Hadamard Code Graph Kernels for Classifying Graphs

Tetsuya Kataoka and Akihito Inokuchi
School of Science and Technology, Kwansei Gakuin University, 2-1 Gakuen, Sanda, Hyogo, Japan

Keywords: Graph Classification, Support Vector Machine, Graph Kernel, Hadamard Code.

Abstract: Kernel methods such as Support Vector Machines (SVMs) are becoming increasingly popular because of their high performance on graph classification problems. In this paper, we propose two novel graph kernels called the Hadamard Code Kernel (HCK) and the Shortened HCK (SHCK). These kernels are based on the Hadamard code, which is used in spread spectrum-based communication technologies to spread message signals. The proposed graph kernels are equivalent to the Neighborhood Hash Kernel (NHK), one of the fastest graph kernels, and comparable to the Weisfeiler-Lehman Subtree Kernel (WLSK), one of the most accurate graph kernels. The fundamental performance and practicality of the proposed graph kernels are evaluated using three real-world datasets.

1 INTRODUCTION

A natural way of representing structured data is to use graphs (Vinh, et. al, 2010). 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 raised great interest, and research in the field has increased rapidly in recent years. Furthermore, because the need for classifying graphs has increased in many real-world applications, e.g., the analysis of proteins in bioinformatics and chemical compounds in cheminformatics (Schölkopf, et. al, 2004), graph classification has also 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 Support Vector Machine (SVM) are becoming increasingly popular because of their high performance on graph classification problems (Kashima, et. al, 2003). 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 very high and this approach includes the subgraph isomorphism matching problem that is known to be NP-complete (Garey and Johnson, 1979), kernels deliberately avoid the explicit computation of feature values and instead employ efficient procedures.

One representative graph kernel is the Random Walk Kernel (RWK) (Schölkopf and Smola, 2002; Kashima, et. al, 2003), 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) (Hido and Kashima, 2009) and the Weisfeiler-Lehman Subtree Kernel (WLSK) are two other recently proposed kernels that compute $k(g_i, g_j)$ faster than RWK. The NHK uses logical operations such as exclusive-OR on the label set of adjacent vertices, while the 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 very well with benchmark data consisting of graphs.

The computation of NHK is very efficient because its computation is a logical operation between fixed-length bit strings and does not require any string

Kataoka, T. and Inokuchi, A.
Hadamard Code Graph Kernels for Classifying Graphs.
DOI: 10.5220/0005634700240032
In *Proceedings of the 5th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2016)*, pages 24-32
ISBN: 978-989-758-173-1
Copyright © 2016 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

sorting. However, its drawback is hash collision, which occurs when different induced subgraphs have an identical hash value. Although WSLK must sort the vertex labels, it has high expressiveness because each vertex v has a distribution of vertex labels within i steps from v. To overcome these drawbacks, in this paper, we propose a novel graph kernel that is equivalent with NHK in terms of time and space complexities and comparable to WLSK in term of expressiveness. The graph kernel proposed in this paper is based on the Hadamard code. The Hadamard code is used in spread spectrum-based communication technologies such as Code Division Multiple Access (CDMA) to spread message signals. Because the probability of occurrences 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 the binomial distribution with zero mean under a certain assumption. Therefore, the expected value of the label is 0, and for such labels, a large memory space is not required. This characteristic is used to compress vertex labels in graphs, enabling the proposed graph kernel to be computed quickly.

The rest of this paper is organized as follows. In Section 2, we define the graph classification problem and explain the framework of the existing graph kernels. In Section 3, we propose 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. In Section 4, we provide a theoretical discussion of the effect of overflow on the proposed graph kernel. In Section 5, the fundamental performance and practicality of the proposed method are demonstrated through experiments. Finally, we conclude the paper in Section 6.

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 represented as $V(g)$. Although we assume that only the vertices in the graphs have labels in this paper, the methods 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 called simple if and only if the path does not have repeating vertices. Paths in this paper are not always simple. Given two graphs $g = (V, E, L, \ell)$ and $g' = (V', E', L', \ell')$, g' is called a subgraph of g, if there exists an injective function $\varphi : V' \to V$ that satisfies the following three conditions for $\forall v, v_1, v_2 \in V'$.

