การประยุกต์เครื่องเวกเตอร์สนับสนุนด้วยการเชื่อมโยงของโครงสร้างสารประกอบชีวเคมีปฐมฐานเพื่อระบุลักษณะเป้าหมายของโมเลกุล

นายภีมพัฒน์ วงศ์ศรีพิสันต์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาคณิตศาสตร์ประยุกต์และวิทยาการคณนา
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2562

APPLYING SUPPORT VECTOR MACHINE WITH CONNECTIVITY OF PRIMITIVE BIOCHEMICAL COMPOUND STRUCTURE TO IDENTIFY TARGET CHARACTERISTICS OF MOLECULES

Mr. Peemapat Wongsriphisant

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Applied Mathematics and Computational Science

Department of Mathematics and Computer Science

Faculty of Science

Chulalongkorn University

Academic Year 2019

Thesis Title	APPLYING SUPPORT VECTOR MACHINE WITH CONNECTIVITY OF PRIMITIVE BIOCHEMICAL COMPOUND STRUCTURE TO IDENTIFY TARGET CHARACTERISTICS OF MOLECULES
By	Mr. Peemapat Wongsriphisant
Field of Study	Applied Mathematics and Computational Science
Thesis Advisor	Assistant Professor Kitiporn Plaimas, Ph.D.
Thesis Co-advisor	Professor Chidchanok Lursinsap, Ph.D.

Accepted by the Faculty of Science, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree

ภีมพัฒน์ วงศ์ศรีพิสันต์ : การประยุกต์เครื่องเวกเตอร์สนับสนุนด้วยการเชื่อมโยงของโครงสร้างสารประกอบชีวเคมีปฐมฐานเพื่อระบุลักษณะเป้าหมายของโมเลกุล. (APPLYING SUPPORT VECTOR MACHINE WITH CONNECTIVITY OF PRIMITIVE BIOCHEMICAL COMPOUND STRUCTURE TO IDENTIFY TARGET CHARACTERISTICS OF MOLECULES) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ผศ.ดร.กิติพร พลายมาศ, อ.ที่ปรึกษาวิทยานิพนธ์ร่วม : ศ.ดร.ชิดชนก เหลือสินทรัพย์ 64 หน้า.

ข้อมูลในรูปของกราฟเชิงโครงสร้างมีการใช้อย่างแพร่หลายในหลากหลายสาขาวิชา ในการศึกษาเกี่ยวกับเคมีสารสนเทศศาสตร์ ข้อมูลของสารประกอบทางชีวเคมีสามารถเก็บอยู่ในรูปของกราฟ โดยที่จุดยอดแสดงถึงอะตอม และเส้นเชื่อมระหว่างจุดแสดงถึงพันธะทางเคมีระหว่างอะตอม ด้วยการเก็บข้อมูลในรูปแบบดังกล่าว เราสามารถทำการทำนายลักษณะเป้าหมายของสารประกอบทางชีวเคมีที่เราไม่รู้จักได้ โดยการพิจารณาความคล้ายคลึงกันระหว่างกราฟของสารประกอบแต่ละชนิด ปัญหาดังกล่าวมีชื่อเรียกว่า การจำแนกประเภทข้อมูลประเภทกราฟ ในงานวิจัยนี้ เราได้ทำการเสนอขั้นตอนวิธีในการจำแนกข้อมูลของกราฟสารประกอบชีวเคมี โดยการสกัดโครงสร้างพื้นฐานของสารประกอบ แล้วทำการสร้างกราฟใหม่ขึ้นจากโครงสร้างพื้นฐานเหล่านั้น จากนั้นนำค่าความคล้ายคลึงกันของกราฟใหม่ที่ได้มาทำการจำแนกด้วยเครื่องเวกเตอร์สนับสนุน เราได้ทำการทดสอบขั้นตอนวิธีนี้กับชุดข้อมูลของสารประกอบชีวเคมีมาตรฐานทั้งหมด 4 ชุด กับวิธีการหาค่าความ คล้ายคลึงกันระหว่างกราฟ 4 แบบ โดยได้ค่าความแม่นยำเฉลี่ยที่ 84.7% นอกจากนี้ประสิทธิภาพของวิธีที่ได้นำเสนอยังสูงกว่าการจำแนกข้อมูลโดยใช้แค่กราฟเคอร์เนลเพียงอย่างเดียวอีกด้วย โดยสรุปแล้ววิธีการสกัดโครงสร้างพื้นฐานที่ได้เสนอนี้ เป็นเครื่องมือที่มีประโยชน์ในการช่วยสกัดข้อมูลเชิงโครงสร้างเพื่อนำไปใช้ในการจำแนกข้อมูลของสารประกอบชีวเคมีได้

ภาควิชา	คณิตศาสตร์และ	ลายมือชื่อนิสิต ....
	วิทยาการคอมพิวเตอร์	ลายมือชื่อ อ.ที่ปรึกษาหลัก ....
สาขาวิชา	คณิตศาสตร์ประยุกต์	ลายมือชื่อ อ.ที่ปรึกษาร่วม..................
	และวิทยาการคณนา	
ปีการศึกษา	2562	

6172036223 : MAJOR APPLIED MATHEMATICS AND COMPUTATIONAL SCIENCE
KEYWORDS : GRAPH CLASSIFICATION/ BIOCHEMICAL COMPOUNDS

PEEMAPAT WONGSRIPHISANT : APPLYING SUPPORT VECTOR MACHINE WITH CONNECTIVITY OF PRIMITIVE BIOCHEMICAL COMPOUND STRUCTURE TO IDENTIFY TARGET CHARACTERISTICS OF MOLECULES. ADVISOR : ASST. PROF. KITIPORN PLAIMAS, Ph.D., THESIS COADVISOR : PROF. CHIDCHANOK LURSINSAP, Ph.D., 64 pp.

Graph-structured data are widely used in many fields of study. For example, in cheminformatics, the data of biochemical compounds can be encoded as a compound graph where atoms are represented by vertices and chemical bonds between pairs of atoms are represented by edges. By this setting, we can predict the molecular characteristics of an unseen biochemical compound by considering the similarity between two compound graphs. This problem is called a graph classification. This study, proposes an algorithm for compound graph classification by finding primitive substructures of the compounds and encode them in a new graph form. Then, their similarities are computed and fed to a support vector machine. The proposed algorithm was evaluated on four real-world data sets with four similarity measures, yielding the average accuracy of 84.7%. Furthermore, the performance of the algorithm was also better than the classification method that solely uses graph kernels. In conclusion, the extraction of primitive structures can be a great tool for extracting compound structure features in biochemical compound classification.

Department	: Mathematics and Computer Science	Student's Signature	....
		Advisor's Signature	....
Field of Study	: Applied Mathematics and Computational Science	Co-advisor's Signature	....
Academic Year	: 2019		

ACKNOWLEDGEMENTS

This thesis was completed with kind support and help from many individuals.

First, I would like to express my gratitude to the Science Achievement Scholarship of Thailand (SAST) for my scholarship as well as the Department of Mathematics and Computer science, Faculty of Science, Chulalongkorn University, and Advanced Virtual and Intelligent Computing Center (AVID) for their support.

I would like to thank my advisor and co-advisors, Assistant Professor Dr. Kitiporn Plaimas, Professor Dr. Chidchanok Lursinsap, and Assistant Professor Dr. Apichat Suratanee including my thesis committee, Assistant Professor Dr. Krung Sinapiromsaran, Assistant Professor Dr. Boonyarit Intiyot, and Assistant Professor Dr. Saichon Jaiyen for guidance and advice to make this study into success.

I also would like to thank people in my research group, Miss Pajaree Songsungsan, Miss Satanat Kitsiranuwat, Mr. Thitipong Kawichai, Miss Panisa Janyasupab, Mr. Thummarat Paklao and Mr. Pramod Kumar for many useful discussions.

Finally, I would like to thank my friends and family for constant support and inspiration.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This chapter gives information about the problem description, the motivation, the related works, and the objective of this study.

1.1 Motivation and Literature Surveys

In recent years, the graph-structured data recieved increasing attention from researchers and entrepreneurs, as well as many organizations such as Oracle and Neo4j try to focus on developing the graph databases. The advantage of graphs is that it contains information about the relationships between each data point which is hard to explain with traditional relational data. Therefore, the graph-structured data is widely used in many fields of study such as social network analysis, fraud detection, recommendation system, bioinformatics, and cheminformatics.

In cheminformatics, the data of biochemical compounds can be encoded as graphs where the vertex of the graph is an atom and the edge of the graph depicts chemical bond between a pair of atoms. There is an important problem based on this representation of the biochemical compound, which is to predict the molecular function of an unseen biochemical compound under the assumption that biochemical compounds with similar structures will have similar molecular functions. The problem of classifying the graph-structured data with its similarity is called a graph classification.

The graph classification method is used to classify graph-structured data. A traditional graph classification method has two main steps. First, transforming a graph into a vector space that preserves the graph structure properties by using graph embedding techniques or calculating a similarity between each pair of graphs by using graph similarity measures. Second, machine learning such as a support vector machine, a decision tree, or a neural network is applied to classify each graphs from its feature vector or its similarity.

Many classification methods have been proposed. In the 2000s, there was a method based on a graph kernel, by using a kernel function for calculating similarities between a pair of graphs and classifying them with a kernel method. Traditional graph kernel can be categorized into three classes[1]: (1) a graph kernel based on walks and paths such as the random walk kernel (RW)[2], and the shortest-path kernel (SP)[3], (2) a graph kernel based on limited-size subgraphs such as the graphlet count kernel (GK)[4], and (3) a graph kernel based on subtree patterns such as the subtree kernel (ST)[5]. However, these methods take such high computational time when calculating the kernel function on the large graph. In 2011, N. Shervashidze et al. proposed the Weisfeiler-Lehman framework (WL) [1] which was inspired by the Weisfeiler-Lehman test of graph isomorphism [4]. This framework was operated with another graph kernel. The popular kernel with this framework was the Weisfeiler-Lehman subtree kernel (WL-ST) which was a Weisfeiler-Lehman framework operating with a vertex histogram kernel. This kernel had better computational complexity compared to the original subtree kernel. Therefore, it can be easily performed for a huge graph. Another interesting Weisfeiler-Lehman kernel was the Weisfeiler-Lehman shortest-path kernel (WL-SP) which had better performance than the original shortest-path kernel on some data sets.

