	$\boldsymbol{\ell}_3^{(5)}(v_2) = (61\ 61\ 61\ 60)$
6	$\boldsymbol{\ell}_3^{(6)}(v_2) = (182\ 183\ 182\ 182)$
7	$\boldsymbol{\ell}_3^{(7)}(v_2) = (547\ 547\ 547\ 546)$
8	$\boldsymbol{\ell}_3^{(8)}(v_2) = (1640\ 1641\ 1640\ 1640)$
9	$\boldsymbol{\ell}_3^{(9)}(v_2) = (4921\ 4921\ 4921\ 4920)$
10	$\boldsymbol{\ell}_3^{(10)}(v_2) = (14762\ 14763\ 14762\ 14762)$

Table 2. Elements in a label in HCK.

h	Label
0	$\boldsymbol{\ell}_4^{(0)}(v_2) = (1\ -1\ -1\ 1)$
1	$\boldsymbol{\ell}_4^{(1)}(v_2) = (3\ -1\ -1\ -1)$
2	$\boldsymbol{\ell}_4^{(2)}(v_2) = (9\ -1\ -1\ 1)$
3	$\boldsymbol{\ell}_4^{(3)}(v_2) = (27\ -1\ -1\ -1)$
4	$\boldsymbol{\ell}_4^{(4)}(v_2) = (81\ -1\ -1\ 1)$
5	$\boldsymbol{\ell}_4^{(5)}(v_2) = (243\ -1\ -1\ -1)$
6	$\boldsymbol{\ell}_4^{(6)}(v_2) = (729\ -1\ -1\ 1)$
7	$\boldsymbol{\ell}_4^{(7)}(v_2) = (2187\ -1\ -1\ -1)$
8	$\boldsymbol{\ell}_4^{(8)}(v_2) = (6561\ -1\ -1\ 1)$
9	$\boldsymbol{\ell}_4^{(9)}(v_2) = (19683\ -1\ -1\ -1)$
10	$\boldsymbol{\ell}_4^{(10)}(v_2) = (59049\ -1\ -1\ 1)$



(a) A graph $g^{(0)}$ relabeled by LAK (b) A graph $g^{(0)}$ relabeled by HCK

Fig. 6. Relabeled graphs.

fragment of length ρ is assigned to store the second element, and so on. Here, ρ is a positive integer fulfilling $\rho(|H|-1) = \rho(2^{\lceil \log_2 |\Sigma| \rceil} - 1) \leq L$. Additionally, each element of $\boldsymbol{\ell}_4^{(0)}(v)$ is represented by its two's complement in $\ell_5^{(0)}(v)$ for the purpose of the following summation, which defines $\ell_5^{(h)}(v)$.

$$\ell_5^{(h)}(v) = \ell_5^{(h-1)}(v) + \sum_{u \in N(v)} \ell_5^{(h-1)}(u).$$

Because $\ell_5^{(h)}(v)$ is a fixed-length binary bit string and $\ell_5^{(h)}(v)$ is the summation of the values represented as bit strings, both the time and space complexities of SHCK are equivalent to those of NHK. Additionally, the expressiveness of SHCK is equivalent that of LAK, if overflow of the fixed-length bit array does not occur. The theoretical discussion on the overflow can be found in [13]. In the next section, we demonstrate that the proposed graph kernel, SHCK, has the ability to classify graphs with high accuracy.

4 Experimental Evaluation

The proposed method was implemented in Java. All experiments were done on an Intel Xeon X5670 2.93-GHz computer with 48-GB memory running Microsoft Windows 8. We compared the computation time and accuracy of the prediction performance of HCK and SHCK with those of NHK and WLSK. To learn from the kernel matrices generated by the above graph kernels, we used the LIBSVM package [2][2] using 10-fold cross validation.

4.1 Experiments on Artificial Datasets

We generated artificial datasets of graphs using the four parameters and their default values listed in Table 3. For each dataset, $|D|$ graphs, each with an average of $|V(g)|$ vertices, were generated. Two vertices in a graph were connected with probability p of the existence of an edge, and one of $|\Sigma|$ labels was assigned to each vertex in the graph. In parallel with the dataset generation and using the same parameters p and $|\Sigma|$, a graph pattern g_s with 15 vertices was also generated for embedding in some of the $|D|$ graphs as common induced subgraphs.