1. $(\varphi(v_1), \varphi(v_2)) \in E$, if $(v_1, v_2) \in E'$,
2. $\ell'(v) = \ell(\varphi(v))$,
3. $\ell'((v_1, v_2)) = \ell((\varphi(v_1), \varphi(v_2)))$.

Additionally, a subgraph g' of g is an "induced subgraph," where $\varphi(v_1)$ and $\varphi(v_2)$ are adjacent in g if and only if v_1 and v_2 in $V(g')$ are adjacent in g'.

The graph classification problem is defined as follows. Given a set of n training examples $D = \{(g_i, y_i)\}$ $(i = 1, \cdots, n)$, where each example is a pair consisting of a labeled graph g_i and the class $y_i \in \{+1, -1\}$ to which it belongs, the objective is to learn a function f that correctly predicts the classes of the test examples.

In this paper, graphs are classified by a 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

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

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 follows.

$$\begin{aligned} k'(g_i, g_j) &= \phi(g_i)^T \phi(g_j) \\ &= \sum_{v_i \in V(g_i)} \sum_{v_j \in V(g_j)} \delta(\ell(v_i), \ell(v_j)), \end{aligned}$$

where δ is the Kronecker delta.

Given a $g^{(h)} = (V, E, \Sigma, \ell^{(h)})$, a procedure to convert $g^{(h)}$ 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

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

Because k is a summation of semi-positive definite kernels, k is also semi-positive definite (Cristianini and Taylor, 2000).

Recently, various graph kernels have been applied to the graph classification problem. Representative graph kernels such as the 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 an identical label 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 follows.

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

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 by 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 an identical label 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.

We next describe the 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,

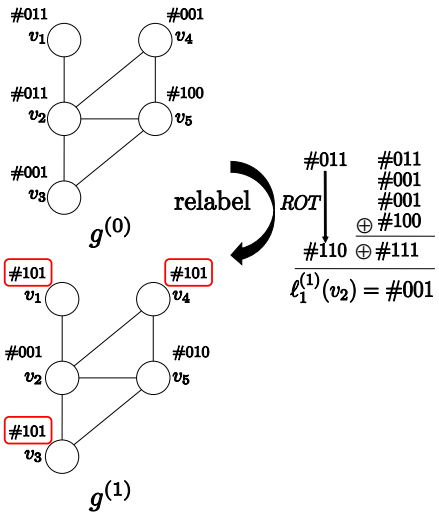


Figure 1: Relabeling $g^{(0)}$ to $g^{(1)}$ in NHK.

$\ell_2^{(h)}(v)$ 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 to obtain 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 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, then concatenates these labels. In $g^{(1)}$, v_3 has label BAC, meaning that v_3 has label B in $g^{(0)}$ and two adjacent vertices whose labels are A and C.

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, $\ell_3^{(0)}(v)$ 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 0. In LAK, $\ell_3^{(h)}(v)$ is defined as

$$\ell_3^{(h)}(v) = \ell_3^{(h-1)}(v) + \sum_{u \in N(v)} \ell_3^{(h-1)}(u).$$

[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 that WLSK requires are reduced.

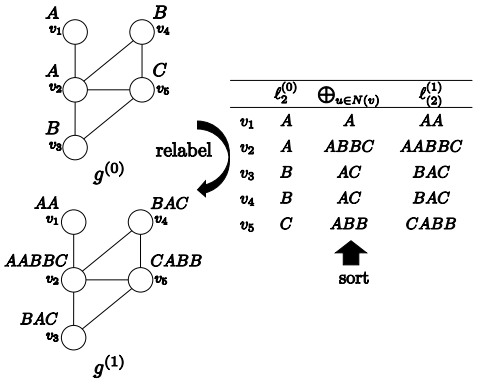


Figure 2: Relabeling $g^{(0)}$ to $g^{(1)}$ in WLSK.

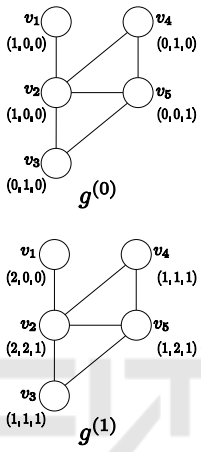


Figure 3: Relabeling $g^{(0)}$ to $g^{(1)}$ in LAK.