The using of graph embedding techniques was performed for a graph classification task. In 2016, M. Niepert et al.[6] proposed the Patchy-San technique for graph embedding and applied convolutional neural networks (CNN) for the classification, so-called PSCN. In the following year, A. Narayanan et al.[7] proposed a whole-graph embedding technique, namely graph2vec for converting each graph into a vector. After that, these vector features were then fed to the support vector machine. In the same year, 2D convolutional neural networks (2DCNN) was proposed by A. Tixier et al.[8]. They employed a node embedding technique based on a random walk, namely node2vec [9], and converted feature vectors into an image form by using a 2D histogram and then classified the data with CNN.

However, these methods have some downsides. A graph kernel decomposes a graph into substructures then counts the number of occurrence of these substructures. Thus, a graph classification method based on graph kernels has some problems when it is used on graphs with the same local properties, but different global structures. The problem of the method based on graph embedding is its complication. For example, the classification method depending on node2vec has many steps such as sampling walks from a graph and then these walks are used as input data for the skip-gram neural network to embed each node into a vector space, which after that each node will be grouped to create a 2D histogram. Then, these 2D histograms are fed to a convolutional neural network.

In this study, we focus on developing a classification method on biochemical compounds. Biochemical compounds compose of many identical molecules which are similar to a simple graph, for example, a ring of 6 carbons in benzene is similar to a cycle graph in graph theory, propane is similar to a star graph with 3 vertices, isobutane is similar to a star graph with 4 vertices. We assumed that the molecular function of compounds can be analyzed by using the set of common graph structures of the compounds.

We thus proposed a classification framework for biochemical compounds encoded as a graph by finding a common molecule or substructure that frequently appeared in every graph called “primitive structure”. After that, we recreated a new graph called “primitive structure graph” from these substructures. We calculated the similarities between a pair of graphs by various techniques, such as counting the number of edges between each type of the primitive structure, using graph kernels such as the Weisfeiler-Lehman subtree kernel which counts the number of common subtree patterns between two graphs. These similarity measures were fed into the machine learning technique called the “kernel method”. Finally, the best technique was reported and compared with the existing one.

1.2 Research Objectives

The objective of this research was to develop the method to classify the biochemical compounds based on the connectivity of primitive structures. First, the algorithm selecting the primitive structure from a set of candidate structures was applied. Then, these structures were extracted from a graph in the data set and the new graph based on the connection of these structures was created by our developed algorithm. The proposed classification method was implemented and experimented on the real-world dataset. Finally, the results were then compared with other methods using the accuracy, precision, recall, and F1-score as the performance measures.

1.3 Thesis Overview

The remainder of this thesis consists of the following chapters. Chapter II provides the relevant background of the graph, the graph kernel function, the kernel method, and the sub-graph isomorphism problem. In Chapter III, the definition of a primitive structure and the algorithms to classify the graph-structured data by the primitive structure is proposed. Chapter IV is the experiment results on the real-world data sets. The final chapter is the conclusion that provided the discussion and the conclusion of this work.

CHAPTER II

BACKGROUND KNOWLEDGE

The objective of this chapter is to describe the preliminaries of this thesis. Section 2.1 is the basic graph theory. Section 2.2 is a graph kernel function and its examples. Section 2.3 is a kernel method for classification. Section 2.4 is a sub-graph isomorphism problem.

2.1 Graph theory

These definitions are basic concepts from graph theory [10].

Definition 2.1.1 (Graph)**.** A graph $G = (V, E)$ consists of a set of vertices (or nodes) V and a set of edges (or links) $E \subseteq V \times V$ which an edge connect a pair of vertices. A graph in which edges are undirected is said to be undirected graph and a graph with directed edges is said to be the directed graph.

Definition 2.1.2 (Labeled graph)**.** A labeled graph is a graph $G = (V, E)$ endowed with a labeling function $l : V \cup E \to \sum$ that assigns labels to the vertices and edges of the graph from a discrete set of labels $\sum$.

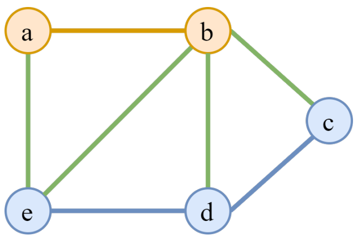


Figure 2.1: A labeled graph with 5 vertices where the label of each vertex (edge) is represented by their color.

Definition 2.1.3 (Degree)**.** Given an undirected graph $G = (V, E)$ and a vertex $v_i \in V$, the degree of v_i in G is the number of edges incident to v_i, defined as

$$deg_g(v_i) = |\{v_j : \{v_i, v_j\} \in E\}| \tag{2.1}$$

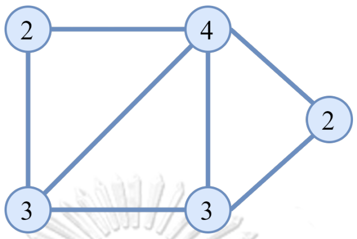


Figure 2.2: An undirected graph with 5 vertices where each vertex labeling with its degree.

Definition 2.1.4 (Graph isomorphism)**.** Let $G = (V, E)$ and $G' = (V', E')$ be labeled graphs with labeling function l and l', respectively. An isomorphism of graph is a bijective function $f : V \to V'$, such that

1 $\forall v_i \in V, f(v_i) \in V'$ and $l(v_i) = l'(f(v_i))$

2 $\forall \{v_i, v_j\} \in E, \{f(v_i), f(v_j)\} \in E'$ and $l(\{v_i, v_j\}) = l'(\{f(v_i), f(v_j)\})$

If there is an isomorphism between graphs G and G', then G and G' are isomorphic.

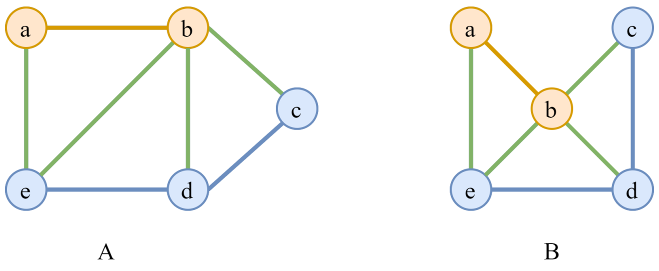


Figure 2.3: Example of graph isomorphism, graph A and graph B are isomorphic.

Definition 2.1.5 (Cycle graph)**.** A cycle graph C_n is a graph that consists of a single cycle with n vertices.

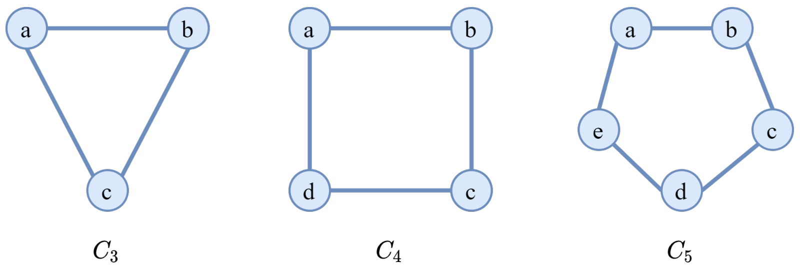


Figure 2.4: Example of cycle graphs.

Definition 2.1.6 (Star graph)**.** A star graph S_n is a graph that consists of $n+1$ vertices with one vertex having degree of n and the others having degree of 1.

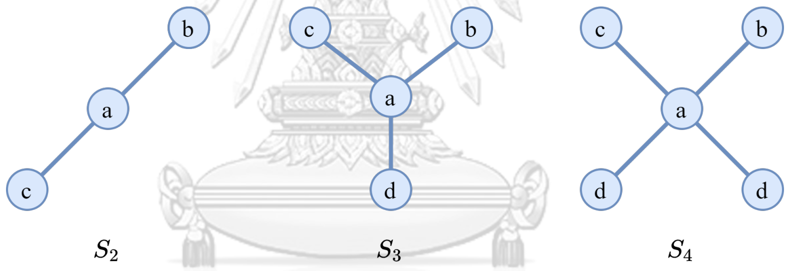


Figure 2.5: Example of star graphs.

Definition 2.1.7 (Path graph)**.** A path graph P_n is a graph that consists of n vertices with two terminal vertices (vertices that have degree of 1), while all others have degree of 2.

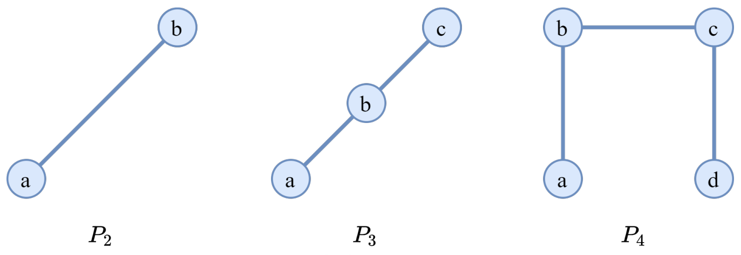


Figure 2.6: Example of path graphs.

Definition 2.1.8 (Shortest-path graph)**.** Let $G = (V, E)$ be a graph. A shortest-path graph of graph G is a graph with the same set of vertices V. Shortest-path graph's vertices are connected if there exists a walk between them in G, and shortest-path graph's edges are labeled with the shortest distance between its endpoints in G.

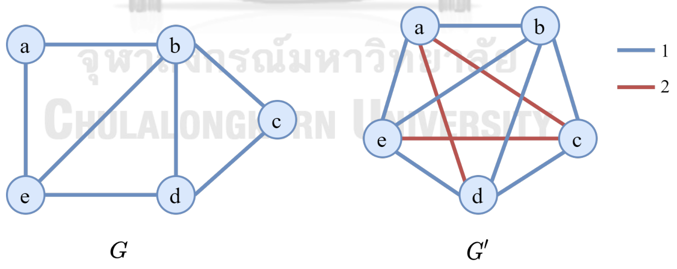


Figure 2.7: Graph G and its shortest-path graph G' where the blue edge represents the shortest-path with a distance equal to 1 and the red edge represents the shortest-path with a distance equal to 2.