[2] http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

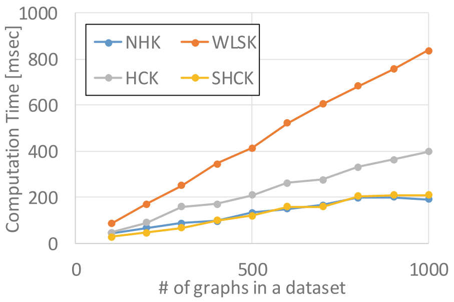


Fig. 7. Computation time for various $|D|$.

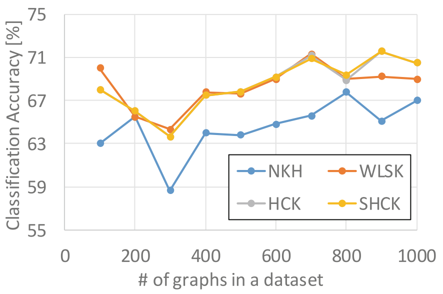


Fig. 8. Accuracy for various $|D|$.

Table 3. Default parameters for the data generation program.

	# of graphs in a dataset	Average number of vertices in graphs	Prob. of edge existence	# of vertex labels
Default values	$\|D\| = 200$	$\|V(g)\| = 100$	$p = 5\%$	$\|\Sigma\| = 2$

g_s was randomly embedded in half of the $|D|$ graphs in a dataset, and the class label 1 was assigned to the graphs containing g_s, while the class label -1 was assigned to the other graphs.

First, we varied only $|D|$ to generate various datasets with the other parameters set to their default values. The number of graphs in each dataset was varied from 100 to 1,000. Figures 7 and 8 show the computation time to relabel graphs in each dataset and classification accuracy, respectively, for the four graph kernels. In these experiments, h and ρ were set to 5 and 3, respectively, and the computation time does not contain time to generate kernel matrices from $g^{(0)}, g^{(1)}, \cdots, g^{(h)}$. As shown in Fig. 7, the computation time for all of the graph kernels is proportional to the number of graphs in a dataset, because the relabels in the kernels are applied to each graph independent to the other graphs in the dataset. As shown in Fig. 8, the classification accuracy increases when the number of graphs is increased. This is because the number of graphs in training datasets of SVMs also increase, when the number of graphs is increased. The classification accuracies for WLSK, HCK, and SHCK are superior to one for NHK, because they have high expressiveness. The classification accuracy for HCK is almost as equivalent as one for SHCK. Therefore, overflows in SHCK do not make much impact to the accuracy, which will be also shown in the other experiments.

Next, we varied only $|\Sigma|$ to generate various datasets with the other parameters set to their default values. Figures 9 and 10 show the computation time to relabel graphs in each dataset and classification accuracy, respectively, for the four graph kernels when the number of labels in each dataset was varied from 1 to 10. The computation time for NHK, WLSK, and SHCK is constant to the

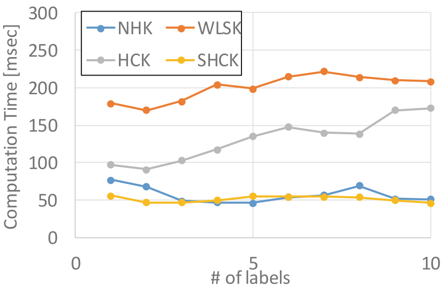


Fig. 9. Computation time for various $|\Sigma|$.

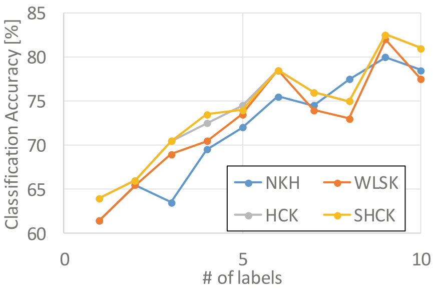


Fig. 10. Accuracy for various $|\Sigma|$.

number of labels in datasets, while one for HCK increases when the number of labels is increased. This is because HCK represents a label of each vertex in a graph by a $|\Sigma|$-dimensional vector and it sums up such $|\Sigma|$-dimensional vectors. On the other hand, NHK and SHCK represent the label by a fix-length bit string whose length is independent to the number of labels of graphs. The classification accuracy for all of the graph kernels increases when the number of labels in datasets, because labels which vertices in graphs have carry important information to classify graphs correctly. Especially in WLSK, HCK, and SHCK, since each vertex v has a distribution of vertex labels within i steps from v, their accuracies are superior to one for NHK.