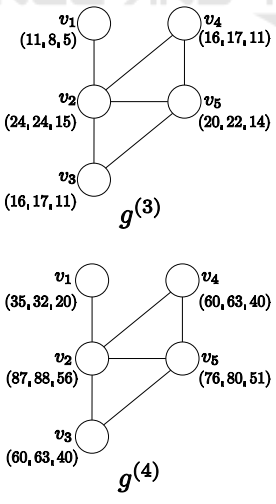


Figure 4: Relabeling $g^{(3)}$ to $g^{(4)}$ in LAK.

The i-th element in $\ell_3^{(h)}(v)$ is the frequency of occurrence of character σ_i in the string $\ell_2^{(h)}(v)$ concatenated by WLSK. Therefore, $\ell_3^{(h)}(v)$ has information on the distribution of labels within h steps from v. Therefore, 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 $\ell_3^{(h)}(v)$ 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. Thus, LAK requires a large amount of memory space.

Table 1: Graph Kernel Characteristics.

	advantages	drawbacks
NHK	computation time	hash collision
WSLK	expressiveness	computation time
LAK	expressiveness & computation time	memory space

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 $\ell_3^{(h)}(v)$ when relabeling $g^{(3)}$ to $g^{(4)}$ is large.

2.2 Existing Graph Kernel Drawbacks

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. Although WSLK requires vertex label sorting, it has high expressiveness because $\ell_2^{(h)}(v)$ contains the distribution of the vertex labels within h' steps $(0 \leq h' \leq h)$ from v. LAK requires a large amount of memory space to store vectors for high h although it does not require label sorting. To overcome these drawbacks, 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 Hademard matrix is a square $(-1,1)$-matrix in which any two row vectors are orthogonal, defined as follows:

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (1)$$

$$H_{2^k} = \begin{pmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{pmatrix} \quad (2)$$

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 the HCK as follows.

HCK: Let H be a Hadamard matrix of order $2^{\lceil \log_2 |\Sigma| \rceil}$ and $\ell_4^{(0)}(v)$ 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. Then $\ell_4^{(h)}(v)$ is defined as follows.

$$\ell_4^{(h)}(v) = \ell_4^{(h-1)}(v) + \sum_{u \in N(v)} \ell_4^{(h-1)}(u).$$

When ℓ_{σ_i} is a Hadamard code for a vertex label σ_i, $\ell_{\sigma_i}^T \ell_4^{(h)}(v)/|H|$ is the occurrence of σ_i in a string $\ell_2^{(h)}(v)$ generated by WLSK. Therefore, HCK 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 can obtain the distribution of the vertex labels within one step of v using the following calculation:

$$\begin{aligned} & \frac{1}{|H|} H \ell_4^{(1)}(v_5) \\ = & \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 4 \\ 0 \\ 2 \\ -2 \end{pmatrix} \\ = & \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \end{pmatrix}. \end{aligned}$$

That is, 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 $\ell_3^{(1)}(v_5)$, as shown in Fig. 3. The reason why we divide $H\ell_4^{(h)}(v)$ by four is that the order of the Hadamard matrix used is $|H| = 4$.

If each element in $\ell_4^{(h)}(v)$ is stored in four bytes (the commonly used size of integers in C, Java, and other languages) the space complexity of HCK is equivalent to LAK. Therefore, we have not overcome the drawback of LAK yet. In this paper, we assume that each vertex label is assigned to a vertex with equal probability. Because the probability of occurrence of 1 and -1 are equivalent in the each column in the Hadamard matrix except for the first column, the i-th element ($1 < i \leq |\Sigma|$) in $\ell_4^{(h)}(v)$ follows a binomial distribution with zero mean under this assumption. Therefore, the expected value of the element in $\ell_4^{(h)}(v)$ is 0, and for the elements, a large

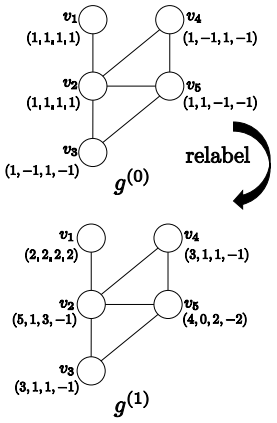


Figure 5: Relabeling $g^{(0)}$ to $g^{(1)}$ in HCK.