2.2 Graph kernel function

The general solution to calculate the similarity between graphs is using a graph kernel. Most graph kernels are constructed by using the framework called R-convolution kernels [11]. Graphs will be decomposed into small substructures such as subgraph, path, and subtree with a feature mapping function. After that, the kernel will be computed by considering the pairwise similarity between these feature vectors. Let G and G' be graphs, then the graph kernel can be defined as follows:

$$K(G, G') = \langle \phi(G), \phi(G') \rangle \tag{2.2}$$

where ϕ is a feature mapping function on graphs based on each type of the graph kernel.

A graph kernel can also serves as a graph embedding technique by transforming a graph into its explicit feature mapping $\phi(G)$, for example, the feature mapping $\phi(G)$ of the graphlet kernel is the vector that each element is the number of each type of graphlets in the graph G.

In this study, the following graph kernels are deployed.

2.2.1 Shortest-path kernel

Definition 2.2.1 (Shortest-path kernel)**.** Let G, G' be graphs, and $S = (V, E)$, $S' = (V', E')$ be their corresponding shortest-path graph. The shortest-path kernel is defined as

$$K_{\mathrm{SP}}(G, G') = \sum_{e \in E} \sum_{e' \in E'} k^{(1)}_{\mathrm{walk}}(e, e'), \tag{2.3}$$

where $k^{(1)}_{\mathrm{walk}}(e, e')$ is a kernel on edge walks of length 1.

Let $e = \{v, u\} \in E$, $e' = \{v', u'\} \in E'$ and δ is the Dirac kernel, it is 1 when its arguments are equal, otherwise it is 0. Then, $k^{(1)}_{\mathrm{walk}}(e, e')$ is usually defined as

$$\begin{aligned} k^{(1)}_{\mathrm{walk}}(e, e') = {} & \delta(l(v), l(v'))\delta(l(e), l(e'))\delta(l(u), l(u')) \\ & + \delta(l(v), l(u'))\delta(l(e), l(e'))\delta(l(u), l(v')) \end{aligned} \tag{2.4}$$

2.2.2 Weisfeiler-Lehman kernel

In 2011, Nino Shervashidze proposed the Weisfeiler-Lehman framework for the graph kernel. This framework was inspired by the Weisfeiler-Lehman test of graph isomorphism. It calculated a similarity between graphs by considering on new graphs which were created with its "relabeling process".

Let G be a graph and $G_0 = G$. Let l_i be a labeling function of graph G after i^{th} relabeling process called G_i. Then, the i^{th} relabeling process of the Weifeiler-Lehman framework is defined as follow

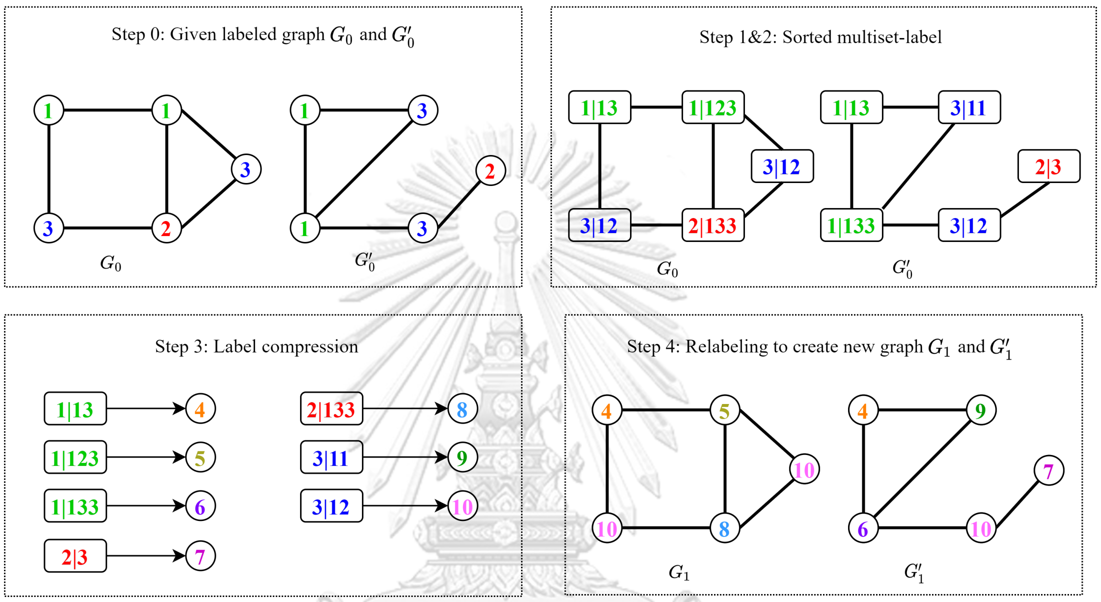
1 Assign a multiset-label $M_i(v)$ to each vertex v in G_{i-1}. $M_i(v) = \{l_{i-1}(u) | u \in \mathcal{N}(v)\}$ where $\mathcal{N}(v)$ is a set of neighbors of v.

2 Sort elements in $M_i(v)$ in ascending order and concatenate them into a string

$s_i(v)$. Then, add $l_{i-1}(v)$ as a prefix to $s_i(v)$.

3 Map each string $s_i(v)$ to a compressed label using function $f : \Sigma* \to \Sigma$ such that $f(s_i(v)) = f(s_i(u))$ if and only if $s_i(v) = s_i(u)$.

4 Let $G_i = G_{i-1}$. Then, set $l_i(v) := f(s_i(v))$ for all vertices in G_i.



Step 0: Given labeled graph G_0 and G'_0
G_0
G'_0
Step 1&2: Sorted multiset-label
1|13
1|123
3|12
2|133
3|11
1|133
2|3
Step 3: Label compression
Step 4: Relabeling to create new graph G_1 and G'_1
G_1
G'_1

Figure 2.8: Illustration of the Weifeiler-Lehman relabeling process at iteration $i = 1$ of labeled graph G and G'

Definition 2.2.2 (Weisfeiler-Lehman kernel)**.** Let G, G' be graphs and $i = 1, 2, \ldots, h$. Graphs G_i and G'_i are graphs obtained after i^{th} iteration of the Weisfeiler-Lehman relabeling process of graphs G and G', respectively. Let K_b be graph kernel called a base graph kernel. Then the Weisfeiler-Lehman kernel with h iterations is defined as

$$K_{\text{WL}}(G, G') = K_b(G_0, G'_0) + K_b(G_1, G'_1) + \cdots + K_b(G_h, G'_h). \tag{2.5}$$

2.2.2.1 Weisfeiler-Lehman sub-tree kernel

The Weisfeiler-Lehman sub-tree kernel is the Weisfeiler-Lehman kernel where the base graph kernel (K_b) is a vertex histogram kernel (VH). The vertex histogram kernel is a graph kernel based on the number of matching vertex labels between each graph. The vertex histogram kernel is defined as

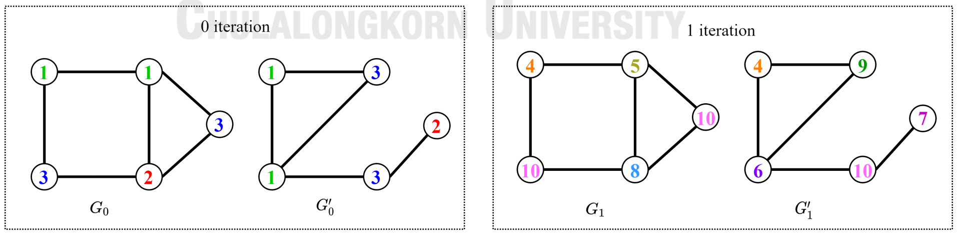
$$K_{\mathrm{VH}}(G, G') = \sum_{v \in V} \sum_{v' \in V'} \delta(l(v), l(v')), \tag{2.6}$$

where δ is the Dirac kernel and V, V' are sets of vertices of graph G, G', respectively.

Then, the Weisfeiler-Lehman sub-tree kernel (WL-ST) between graph $G = (V, E)$ and $G' = (V', E')$ with h iterations is defined as

$$K_{\text{WL-ST}}(G, G') = K_{\mathrm{VH}}(G_0, G'_0) + K_{\mathrm{VH}}(G_1, G'_1) + \cdots + K_{\mathrm{VH}}(G_h, G'_h), \tag{2.7}$$

where G_i and G'_i are graphs obtained after i^{th} iteration of the Weisfeiler-Lehman relabeling process of graphs G and G', respectively.



$$\phi_{VH}(G_0) = \; < 2, 1, 2 > \qquad \phi_{VH}(G_1) = \; < 1, 1, 0, 0, 1, 0, 2 >$$

$$\phi_{VH}(G'_0) = \; < 2, 1, 2 > \qquad \phi_{VH}(G'_1) = \; < 1, 0, 1, 1, 0, 1, 1 >$$

$$K_{\text{WL-ST}}(G, G') = K_{\mathrm{VH}}(G_0, G'_0) + K_{\mathrm{VH}}(G_1, G'_1) = 9 + 3 = 12$$

Figure 2.9: Illustration of the computaion of the Weifeiler-Lehman sub-tree kernel with $h = 1$ for graphs G and G'.

2.2.2.2 Weisfeiler-Lehman shortest-path kernel

The Weisfeiler-Lehman shortest-path kernel is the Weisfeiler-Lehman kernel where the base graph kernel is the shortest-path kernel that is defined in Eq. (2.3). Then, the Weisfeiler-Lehman shortest-path kernel between graph $G = (V, E)$ and $G' = (V', E')$ with h iterations is defined as

$$K_{\text{WL-SP}}(G, G') = K_{\text{SP}}(G_0, G'_0) + K_{\text{SP}}(G_1, G'_1) + \cdots + K_{\text{SP}}(G_h, G'_h), \tag{2.8}$$

where G_i and G'_i are graphs obtained after i^{th} iterations of the Weisfeiler-Lehman relabeling process of graphs G and G', respectively.