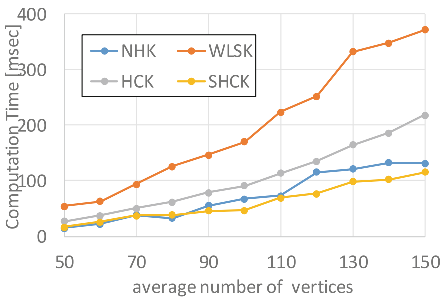


Fig. 11. Computation time for various $|V(g)|$.

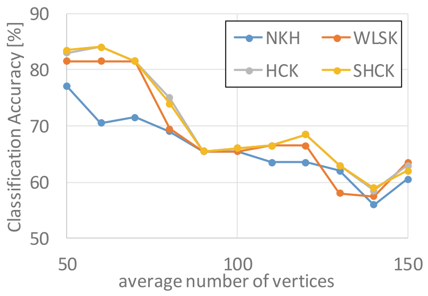


Fig. 12. Accuracy for various $|V(g)|$.

We varied only $|V(g)|$ to generate various datasets with the other parameters set to their default values. Figures 11 and 12 show the computation time to relabel graphs in each dataset and classification accuracy, respectively, when the average number of vertices in graphs was varied from 50 to 150. The computation time for all of the graph kernels increases with the square of the average number $|V(g)|$ of vertices in graphs. This is because a new label for each vertex v in a graph $g^{(h+1)}$ is obtained from labels of v and its adjacent vertices U in a graph

$g^{(h)}$ and the relabel for $g^{(h)}$ for any h is computable in $O(|V(g)| \times |U|)$, where $|U|$ is $p \times |V(g)|$. Since graphs in most of real world datasets are sparse, the graph kernels in this paper are applicable enough to such sparse graphs even if the number of vertices in graphs is large. For example, chemical compounds which can be represented graphs where each vertex and edge correspond to an atom and edge in compounds, respectively, are sparse, since atoms in chemical compounds have two chemical bonds in average. The classification accuracy decreases, when the average number of vertices in graphs is increased, because the number of graphs whose class labels are -1 and which contain some small subgraphs of the embedded pattern g_s increases when the average number of vertices in graphs is increased.

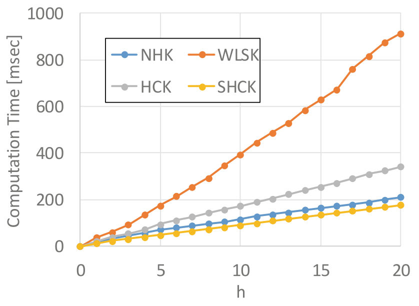


Fig. 13. Computation time for various h.

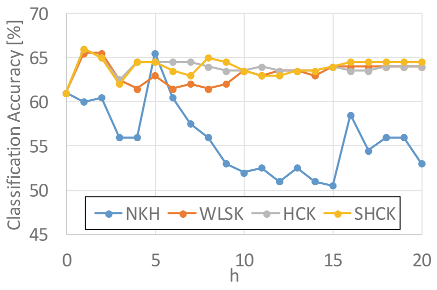


Fig. 14. Accuracy for various h.

We varied h for the graph kernels. Figures 13 and 14 show the computation time to relabel graphs in each dataset and classification accuracy, respectively, for the four graph kernels in the experiments, when h was varied from 1 to 20. The computation time for all of the graph kernels is proportional to h. When h is increased, the graph kernel has information on a large subgraph induced by vertices reachable within h steps from every vertex in a graph. However, the classification accuracy does not always increase, when h is increased. We do not know an adequate value for h in advance of training SVMs. One of ways for determining an adequate value for h for a dataset is to divide the dataset into three portions, one of them is for the training dataset of an SVM, another is for the dataset to obtain an adequate value for h and the SVM's parameters, and the other is for testing the SVM.