memory space is not required. For example, Tables 2 and 3 represent values of the i-th elements in $\ell_3^{(h)}(v_2)$ and $\ell_4^{(h)}(v_2)$, respectively, in a graph $g^{(h)}$, when $g^{(0)}$ (shown in Fig. 6) is relabeled iteratively h times. Under this assumption of vertex label probability, the expected value of all elements in $\ell_4^{(h)}(v_2)$ except for the first element becomes 0. The first element represents the number of paths from v_2 to the vertices reachable within one step. Based on this observation, we assign bit arrays of length ρ in the L bit array to the elements as follows.

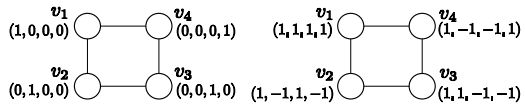
SHCK: Similar to NHK, $\ell_5^{(0)}(v)$ is a fix-length bit array of length L. The bit array is divided into $|H|$ fragments, one of which is a bit array of length $L - \rho(|H| - 1)$ and the rest are bit arrays of length ρ. The first fragment of length $L - \rho(|H| - 1)$ is assigned to store the first element of $\ell_4^{(0)}(v)$, the next 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 $\ell_4^{(0)}(v)$ is represented by its one'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 to LAK, if overflow of the fix-length bit array does not occur.

Table 2: Elements in a label in LAK.

h	Label				
0	$\ell_3^{(0)}(v_2)=($	0	1	0	0)
1	$\ell_3^{(1)}(v_2)=($	1	1	1	0)
2	$\ell_3^{(2)}(v_2)=($	2	3	2	2)
3	$\ell_3^{(3)}(v_2)=($	7	7	7	6)
4	$\ell_3^{(4)}(v_2)=($	20	21	20	20)
5	$\ell_3^{(5)}(v_2)=($	61	61	61	60)
6	$\ell_3^{(6)}(v_2)=($	182	183	182	182)
7	$\ell_3^{(7)}(v_2)=($	547	547	547	546)
8	$\ell_3^{(8)}(v_2)=($	1640	1641	1640	1640)
9	$\ell_3^{(9)}(v_2)=($	4921	4921	4921	4920)
10	$\ell_3^{(10)}(v_2)=($	14762	14763	14762	14762)



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

Figure 6: Relabeled graphs.

Table 3: Elements in a label in HCK.

h	Label				
0	$\ell_4^{(0)}(v_2)=($	1	-1	-1	1)
1	$\ell_4^{(1)}(v_2)=($	3	-1	-1	-1)
2	$\ell_4^{(2)}(v_2)=($	9	-1	-1	1)
3	$\ell_4^{(3)}(v_2)=($	27	-1	-1	-1)
4	$\ell_4^{(4)}(v_2)=($	81	-1	-1	1)
5	$\ell_4^{(5)}(v_2)=($	243	-1	-1	-1)
6	$\ell_4^{(6)}(v_2)=($	729	-1	-1	1)
7	$\ell_4^{(7)}(v_2)=($	2187	-1	-1	-1)
8	$\ell_4^{(8)}(v_2)=($	6561	-1	-1	1)
9	$\ell_4^{(9)}(v_2)=($	19683	-1	-1	-1)
10	$\ell_4^{(10)}(v_2)=($	59049	-1	-1	1)

4 SHCK OVERFLOW

As explained in the previous section, in SHCK, the fixed-length bit array L is divided into small fragments, each of which corresponds to an element in $\ell_4^{(h)}(v)$. We sum such bit arrays to relabel vertices. Because all elements in $\ell_4^{(h)}(v)$ except for the first element are represented as a bit array of length ρ, we face the possibility of overflow when iteratively summing up these bit arrays. In this section, we theoretically discuss the probability of overflow in SHCK.