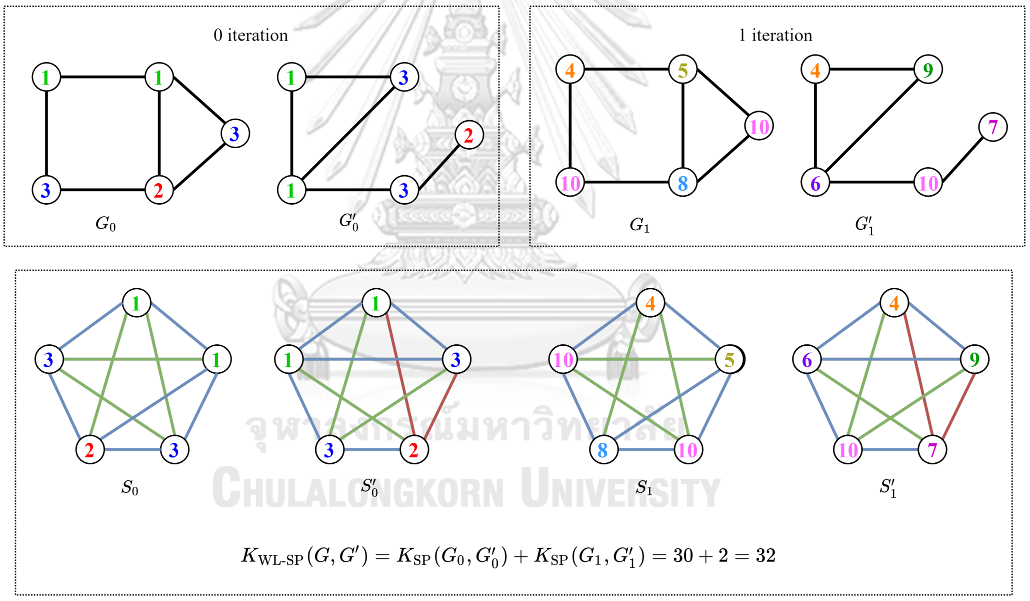

0 iteration
1 iteration
G_0
G'_0
G_1
G'_1
S_0
S'_0
S_1
S'_1
$K_{\text{WL-SP}}(G, G') = K_{\text{SP}}(G_0, G'_0) + K_{\text{SP}}(G_1, G'_1) = 30 + 2 = 32$

Figure 2.10: Illustration of the computaion of the Weisfeiler-Lehman shortest-path kernel with $h = 1$ for graphs G and G'.

2.3 Kernel method for classification

Kernel methods are a class of methods for pattern analysis. They use kernel or similarity function to implicitly transform the data into a higher-dimensional space then compute the similarity by the dot product. The most recognized kernel method is the support vector machine (SVM).

2.3.1 Support vector machine

A support vector machine (SVM) is a supervised machine learning method to solve binary classification problems. It was first invented by Vladimir Vapnik and Alexey Chervonenkis in 1963, the current version of the support vector machine was proposed by Corinna Cortes and Vladimir Vapnik in 1995[12]. The main idea of the support vector machine is to find the optimum hyperplane that can best separate data into each class. The problem to find the optimum hyperplane can be written as

$$\begin{aligned} \min \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to:} \quad & y_i(w \cdot x_i + b) \geq 1 - \xi_i \\ & \xi_i > 0 \\ & \forall i = 1, 2, \ldots, n \end{aligned} \tag{2.9}$$

where y_i is a class of i^{th} sample and $y_i \in \{-1, 1\}$,
w is a normal vector to the hyperplane,
b is a bias,
C is a penalty parameter.

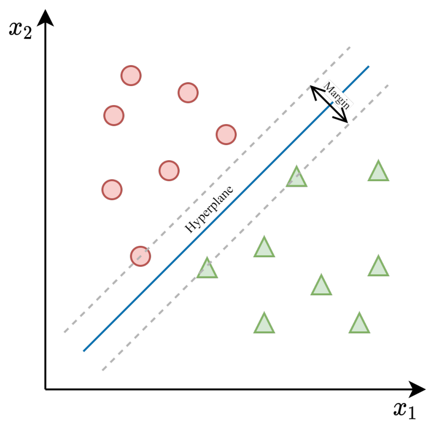


Figure 2.11: Classification of data by the support vector machine (SVM).

The optimization problem of the support vector machine can be solved with the dual problem of Lagrange multipliers which is quadratic programing.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} L_i - \frac{1}{2}\sum_{i,j} L_i L_j y_i y_j x_i \cdot x_j \\ \text{subject to:} \quad & 0 \leq L_i \leq C \\ & \sum_{i=1}^{n} L_i y_i = 0 \\ & \forall i = 1, 2, \ldots, n \end{aligned} \tag{2.10}$$

where $w = \sum_{i=1}^{n} L_i y_i x_i$,

y_i is a class of i^{th} sample and $y_i \in \{-1, 1\}$,

L_i is a Lagrange multiplier,

b is a bias,

C is a penalty parameter.

2.3.1.1 Support vector machine with kernel function

The support vector machine can be used as a nonlinear classifier by applying the kernel trick to the maximum-margin hyperplane problem [13]. Since the maximum-margin hyperplane problem depends on a dot product of the data. Therefore, we can use the kernel function to solve the problem in the high-dimensional space. Then, the optimization problem of the support vector machine with kernel function can be written as follow:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} L_i - \frac{1}{2}\sum_{i,j} L_i L_j y_i y_j K(x_i, x_j) \\ \text{subject to:} \quad & 0 \leq L_i \leq C \\ & \sum_{i=1}^{n} L_i y_i = 0 \\ & \qquad \forall i = 1, 2, \ldots, n \end{aligned} \tag{2.11}$$

where ϕ is a feature mapping function,

$w = \sum_{i=1}^{n} L_i y_i \phi(x_i)$,

$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ is a kernel function,

y_i is a class of i^{th} sample and $y_i \in \{-1, 1\}$,

L_i is a Lagrange multiplier,

b is a bias,

C is a penalty parameter.

2.3.2 Example of a kernel function

As previously mentioned, there are many kernel functions for graph-structured data such as the vertex histogram kernel, the shortest-path kernel, the Weisfeiler-Lehman sub-tree kernel, and the Weisfeiler-Lehman shortest-path kernel. The following examples are common kernel functions for data in the form of vector.

2.3.2.1 Linear kernel

The linear kernel is the simplest kernel function. It is defined by the inner product of two samples plus a constant c. Let x and x' be considered samples. Then

$$K_{\text{Linear}}(x, x') = x^T x' + c, \tag{2.12}$$

where c is the free parameter.

2.3.2.2 Polynomial kernel

The polynomial kernel is a kernel function that represents the similarity of samples in a feature space over polynomials. The Polynomial kernel for two samples x, x' is defined as

$$K_{\text{Polynomial}}(x, x') = (\alpha x^T x' + c)^d, \tag{2.13}$$

where α and c are free parameters and d is the polynomial degree.

2.3.2.3 Radial basis function kernel

Radial basis function kernel (RBF kernel) is a common kernel function for support vector machine classification. The RBF kernel for two samples x, x' is defined as

$$K_{\mathrm{RBF}}(x, x') = \exp(-\gamma ||x - x'||^2), \tag{2.14}$$

where γ is the free parameter.

2.4 Sub-graph isomorphism problem

The problem of matching graphs or sub-graphs is called sub-graph isomporphism problem. In cheminformatics, this problem is implemented to find the substructure of biochemical and chemical compound. H. Ehrlich and M. Rarey recommended using the VF2 algorithm for molecular substructure searching[14].

2.4.1 VF2 algorithm

The VF2 algorithm[15] is a well-known state-of-the-art algorithm for solving sub-graph isomorphism problem. It is based on a state space representation, each state is a partial mapping between the two considered graphs. The final state is a complete mapping which is a graph isomorphism. The VF2 algorithm is as followed:

Algorithm 1: VF2

```
Function Match($G_1, G_2, s$)
    input  : a target graph $G_1$, a pattern graph $G_2$, an intermediate
             state $s$; the initial state $s_0$ has $M(s_0) = \emptyset$
    output: the mapping between two graph
    if $M(s)$ covers all the vertices of $G_2$ then
        return $M(s)$
    else
        $P(s) \leftarrow$ the set of the pairs candidate for inclusion in $M(s)$
        for $p = (u, v) \in P(s)$ do
            if $F(s, u, v)$ is true then
                Compute the state $s'$ obtained by adding $p$ to $M(s)$
                Match($G_1, G_2, s'$)
```

Let $G_1 = (V_1, E_1)$ be a target graph and $G_2 = (V_2, E_2)$ be a pattern graph. In each state s of the algorithm, mapped couples are stored in a set $M(s)$ called the core set. $M_1(s)$ and $M_2(s)$ are sets of vertices in $M(s)$ for first and second graph, respectively. Neighbors of vertices in the core set are kept in a two-set, $T_1(s)$ and $T_2(s)$, namely terminal sets. VF2 algorithm selects the candidate pair (u, v) by picking u from $T_1(s)$ and v from $T_2(s)$ if $T_1(s) \neq \emptyset$ and $T_2(s) \neq \emptyset$. Otherwise, the candidate pair will be picked from the remaining vertices of G_1 and G_2 that are not in the core or in the terminal sets. The example of core sets and terminal sets are shown in Figure 2.12.

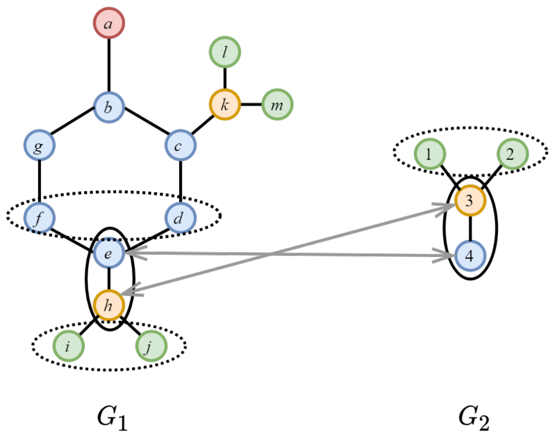


Figure 2.12: Core sets (solid line) and terminal sets (dotted line) used by VF2 of a target graph G_1 and a pattern graph G_2. In this example, $M(s) = \{(e, 4), (h, 3)\}, M_1(s) = \{e, h\}, M_2(s) = \{3, 4\}, T_1(s) = \{f, d, i, j\}, T_2(s) = \{1, 2\}$.