From these experimental results using artificial datasets, we confirmed that the proposed graph kernel SHCK is equivalent to NHK in terms of time complexity and equivalent to WLSK in terms of classification accuracy. The experimental results do not contain time for computing kernel matrices from $g^{(0)}, g^{(1)}, \cdots, g^{(h)}$ obtained by the graph kernels. A kernel matrix for a dataset D is obtained in $O(h|D|^2|V(g)|)$, where $|V(g)|$ is the average number of vertices of graphs in D, because we need to run Algorithm 1 h times for every pair of graphs in D.

4.2 Experiments on Real-World Graphs

To assess the practicability of our proposed method, we used five real-world datasets. The first dataset, MUTAG [4], contains information on 188 chemical compounds and their class labels. The class labels are binary values that indicate the mutagenicity of chemical compounds. The second dataset, ENZYMES, contains information on 600 proteins and their class labels. The class labels are one of six labels showing the six EC top-level classes [1,12]. The third dataset, D&D, contains information on 1178 protein structures, where each amino acid corresponds to a vertex and two vertices are connected by an edge if they are less than 6 Ångstroms apart [5]. The remaining datasets, NCI1 and NCI109, represent two balanced subsets of data sets of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively [15]. These datasets contain about 4000 chemical compounds, each of which has a class label among positive and negative. 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 vertices in graphs have labels, the chemical graphs are converted following the literature [7]; i.e., 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. Table 4 summarizes the datasets.

Table 4. Summary of evaluation datasets.

	MUTAG	ENZYMES	D&D	NCI1	NCI109
Number of graphs $\|D\|$	188	600	1178	4110	4127
Maximum graph size	84	126	5748	349	349
Average graph size	53.9	32.6	284.3	94.5	93.9
Number of labels $\|\Sigma\|$	12	3	82	40	41
Number of classes (class distribution)	2 (126,63)	6 (100,100,100, 100,100,100)	2 (487, 691)	2 (2053, 2057)	2 (2048, 2079)
Average degree of vertices	2.1	3.8	5.0	2.7	2.7

Figures 15, 16, 17, 18, and 19 show the computation time required to obtain a graph $g^{(h)}$ from a graph $g^{(0)}$ in NHK, WLSK, HCK, and SHCK for various h for the MUTAG, ENZYMES, D&D, NCI1, and NCI109 datasets, respectively. As shown in the figures, NHK and SHCK are faster than HCK, and much faster than WLSK. Additionally, the computation times of NHK, HCK, and SHCK increase

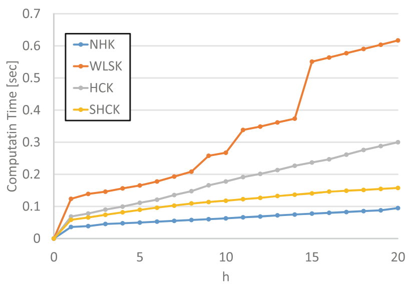


Fig. 15. Computation time for various h (MUTAG).

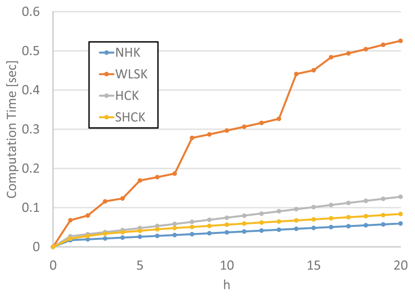


Fig. 16. Computation time for various h (ENZYMES).

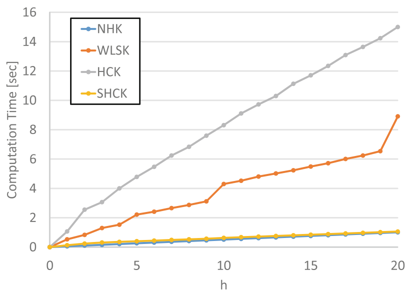


Fig. 17. Computation time for various h (D&D).

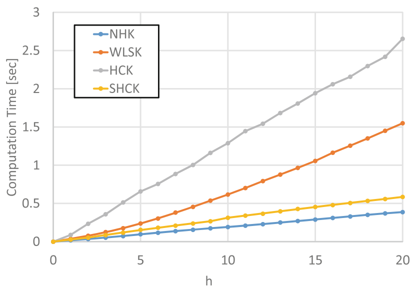


Fig. 18. Computation time for various h (NCI1).