Let x_i^k be the i-th element in $\ell_4^{(h)}(v)$, which is the label of vertex v and is a value generated by summing up the base Hadamard codes k times. If $i=1$, $x_i^k=k$. For $i\neq 1$, if $-2^\rho \leq x_i^k \leq 2^\rho-1$, x_i^k fits in a fragment of length ρ without overflowing. Let $p(k,j)$ be the probability that the value of x_i^k is j and x_i^k fits in a bit fragment of length ρ without overflowing. Under the assumption that the probability of any label existing on a vertex is uniform, when $k=1$,

$$p(k,j)=\begin{cases} 1/2 & if\ j=1,\\ 1/2 & if\ j=-1,\ and\\ 0 & otherwise,\end{cases}$$

because an element in the Hadamard matrix is either 1 or -1. If x_i^k fits in a bit array of length ρ without overflowing, x_i^{k-1} also fits in the array. In contrast, if x_i^k cannot fit in a bit array of length ρ without overflowing, x_i^{k+1} also cannot fit in the array. Overflow occurs when x_i^k is $2^{\rho-1}$ and $+1$ sum to x_i^k or when x_i^k is -2^ρ and -1 sums to x_i^k. Therefore, $p(k,j)$ is introduced by the following recurrence formula.

$$p(k,j)=\begin{cases} \frac{1}{2}p(k-1,j-1) & if\ j=2^\rho-1,\\ \frac{1}{2}p(k-1,j+1) & else\ if\ j=-2^\rho,\\ \frac{1}{2}p(k-1,j+1)+\frac{1}{2}p(k-1,j-1) & \\ \qquad else\ if\ -2^\rho<j<2^\rho-1,\\ 0 \qquad otherwise.\end{cases}$$

Accordingly, $p(k)$, which is the probability that x_i^j fits in a bit array of length ρ without overflowing is

$$p(k)=\sum_{j=-2^\rho}^{2^\rho-1} p(k,j).$$

After h relabels of a graph in which the average degree is d, x_i^k is a value that is a summation of $k=(d+1)^h$ binary values. The probability $p(\rho,d,h)$ that overflow does not occur for ρ, d, and h is

$$p(\rho,d,h)=\sum_{j=-2^\rho}^{2^\rho-1} p\left((d+1)^h,j\right). \qquad (3)$$

When h increases, $p(\rho,d,h)$ becomes very small. Nevertheless, in the next section, we demonstrate that the proposed graph kernel SHCK has the ability to classify graphs with high accuracy.

5 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

Table 4: Summary of evaluation datasets.

	MUTAG	ENZYMES	D&D
Number of graphs $\|D\|$	188	600	1178
Maximum graph size	84	126	5748
Average graph size	53.9	32.6	284.3
Number of labels $\|\Sigma\|$	12	3	82
Number of classes (class distribution)	2 (126,63)	6 (100,100,100,100,100,100)	2 (487, 691)
Average degree of vertices	2.1	3.8	5.0

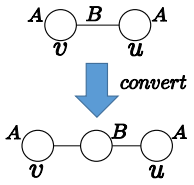


Figure 7: Conversion of a graph.

time and accuracy of the prediction performance of HCK and SHCK with those of HNK and WLSK. To learn from the kernel matrices generated by the above graph kernels, we used the LIBSVM package[2] using 10-fold cross validation.

We used three real-world datasets. The first dataset, MUTAG (Debnath, et. al, 1991), contains information on 188 chemical compounds and their class labels. The class labels are binary values that indicate the mutagenicity of 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 (Schomburg, et. al, 2004). The third dataset, D&D, contains information on 1178 protein structures, in which each amino acid corresponds to a vertex and two vertices are connected by an edge if they are less than 6 Ångstroms apart (Dobson and Doig, 2003). 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 article (Hido and Kashima, 2009), that is, an edge labeled with ℓ that is adjacent to vertices v and u in a chemical graph is replaced with a vertex labeled with ℓ that is adjacent to v and u with unlabeled edges, as shown in Fig. 7. Table 4 summarizes the datasets.

5.1 Scalability

Figures 8, 9, and 10 show the computation time required to obtain a graph $g^{(h)}$ from a graph $g^{(0)}$ in

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

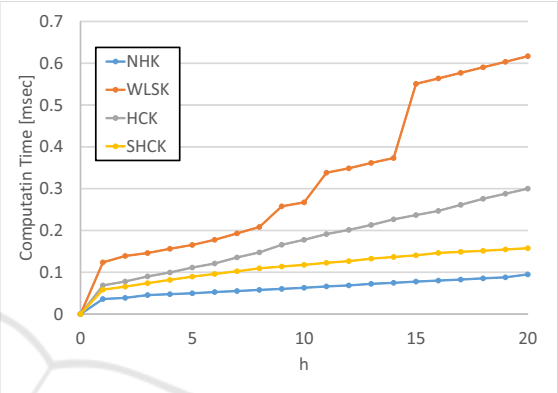


Figure 8: Computation time for various h (MUTAG).