The candidate pair (u, v) will be added in $M(s)$ if it passes the feasibility rules of VF2. Let $F(s, u, v)$ be the feasibility function, which is true if the addition to a state s of the candidate pair (u, v) satisfies all the feasibility rules. The feasibility function is define as:

$$F(s, u, v) = F_{\text{syn}}(s, u, v) \wedge F_{\text{sem}}(s, u, v), \tag{2.15}$$

where F_{syn} is a syntactic feasibility rule, and F_{sem} is a semantic feasibility rule.

The syntactic feasibility rule depends on the structure of graphs. It consists of two sets of rules, the core rule and the look-ahead rules. The core rule is used to check that the candidate pair can be a part of a solution. The look-ahead rules are used to confirm that the algorithm can continue searching for a final solution after including the candidate pair into the partial mapping $M(s)$. The syntactic feasibility rule for the sub-graph isomorphism problem is defined as:

$$F_{\text{syn}}(s,u,v) = R_{\text{core}}(s,u,v) \wedge R_{\text{term}}(s,u,v) \wedge R_{\text{new}}(s,u,v), \tag{2.16}$$

where R_{core} is the core rule, R_{term} is the 1-look-ahead rule, and R_{new} is the 2-look-ahead rule.

The core rule, the 1-look-ahead-rule, and the 2-look-ahead rule for undirected graphs are defined as[16]:

$$\begin{aligned} R_{\text{core}}(s,u,v) &\iff \\ \forall u' \in adj(u) \cap M_1(s), \exists v' \in adj(v) \cap M_2(s) &: (u',v') \in M(s) \\ \wedge \forall v' \in adj(v) \cap M_2(s), \exists u' \in adj(u) \cap M_1(s) &: (u',v') \in M(s), \end{aligned} \tag{2.17}$$

$$R_{\text{term}}(s,u,v) \iff |adj(u) \cap T_1(s)| \geq |adj(v) \cap T_2(s)|, \tag{2.18}$$

$$R_{\text{new}}(s,u,v) \iff |adj(u) \cap \tilde{V}_1(s)| \geq |adj(v) \cap \tilde{V}_2(s)|, \tag{2.19}$$

where $adj(u)$ is the neighbors of the vertex u, $adj(v)$ is the neighbors of the vertex v, $\tilde{V}_1(s) = V_1 - M_1(s) - T_1(s)$, and $\tilde{V}_2(s) = V_2 - M_2(s) - T_2(s)$.

The semantic feasibility rule depends on the attributes of each vertex and edge. The semantic feasibility on undirected labeled graphs is defined as:

$$\begin{aligned} F_{\text{sem}}(s,u,v) &\iff \\ l_1(u) &= l_2(v) \\ \wedge \forall (u',v') \in M(s), (u,u') \in E_1 \rightarrow l_1((u,u')) &= l_2(v,v'), \end{aligned} \tag{2.20}$$

where l_1 is a labeling function of G_1, and l_2 is a labeling function of G_2.

CHAPTER III

PROPOSED METHOD

This chapter proposes the definition of a primitive structure, the relevant algorithm, and the outline of the graph classification method based on primitive structures. The algorithm to select the primitive structure from a set of candidate structures is called "primitive structure selection" and the algorithm to extract these structures from a graph in a data set and recreate the new graph based on the connection of these structures is called "primitive structure extraction".

3.1 Outline of our graph classification method

A graph classification method has two main steps. First, the information extraction method is applied to extract the information such as features of graphs or the similarity between each pair of graphs. After that, the information from the first step is used to classify the graph-structured data.

In this work, we proposed a method to extract the information from biochemical compounds represented as graphs. Our method consisted of three steps. The first step was to identify common sub-graph structures from all graphs of biochemical compounds in the data set. These common sub-graphs were called "primitive structures". In the second step, a considered biochemical compound graph was scanned to identify all primitive structures which were obtained from the first step. Then, the connection of all primitive structures in a considered graph was formed as a new graph. These graph was called "primitive structure graph". In the last step, the similarity between each pair of primitive structure graphs was computed. These similarity measures of all biochemical compounds were fed into kernel methods such as support vector machine to classify them.

Since the main idea of the first and the second step of our method is to identify a primitive structure in a considered graph, this problem was called “sub-graph isomorphism problem” which was one of the NP-class problems. The VF2 algorithm by Cordella et al. [15] was applied to solve this problem. The time complexity of the VF2 algorithm is $O(|V_{\max}|^2)$ in the best case and $O(|V_{\max}||V_{\max}|!)$ in the worst case where $|V_{\max}|$ is the maximum number of vertices of considered graphs.

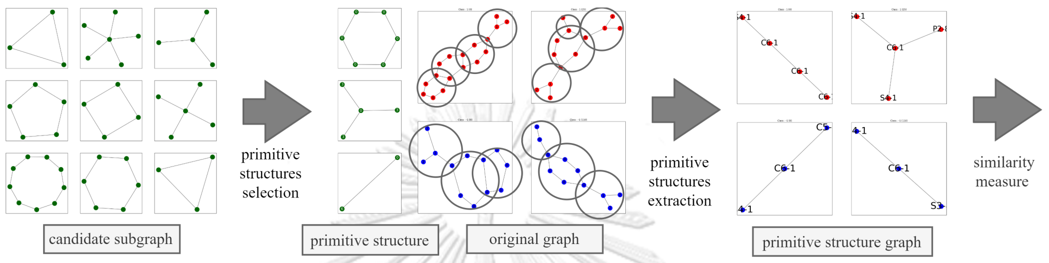

candidate subgraph
primitive structures selection
primitive structure
original graph
primitive structures extraction
primitive structure graph
similarity measure

Figure 3.1: The outline of our method

The diagrams of the illustrated outline for our method is shown in Figure 3.1. The leftmost diagram shows the set of candidate sub-graphs. The second diagram shows some examples of the process of identifying all primitive structures from compound graphs. The third diagram shows the representation of each primitive structure as a vertex and the connection of these representing vertices.

3.2 Primitive structure

The important task of this method was finding the common molecules of biochemical compounds from all compound graphs in the data set. We called these common molecules as “primitive structures”.

Definition 3.2.1 (Primitive structure)**.** A primitive structure is a labeled graph that has similar structure with a chemical molecule that appears as a sub-graph of a compound graph in the data set.

Many molecular structures can be written as a combination of cycle graphs, star graphs, and path graphs as shown in [17], [18]. For example, the molecular structure of propane can be shown as the star graph with 3 vertices (S_2), the molecular structure of isobutane can be shown as the star graph with 4 vertices (S_3), and the molecular structure of benzene can be shown as the cycle graph with 6 vertices (C_6). As in the Figure 3.2 and Figure 3.3, a chemical compound of 2,4-Dinitrochlorobenzene ($(O_2N)_2C_6H_3Cl$) consists of a NO_2 bounds with a C, a benzene, and a Cl bounds with a C where these components are similar to a star graph with 3 vertices (S_2), a cycle graph with 6 vertices (C_6) and a star graph with 2 vertices (S_1) or a path graph with 2 vertices (P_2), respectively.

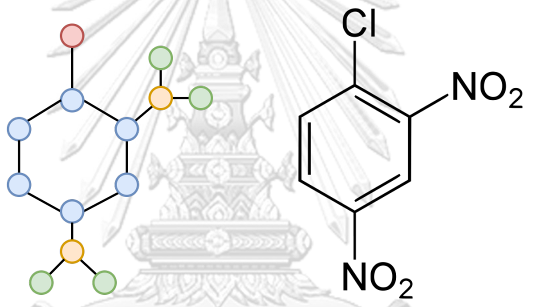


Figure 3.2: 2,4-Dinitrochlorobenzene ($(O_2N)_2C_6H_3Cl$) and representation in the form of graph.

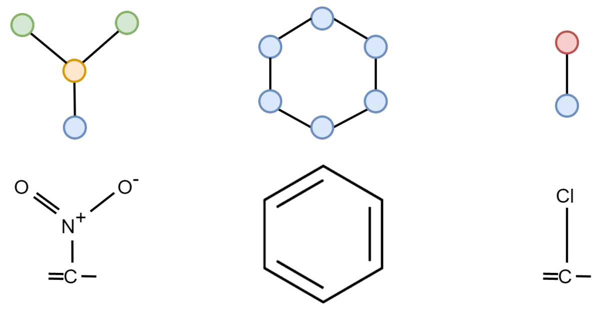


Figure 3.3: Composition of 2,4-Dinitrochlorobenzene. From left to right, a NO_2 bounds with a C in a benzene ring, a benzene ring (C_6H_6), a Cl bounds with a C in a benzene ring.

3.3 Proposed method

The following section is an explanation of each step of our proposed method.

3.3.1 Primitive structure selection

The first step of our method is to identify and collects all common sub-graph structures from all compound graphs in the data set. The algorithm of this step is as follows. First, let C be a set of candidate sub-graphs that consists of cycle graphs, star graphs and path graphs, and G be a data set of considered compound graphs. The candidate graph that is isomorphic with a sub-graph of a graph in the data set is selected as a primitive structure. The time complexity of this algorithm is $O(|C||G||V_{\max}||V_{\max}|!)$ where $|V_{\max}|$ is the maximum number of vertices of considered graphs. Code of this algorithm in python is shown in APPENDIX A.

Algorithm 2: Primitive structure selection

```
input  : set of candidate sub-graphs $C$ where $c \in C$ is a unlabeled graph,
         set of graphs $G$ where $g \in G$ is a labeled compound graph
output: set of primitive structures $P$
$A \leftarrow \emptyset$
for $c \in C$ do
    for $g \in G$ do
        $S \leftarrow$ set of a sub-graph in $g$ that isomorphic with $c$
        for $s$ in $S$ do
            add $s$ to $A$
$P = \emptyset$
for $p \in A$ do
    if $p$ is not isomorphic with a graph in $P$ then
        add $p$ in $P$
```

3.3.2 Primitive structure extraction

In this step, a considered compound graph structure is scanned to identify all primitive structures which are defined from the step of “Primitive structure selection”, then the connection of all primitive structures is formed as a new graph. The algorithm of this step is as follow. Let P be a set of primitive structures, and G be a set of considered compound graphs. Let $p_i \in P$ be the i^{th} primitive structure. For every graph in a data set G, a new graph is created by considering each primitive structure of an original graph as vertex with label i. Each primitive structure sub-graph in the set of all primitive structures is searched and identified inside the considered compound graph, starting from the largest to the smallest cycle structures, the largest to the smallest star structures, and the largest to the smallest path structures.