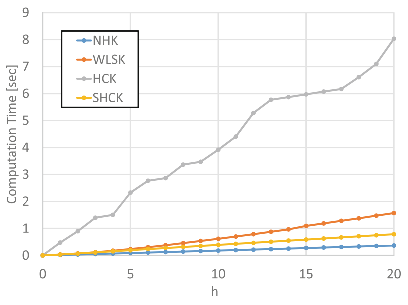


Fig. 19. Computation time for various h (NCI109).

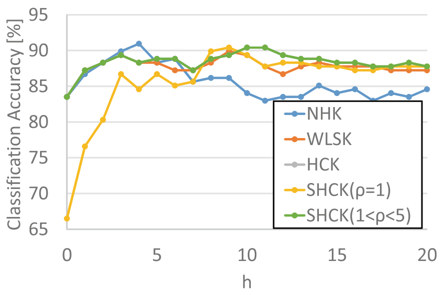


Fig. 20. Accuracy for various h and ρ (MUTAG).

linearly with h. The reason why WLSK requires such a long computation time is that WLSK must sort the labels of adjacent vertices and replace a string of length $|N(v)|+1$ with a string of length 1. This is especially true when $h = 11$ or 15 for the MUTAG dataset, $h = 8$ or 14 for the ENZYMES dataset, and $h = 10$ or 20 for the D&D dataset. In our implementation, this replacement is done with Java's HashMap class, where a string of length $|N(v)| + 1$ is the hash key and a string of length 1 is a value corresponding to that key. Although the average degree in the evaluated datasets is low, WLSK requires further computation time when the average degree of the data increases. HCK requires a long computation time for the D&D, NCI1, and NCI109 datasets, because there are many labels in the datasets and the computation time is proportional to the number of labels.

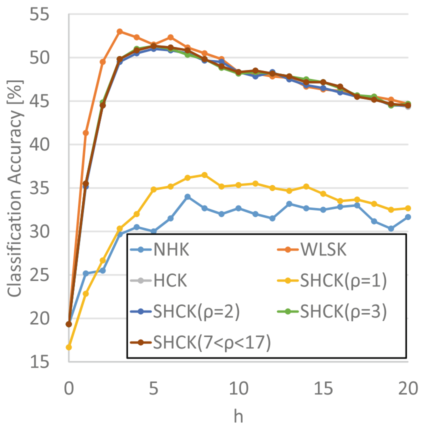


Fig. 21. Accuracy for various h and ρ (ENZYMES).

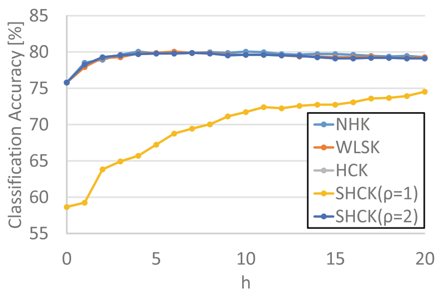


Fig. 22. Accuracy for various h and ρ (D&D).

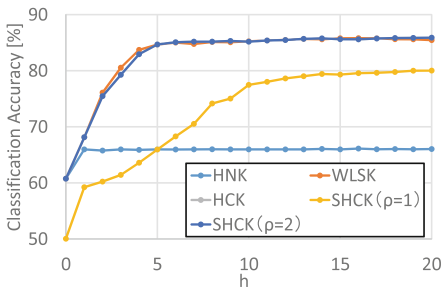


Fig. 23. Accuracy for various h and ρ (NCI1).

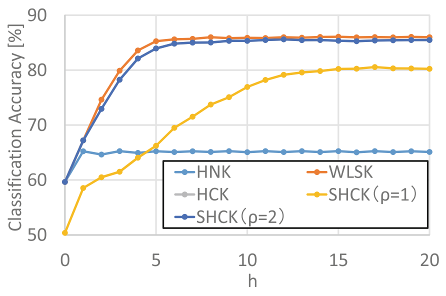


Fig. 24. Accuracy for various h and ρ (NCI109).