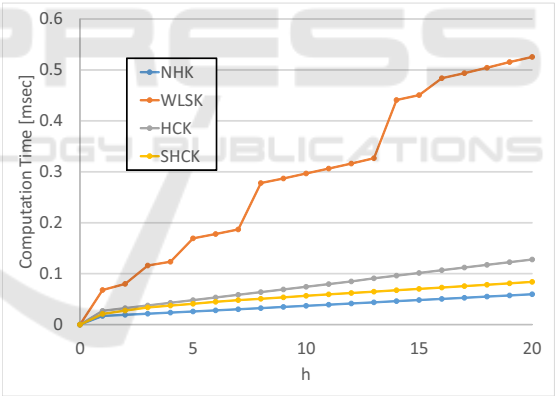


Figure 9: Computation time for various h (ENZYMES).

NHK, WLSK, HCK, and SHCK for various h for the MUTAG, ENZYMES, and D&D datasets, respectively. As shown in the figures, NHK and SHCK are faster than HCK, and much faster than WLSK. Additionally, the computation time of NHK, HCK, and SHCK increases linearly when h is increased. The reason why WLSK requires such a large amount of 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,

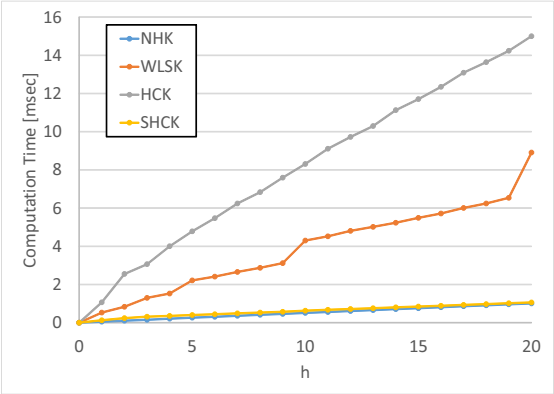


Figure 10: Computation time for various h (D&D).

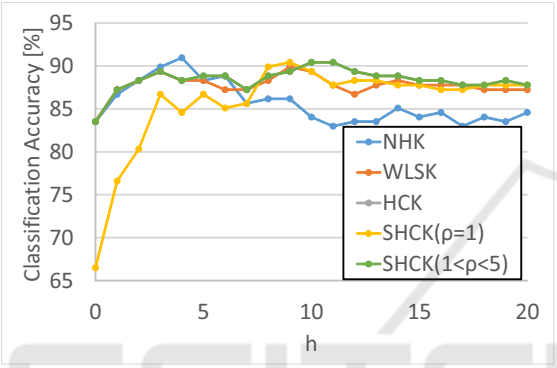


Figure 11: Classification accuracy for various h and ρ (MUTAG).

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 small, WLSK requires further computation time when the average degree of the data increases. HCK requires a large amount of computation time for the D&D dataset because the number of labels in the dataset is large and its computation time is proportional to the number of labels.

5.2 Classification Accuracy

Figure 11 shows the classification accuracy of NHK, WLSK, HCK, and SHCK for various h and ρ for the MUTAG dataset. Their maximum accuracies for various h are almost the same. When $h=0$, the accuracy for SHCK ($\rho=1$) is very low, because 1 or -1 (the values in the Hadamard matrix) cannot be stored as a one'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 12 shows the classification accuracy of NHK, WLSK,