After all primitive structures are found, a primitive structure graph is created. Each primitive structure is represented by a vertex in the new graph if it is not share more than half of its vertices with other registered primitive structures. The procedure to check this condition is the same as the previos order, starting from the largest to the smallest cycle structures, the largest to the smallest star structures, and the largest to the smallest path structures. There is an edge which connect between each pair of primitive structure vertices if they share the same vertex in the original graph. The time complexity of this algorithm is $O(|P||G||V_{\max}||V_{\max}|!)$ where $|V_{\max}|$ is the maximum number of vertices of considered graphs. Code of this algorithm in python is shown in APPENDIX B.

Algorithm 3: Primitive structure extraction

```
input  : set of primitive structures $P$, set of graphs $G$
output: set of primitive structure graphs $H$
$P \leftarrow$ sorted $P$ from type of primitive structure
$H \leftarrow \emptyset$
for $g \in G$ do
    $S_{\text{all}} \leftarrow \emptyset$; $S_{\text{pri}} \leftarrow \emptyset$
    // $S_{\texttt{all}}$ is a set of all founded primitive structures in $g$
    // $S_{\texttt{pri}}$ is a set of primitive structures in $g$ after delete
       some overlap structures
    for $p_i \in P$ do
        $V_p \leftarrow$ set of nodes of sub-graph of $g$ that isomorphic with graph $p_i$
        for $v \in V_p$ do
            add $v$ to $S_{all}$
    for $s_{all} \in S_{all}$ do
        not_overlay $\leftarrow$ True
        for $s_{pri} \in S_{pri}$ do
            $s_{\text{share}} \leftarrow s_{\text{all}} \cap s_{\text{pri}}$
            if $|s_{share}| > \frac{1}{2}|s_{all}|$ then
                not_overlay $\leftarrow$ False
        if not_overlay then
            add $s_{\text{all}}$ to $S_{\text{pri}}$
    $V \leftarrow \emptyset$; $E \leftarrow \emptyset$; $i \leftarrow 1$
    for $s_{pri} \in S_{pri}$ do
        add node $i$ to $V$
        $l(i) \leftarrow$ type of primitive structure of $s_{\text{i}}$
        $j \leftarrow 1$
        for $s_j \in S_{pri}$ do
            if $s_i \cap s_j \neq \emptyset$ and $i \neq j$ then
                add edge $\{i, j\}$ to $E$
            $j \leftarrow j + 1$
        $i \leftarrow i + 1$
    add $h = (V, E)$ to $H$
```

3.3.3 Computing similarity measure

The last step is to extract the information from primitive structure graphs which is obtained in the second step. The following are examples of techniques for extracting the information from the graph-structured data.

3.3.3.1 Graph similarity

As previously mentioned, the similarity between graphs can be computed by using the graph kernel such as the vertex histogram kernel (VH) in equation 2.6, the shortest-path kernel (SP) in equation 2.3, the Weisfeiler-Lehman sub-tree kernel (WL-ST) in equation 2.7, and the Weisfeiler-Lehman shortest-path kernel (WL-SP) in equation 2.8.

3.3.3.2 Counting the number of edges between each type of vertex label

The other way to extract the information from the primitive structure graphs is considering the number of connections between each pair of vertices in graphs, by counting the number of edges between each type of the vertex label.

Let $G = (V, E)$ be a considered graph, l be a labeling function of all graphs in a data set. We will label each edge $e = (u, v) \in E$ by $l(e) = (l(u), l(v))$. Assume that, there are d edge labels in total among all graphs in a data set, say $s_1, s_2, \ldots, s_d$. Then, the vector contains the number of edges between each type of the vertex label of G is

$$f(G) = (f_1, f_2, \ldots, f_d) \tag{3.1}$$

where $f_i = |\{e \in E : l(e) = s_i\}|$ for each $i = 1, 2, \ldots, d$.

CHAPTER IV

EXPERIMENTS AND RESULTS

In this chapter, we evaluated our method by comparing it with graph classification methods based on a graph kernel on the original graph.

4.1 Data sets

The following standard benchmark data sets of biochemical compounds from TUD Datasets were used [19]. Each vertex represented an atom and each vertex label represented a type of an atom. Each edge represented a chemical bond between a pair of atoms. Code to import the benchmark data set in python is shown in APPENDIX C. In this work, we considered only connected graphs to evaluate in this experiment. The statistic result such as the number of graphs, the class ratio, the average number of vertices, the average number of edges, and the number of vertex labels of each data set is shown in Table 4.1 where the class ratio is the ratio of the number of positive data among the number of negative data. The selected data sets in this study were as follows:

Table 4.1: Statistics of data sets in the experiment

Datasets	#Graphs	Class ratio	Avg. #vertices	Avg. #edges	#vertex labels
MUTAG	188	1:2	17.93	19.79	7
NCI1	3865	1:1.1	29.39	32.02	19
BZR	405	1:3.7	35.75	38.36	10
COX2	467	1:3.6	41.22	43.45	8

4.1.1 MUTAG

MUTAG is a data set containing 188 nitro compounds where classes indicate the mutagenic effect on a bacterium named *Salmonella typhimurium*[20].

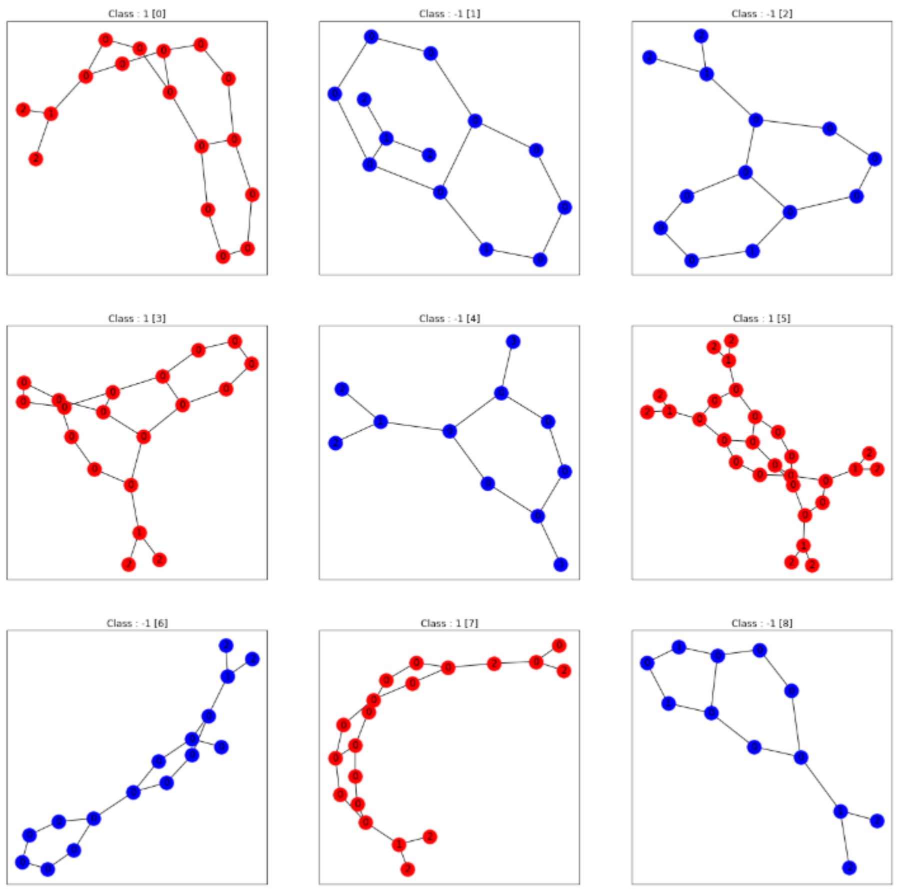


Figure 4.1: The examples of compound graphs from MUTAG data set. Graphs with red vertices are positive while graphs with blue vertices are negative.

4.1.2 NCI1

NCI1 is a data set containing 3,865 chemical compounds where classes indicate the activity against non-small cell lung cancer[21].

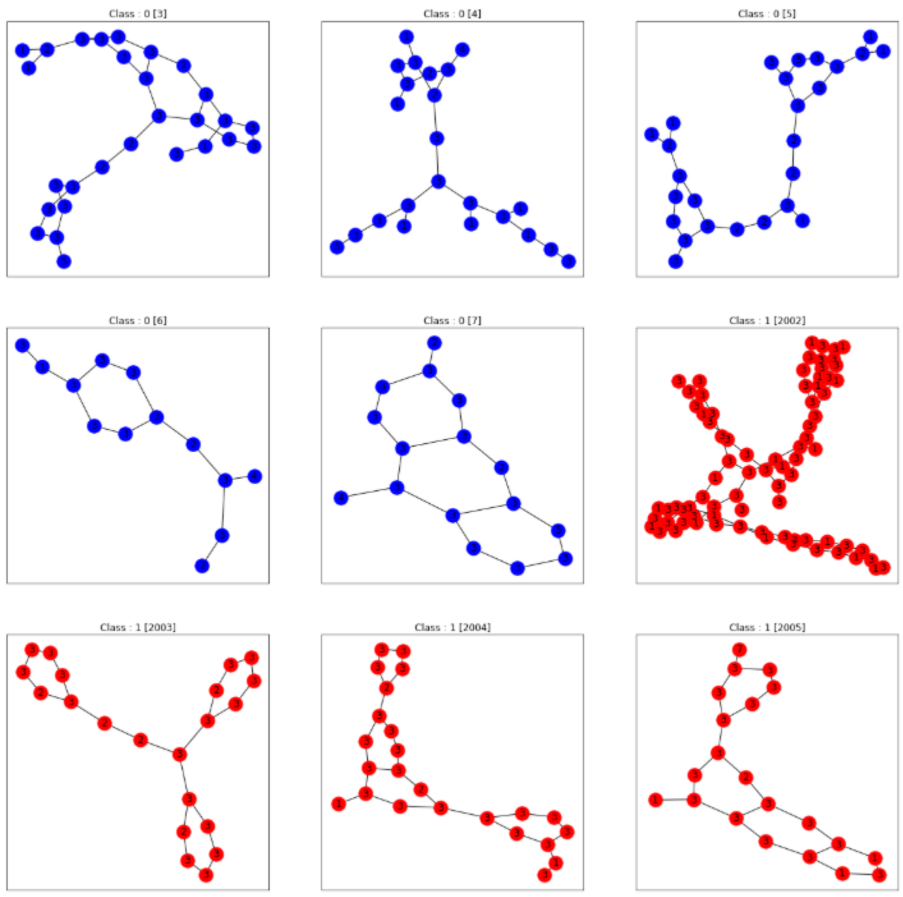


Figure 4.2: The examples of compound graphs from NCI1 data set. Graphs with red vertices are positive while graphs with blue vertices are negative.