Figure 20 shows the classification accuracy of NHK, WLSK, HCK, and SHCK for various h and ρ in the case of the MUTAG dataset. The length of bit strings for NHK and SHCK was set to $L = 64$. The maximum accuracies for various h are almost the same. When $h = 0$, the accuracy for SHCK ($\rho = 1$) is very low because a value of 1 or -1 (the values in the Hadamard matrix) cannot be stored as a two's complement consisting of one bit. The accuracy of HCK is exactly the same as that of SHCK ($1 < \rho < 5$), which means that although overflow may occur in SHCK, the kernel can assign identical vertex labels to the identical subgraphs induced by a vertex v and the vertices within h steps from v. Figure 21 shows the classification accuracy of NHK, WLSK, HCK, and SHCK for various h and ρ in the case of the ENZYMES dataset. WLSK is slightly more accurate than HCK and SHCK ($\rho = 2$, $\rho = 3$, and $7 < \rho < 17$), which are much more accurate than NHK and SHCK ($\rho = 1$). The performance of HCK is exactly the same as that of SHCK for high ρ ($7 < \rho < 17$) and almost the same as that of SHCK for low ρ ($\rho = 2$ and $\rho = 3$). The maximum accuracy of WLSK is 53.0%, while the maximum accuracies of HCK and SHCK ($\rho = 3$, 4, and $7 < \rho < 17$) are both 51.3%. WLSK is slightly more accurate than HCK, because $\ell_2^{(h)}(v)$ contains information on the distribution of labels at h steps from v, while $\boldsymbol{\ell}_4^{(h)}(v)$ contains information on the distribution of all labels within h steps from v. Although the latter distribution can be obtained from the former distribution, the former distribution cannot be obtained from the latter distribution. Therefore, WLSK is more expressive than HCK and SHCK. When ρ is increased to 16, the length of a bit string needed to store the first element of $\boldsymbol{\ell}_4^{(h)}(v)$ is $L - \rho \times 2^{\lceil \log_2 |\Sigma| \rceil} = 64 - 16 \times 2^{\lceil \log_2 3 \rceil} = 0$. Even in this case, the accuracy of SHCK is equivalent to that of HCK, which means that the overflow of the first element of $\boldsymbol{\ell}_4^{(h)}(v)$ has absolutely no effect on the classification accuracy.

Figure 22 shows the classification accuracy of NHK, WLSK, HCK, and SHCK for various h and ρ in the case of the D&D dataset. The lengths of bit strings of NHK and SHCK were set to $L = 256$. All accuracies except for that of SHCK ($\rho = 1$) are almost equivalent. In addition, Figs. 23 and 24 show the classification accuracy of NHK, WLSK, HCK, and SHCK for various h and ρ in the cases of the NCI1 and NCI109 datasets, respectively. All accuracies except those of NHK and SHCK ($\rho = 1$) are almost equivalent. The classification accuracy of NHK is low owing to hash collision, while the classification accuracys of SHCK ($\rho = 1$) is low because a value of 1 or -1 (the values in the Hadamard matrix) cannot be stored as a two's complement consisting of one bit. From the results of these experiments, we recommend setting as large a value as possible for ρ to obtain high classification accuracy. To do so, we set $\rho = \lfloor \frac{L}{|\Sigma|} \rfloor$, where L is the length of the bit array and $|\Sigma|$ is the number of labels in a dataset.

4.3 Effect of Assignment of Initial Labels

The proposed graph kernels HCK and SHCK randomly assign one of the Hadamard codes to each label. The assignment of Hadamard codes to labels affects the classification accuracy especially when there are few labels in a dataset. In this experiment, we demonstrate the relationship between classification accuracies and assignments of initial labels. Figure 25 shows the standard

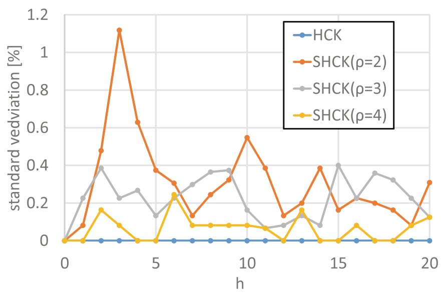


Fig. 25. Effect of assignment of initial labels for various h and ρ (ENZYMES).