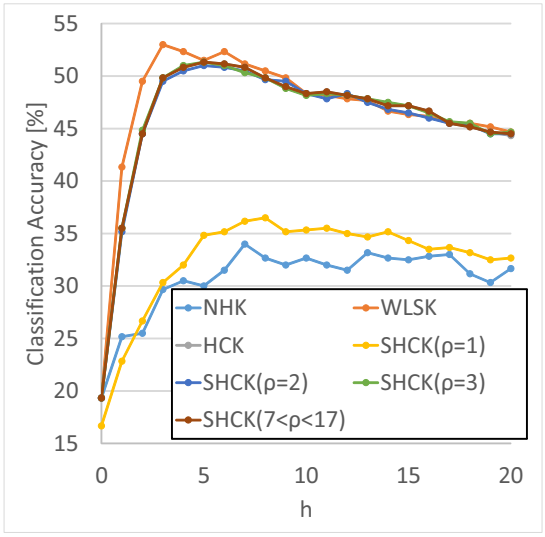


Figure 12: Classification accuracy for various h and ρ (ENZYMES).

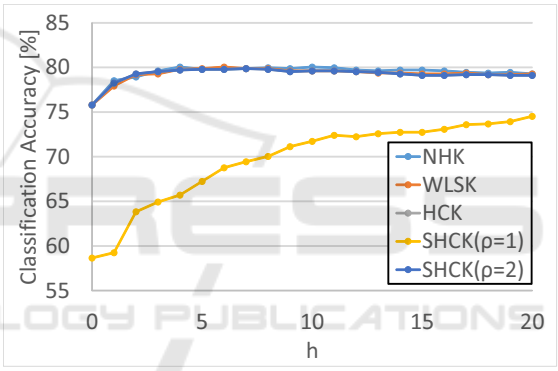


Figure 13: Classification accuracy for various h and ρ (D&D).

HCK, and SHCK for various h and ρ for the ENZYMES dataset. The accuracy of WLSK is slightly superior to those of HCK and SHCK ($\rho=2$, $\rho=3$, and $7<\rho<17$), and their accuracies are much superior to those of 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 of that of SHCK for low ρ ($\rho=2$ and $\rho=3$). The maximum accuracy of WLSK is 53.0%, while the maximum accuracy of both HCK and SHCK ($\rho=3$, 4, and $7<\rho<17$) is 51.3%. The reason why the accuracy of WLSK is slightly superior to that of HCK is that $\ell_2^{(h)}(v)$ contains information on the distribution of labels at h steps from v, while $\ell_4^{(h)}(v)$ 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

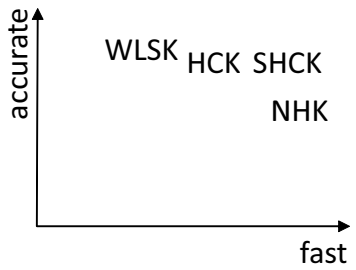


Figure 14: Qualitative performance of evaluated graph kernels.

and SHCK. When ρ is increased up to 16, the length of a bit string to store the first element of $\ell_4^{(h)}(v)$ 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 $\ell_4^{(h)}(v)$ has absolutely no impact on classification accuracy. Figure 13 shows the classification accuracy of NHK, WLSK, HCK, and SHCK for various h and ρ for the D&D dataset. All accuracies except for that of SHCK ($\rho = 1$) are almost equivalent.

6 CONCLUSION

In this paper, we proposed graph kernels based on the Hadamard code to classify graphs. Figure 14 presents a qualitative description of the performance of graph kernels in terms of computation time and classification accuracy. These experimental results show that the proposed graph kernel SHCK is fast and accurate.

REFERENCES

Borgwardt, Karsten M., Cheng, Soon Ong, Schonauer, Stefan, Vishwanathan, S. V. N., Smola, Alex J., and Kriegel, Hans-Peter. 2005. Protein Function Prediction via Graph Kernels. *Bioinfomatics 21 (suppl 1)*: 47–56.

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.

Cristianini, Nello, and Shawe-Taylor, John. 2000. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press.

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.

Dobson, Paul D., and J. Doig, Andrew. 2003. Distinguishing Enzyme Structures from Non-enzymes Without Alignments. *Journal of Molecular Biology* 330(4): 771–783.

Garey, Michael R., and Johnson, David S.. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman.

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

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.

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.

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

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

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.

Vinh, Nguyen Duy, Inokuchi, Akihiro, and Washio, Takashi. 2010. Graph Classification Based on Optimizing Graph Spectra. In Proc. of the International Conference on Discovery Science. 205–220.