4.1.3 BZR

BZR is a data set containing 405 ligands where classes indicate the benzodiazepine receptor[22].

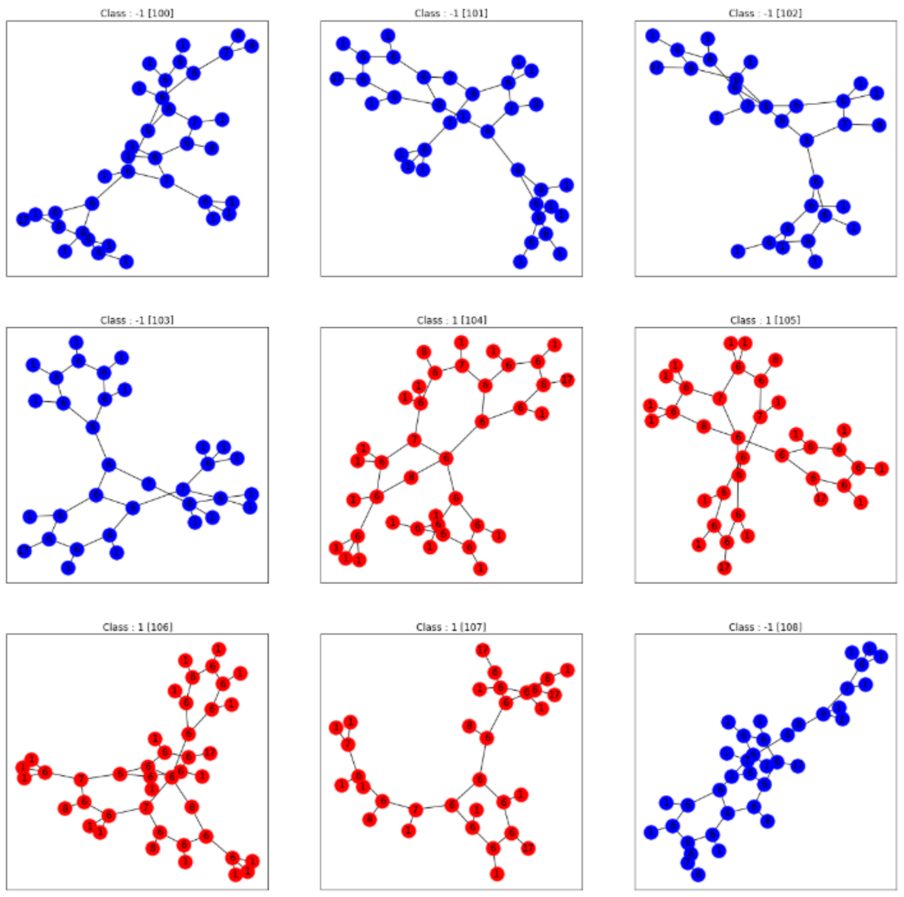


Figure 4.3: The examples of compound graphs from BZR data set. Graphs with red vertices are positive while graphs with blue vertices are negative.

4.1.4 COX2

COX2 is a data set containing 467 cyclooxygenase-2 inhibitors where classes indicate vitro activities against human recombinant enzymes [22].

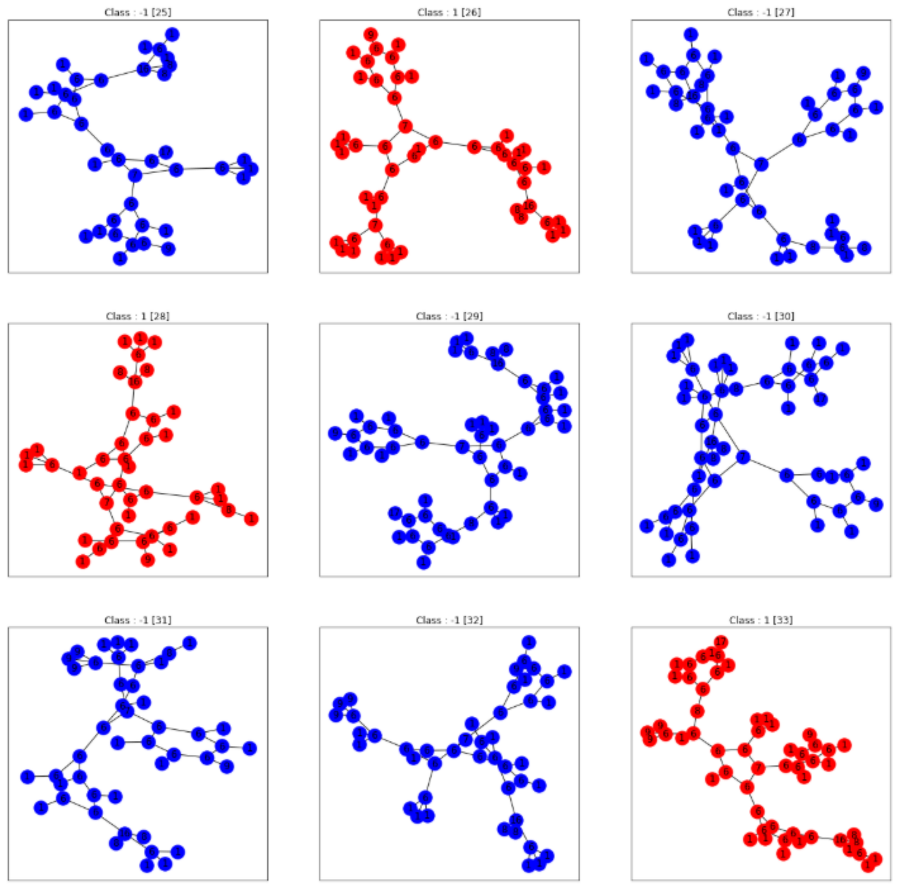


Figure 4.4: The examples of compound graphs from COX2 data set. Graphs with red vertices are positive while graphs with blue vertices are negative.

4.2 Experimental setup

A set consisted of labeled cycle graphs with 3 to 10 vertices, labeled star graphs with 3 to 7 vertices, and a path graph with 2 vertices was selected as a set of candidate sub-graphs. The Weisfieler-Lehman subtree kernel (Primitive WL-ST), the Weisfieler-Lehman shortest-path kernel (Primitive WL-SP), the shortest-path kernel (Primitive SP), and the number of edges between each type of vertex label which is a label of the primitive structure (Primitive Edges) were selected as similarity measures. The obtained results from our proposed method based on different similarity measures were reported. Our methods were compared to the graph classification method on original graphs with following similarity measures, the Weisfieler-Lehman subtree kernel (WL-ST), the Weisfieler-Lehman shortest-path kernel (WL-SP), the shortest-path kernel (SP), and the number of edges between each type of vertex label (Edges).

The shortest-path kernel, the Weisfeiler-Lehman subtree kernel, and the Weisfeiler-Lehman shortest-path kernel were computed by a library of graph kernels for python called GraKel [23]. The sub-graph isomorphism and graph isomorphism problem were computed by graph library for python called graph-tools [24] and networkx [25].

80% of data were for the training set and 20% were for the test set. Five-fold stratified cross-validation of the support vector machine using scikit-learn [26] were used on the training set to exclude the random effect of data separation and to select the best hyperparameters. The experiments were repeated 10 times on each data set. The height parameter for the Weisfieler-Lehman sub-tree kernel was chosen in $\{1, 2, ..., 5\}$, and for the Weisfieler-Lehman shortest-path kernel was chosen in $\{1, 2, 3\}$ except the data set NCI1 whose the height parameter was set to be $h = 1$ due to the memory limitation of the machine. The free parameter

C for the support vector machine was chosen between 10^{-2} and 10^3. Optimal hyper-parameters for each method in this experiment are shown in Table 4.2.

The information obtained by the number of edges between each type of vertex label (edge) was fed into the support vector machine with the radial basis function kernel setting $\gamma = 1/n$, where n was the number of features. Other graph kernels were also used as the kernel function of the support vector machine.

The CPU runtimes in seconds measured on 1.80GHz Intel(R) Core(TM) i7-8565U CPU and 20GB RAM are reported in Table 4.8 and Table 4.9.

Code for classification with SVM by using the Weisfeiler-Lehman sub-tree kernel as a similarity measure in python is shown in APPENDIX D.

Table 4.2: Parameters on each data set where θ is the threshold value of primitive structures selection, h is the height parameter of the Weisfeiler-Lehman framework, and C is a parameter for the support vector machine.

Methods	MUTAG		NCI1		BZR		COX2	
	h	C	h	C	h	C	h	C
WL-ST	1	1000	5	21.54	3	21.54	1	1000
WL-SP	1	278.26	1	21.54	3	77.43	2	77.43
SP	-	1000	-	278.26	-	21.54	-	1000
Edges	-	1.67	-	21.54	-	1.67	-	1.67
Primitive WL-ST	2	1.67	4	5.99	1	21.54	5	5.99
Primitive WL-SP	2	0.46	1	5.99	1	21.54	1	77.43
Primitive SP	-	0.46	-	5.99	-	21.54	-	77.43
Primitive Edges	-	1.67	-	278.26	-	278.26	-	77.43

4.3 Performance measures

The results of the classification can be separated into 4 cases. The first case are called true positives, which are positive data such that a classification predicts them as positive. The second case are called false positives, which are negative data such that a classification predicts it them as positive. The third case are called true negatives, which are negative data such that a classification predicts them as negative. The last case are called false negatives, which are positive data such that a classification predicts them as negative.