deviations of five classification accuracies, each of which is obtained by 10-fold cross validation for the ENZYME dataset, which has the smallest number of labels among the five datasets. Figure 25 shows a high standard deviation for small ρ. In SHCK, a label $\ell_5^{(h)}(v)$ represents a subgraph induced by vertices reachable from v within h step. When there are many labels in a dataset, nonisomorphic subgraphs induced by vertices reachable from a vertex within h steps are likely represented by distinct labels. Meanwhile, when there are few labels in a dataset, the nonisomorphic subgraphs may be represented by the same label. In addition, the small ρ causes the overflow in $\ell_5^{(h)}(v)$, because a subgraph induced by vertices reachable from v within h is represented by an short bit array with length ρ in $\boldsymbol{\ell}_5^{(h)}(v)$. Whether nonisomonphic subgraphs are represented by the same labels varies according to the location where the overflow occurs, which affects the classification accuracy. Therefore, when ρ is small, the standard deviation of classification accuracies of SHCK becomes high. However, it is possible to decrease the standard deviation by decreasing the value of ρ, because this reduces the possibility of overflow occurring. The standard deviation of the classification accuracy for HCK in which overflow does not occur is zero. The same experiment was conducted for the other datasets, and the standard deviation of the classification accuracy was zero because these datasets have enough labels.

5 Conclusion

In this paper, we proposed a novel graph kernel equivalent to NHK in terms of time and space complexities, and comparable to WLSK in terms of expressiveness. The proposed kernel is based on the Hadamard code. Labels assigned by our graph kernel follow a binomial distribution with zero mean. The expected value of a label is zero; thus, such labels do not require large memory. This allows the compression of vertex labels in graphs, as well as fast computation. We presented the Hadamard code kernel (HCK) and shortened HCK (SHCK), a version of HCK that compresses vertex labels in graphs. The fundamental performance and practicality of the proposed method were demonstrated in experiments that compare the computation time, scalability and classification accuracy of HCK and SHCK with those of NHK and WLSK for the artificial and real-world datasets.

References

1. Borgwardt, K.M., Cheng, S.O., Schonauer, S., Vishwanathan, S.V.N., Smola, A.J., Kriegel, H.-P.: Protein function prediction via graph kernels. Bioinfomatics **21**(suppl 1), 47–56 (2005)
2. Chang, C.-C., Lin, C.-J.: LIBSVM: A Library for Support Vector Machines (2001). http://www.csie.ntu.edu.tw/cjlin/libsvm
3. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, Cambridge (2000)
4. Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. J. Med. Chem. **34**, 786–797 (1991)
5. Dobson, P.D., Doig, A.J.: Distinguishing enzyme structures from non-enzymes without alignments. J. Mol. Biol. **330**(4), 771–783 (2003)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, Gordonsville (1979)
7. Hido, S., Kashima, H.: A linear-time graph kernel. In: Proceedings of the IEEE International Conference on Data Mining (ICDM), pp. 179–188 (2009)
8. Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 321–328 (2003)
9. Shervashidze, N., Schweitzer, P., van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-Lehman graph kernels. J. Mach. Learn. Res. (JMLR) **12**, 2539–2561 (2011)
10. Schölkopf, B., Smola, A.J.: Learning with Kernels. MIT Press, Cambridge (2002)
11. Schölkopf, B., Tsuda, K., Vert, J.-P.: Kernel Methods in Computational Biology. MIT Press, Cambridge (2004)
12. Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., Schomburg, D.: BRENDA, the enzyme database: updates and major new developments. Nucleic Acids Res. **32D**, 431–433 (2004)
13. Kataoka, T., Inokuchi, A.: Hadamard code graph kernels for classifying graphs. In: Proceedings of the International Conference on Pattern Recognition Applications and Methods (ICPRAM), pp. 24–32 (2016)
14. Vinh, N.D., Inokuchi, A., Washio, T.: Graph classification based on optimizing graph spectra. In: Pfahringer, B., Holmes, G., Hoffmann, A. (eds.) DS 2010. LNCS (LNAI), vol. 6332, pp. 205–220. Springer, Heidelberg (2010). doi:10.1007/978-3-642-16184-1_15
15. Wale, N., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. In: Proceedings of the IEEE International Conference on Data Mining (ICDM), Hong Kong, pp. 678–689 (2006)