Let TP be the number of true positives, FP be the number of false positives, TN be the number of true negatives and FN be the number of false negatives. We use the following metrics to evaluate the performance of our method.

4.3.1 Accuracy

The accuracy is the rate of correct classifications. It is defined as a proportion of true results among the total number of cases examined.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

4.3.2 Precision

The precision is a measurement of the confidence of a positive prediction. It is defined as a proportion of true positives among all positive predictions.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2}$$

4.3.3 Recall

The recall is a measurement of the completeness of a positive prediction. It is defined as a proportion of true positives among all positive cases.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.3}$$

4.3.4 F1 score

The F1 score is a performance measure of a classification method. It is defined by the harmonic mean of the precision and the recall.

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.4}$$

4.3.5 Wilcoxon signed-ranks test

The Wilcoxon signed-ranks test is a non-parametric statistical hypothesis test proposed by Wilcoxon in 1945[27]. This test is recommended for comparison of two classifiers[28]. It ranks the differences in performances of two classifiers for each data set and compare the rank for the positive and negative differences.

Let c_i^1 and c_i^2 be performance measures of two classifiers on the i^{th} of N observation and let $d_i = c_i^2 - c_i^1$ be the differences and M is the median differences. The one-sided test is used for this experiment where the null hypothesis is: the median of difference is less than or equal to zero ($H_0 : M \leq 0$), and the alternative hypothesis is: the median difference is greater than zero ($H_A : M > 0$). The differences are ranked by their absolute values. Let R^+ be the sum of ranks for the data sets that the second algorithm outperforms the first, and R^- be the sum of ranks for the data sets that the first algorithm outperforms the second. Ranks of $d_i = 0$ are split among the sums:

$$R^+ = \sum_{d_i>0} \mathrm{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \mathrm{rank}(d_i), \tag{4.5}$$

$$R^- = \sum_{d_i<0} \mathrm{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \mathrm{rank}(d_i). \tag{4.6}$$

For this null hypothesis of the Wilcoxon signed-rank test, we consider on the sum of ranks of positive differences, let $T = R^+$. The exact critical values for T is used if N is less than or equal to 25. Otherwise, the test statistics

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \tag{4.7}$$

is distributed approximately normally. The null hypothesis can be rejected if z is greater than a critical value.

4.4 Results

4.4.1 Classification performance

The average classification performance on each data set of the experiments are shown in Table 4.3.

The graph classification method on original graphs with the Weisfeiler-Lehman subtree kernel (WL-ST) had a good performance on the NCI1 data set with the highest recall and F1-score which were 0.819 and 0.826, respectively. This method had the best accuracy among other methods that extract the information from the

Table 4.3: Average classification performance on each data set

Methods	Data sets															
	MUTAG				NCI1				BZR				COX2			
	Acc.	Pre.	Rec.	F1	Acc.	Pre.	Rec.	F1	Acc.	Pre.	Rec.	F1	Acc.	Pre.	Rec.	F1
WL-ST	.842	.866	.909	.884	.833	.835	**.819**	**.826**	.884	.868	.599	.703	.815	.641	**.404**	.489
WL-SP	.826	.850	.907	.873	.813	.808	.806	.807	.879	.826	.632	.709	.809	.653	.402	.480
SP	.797	.823	.890	.852	.734	.729	.722	.725	.860	.792	.553	.649	.812	.674	.369	.456
Edges	.847	.872	**.915**	.888	.717	.735	.653	.691	.843	**.941**	.350	.504	.819	**.781**	.287	.413
Primitive WL-ST	.884	.932	.896	.911	.822	.830	.798	.814	.859	.765	.569	.648	.815	.651	.373	.463
Primitive WL-SP	**.905**	**.951**	.906	**.926**	**.834**	**.854**	.795	.823	.880	.782	**.682**	.727	.801	.595	.367	.446
Primitive SP	.889	.920	.914	.916	.804	.806	.786	.795	.874	.789	.631	.699	.795	.545	.395	.453
Primitive Edges	.871	.937	.866	.899	.794	.804	.763	.783	**.896**	.847	.680	**.752**	**.826**	.700	.400	**.498**

original graph on the NCI1 and BZR data sets.

The method based on the numbers of edges between each type of vertex label on original graphs (Edges) had the highest recall on MUTAG which was 0.915, the highest precision on BZR which was 0.941, and the highest precision on COX2 which was 0.781. This method outperformed other methods based on the original graph on the MUTAG data set. For the COX2 data set, this method had the second best accuracy which was 0.819.

The classification method on primitive structure graphs with the Weisfeiler-Lehman subtree kernel (Primitive WL-ST) had a better performance compared to other methods on original graphs on the MUTAG data set. For the other data sets, this method had moderate performance.

Our method with the Weisfeiler-Lehman shortest-path kernel (Primitive WL-SP) had the best accuracy, precision, and F1-score on the MUTAG data set. The accuracy of this method on the MUTAG data set was 0.905, the precision was 0.951, and the F1-score was 0.926. It also had good performance on the NCI1 which its accuracy was 0.834, its precision was 0.854, and its F1-score was 0.824 which was the runner-up method in term of the F1-score on this data set. This method had the highest recall on the BZR data set which was 0.682 and the runner-up F1-score which was 0.727.

Our method with the shortest-path kernel (Primitive SP), this method had the runner-up accuracy and F1-score on the MUTAG data set. The accuracy of this method on the MUTAG data set was 0.889, and the F1-score of this method on the MUTAG data set was 0.916. However, this method had the worst accuracy on the COX2 data set.

The last method was the method based on the numbers of edges between each type of vertex labels on primitive structure graphs (Primitive Edges). This method had the greatest accuracy and the F1-score on the BZR and the COX2 data sets. The accuracy and the F1-score on the BZR were 0.896 and 0.752, respectively. For the COX2 data set, the accuracy was 0.826 and the F1-score was 0.498.

Our method had greater performance than the method based on the original graph on the MUTAG data set. Moreover, using the Weisfeiler-Lehman shortest-path kernel, the shortest-path kernel, and the numbers of edges between each type of vertex labels improved the performance of the classification on the NCI1 and the BZR data set. However, on the COX2 data set, our method had better performance than the method on the original graph only when used the numbers of edges between each type of vertex labels as the similarity measure. Since compounds in the COX2 data set were bigger than compounds in other data set. Their average number of vertices was 41.22 and the average number of edges was 43.45. In the opinion of the author, path graphs with size more than 2 should include the set of candidate sub-graphs to handle this data set.

4.4.2 Comparing between the classification methods based on original graphs and primitive structure graphs

In this section, we compared between the classification methods based on original graphs and the classification methods based on primitive structure graphs with the one-sided Wilcoxon signed-ranks test. Let c_i^1 be a performance measure of classification methods based on original graphs of i^{th} observation, and c_i^2 be a performance measure of classification methods based on primitive structure graphs of i^{th} observation. In this experiment, we had totally 40 observations for each pair of classifiction methods. The accuracy and the F1-score were used as the performance measures to comparing classification methods.

4.4.2.1 WL-ST and Primitive WL-ST

The average accuracy and the average F1-score between the method using the Weisfeiler-Lehman subtree kernel on original graphs and the method using the Weisfeiler-Lehman subtree kernel on primitive structure graphs are shown in Figure 4.5.

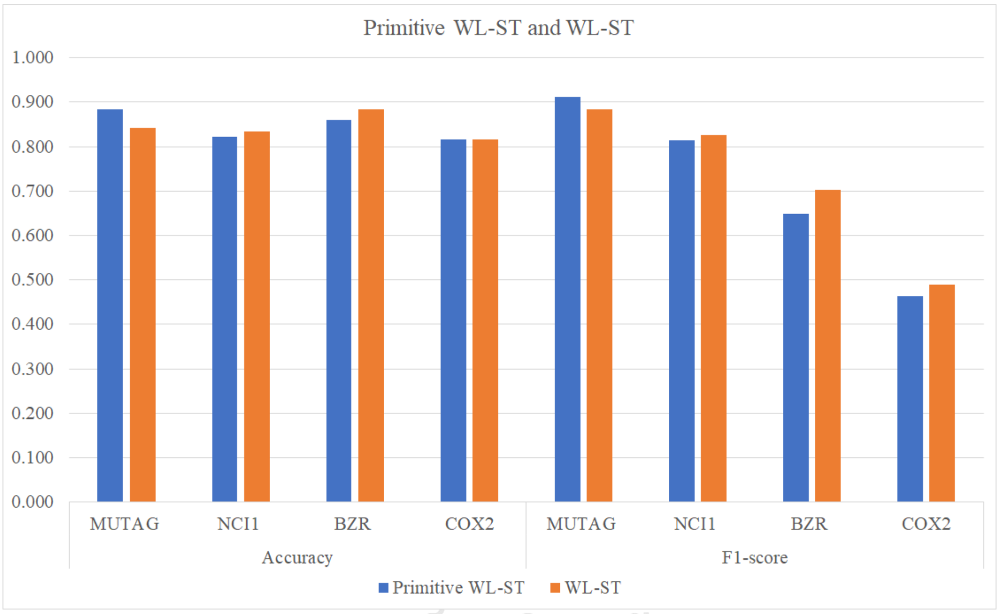


Figure 4.5: The graph shows comparison between the average accuracy and the average F1-score on each data set of Primitive WL-ST and WL-ST

The Wilcoxon signed-ranks test between these methods is shown in Table 4.4.

Table 4.4: Test statistics to comparing between WL-ST and Primitive WL-ST

Accuracy		F1-score	
R^+	301	R^+	261
R^-	519	R^-	559
z	-1.465	z	-2.003
p-value	0.92855	p-value	0.97740

The null hypothesis could not be rejected on both accuracy and F1-score. Thus, our method could not improve the classification performance when combining with the Weisfeiler-Lehman subtree kernel. The method using the Weisfeiler-Lehman subtree kernel on primitive structure graphs had better performance only on the MUTAG data set where the compounds in this data set were small.

4.4.2.2 WL-SP and Primitive WL-SP

The average accuracy and the average F1-score between the method using the Weisfeiler-Lehman shortest-path kernel on original graphs and the method using the Weisfeiler-Lehman shortest-path kernel on primitive structure graphs are shown in Figure 4.6.

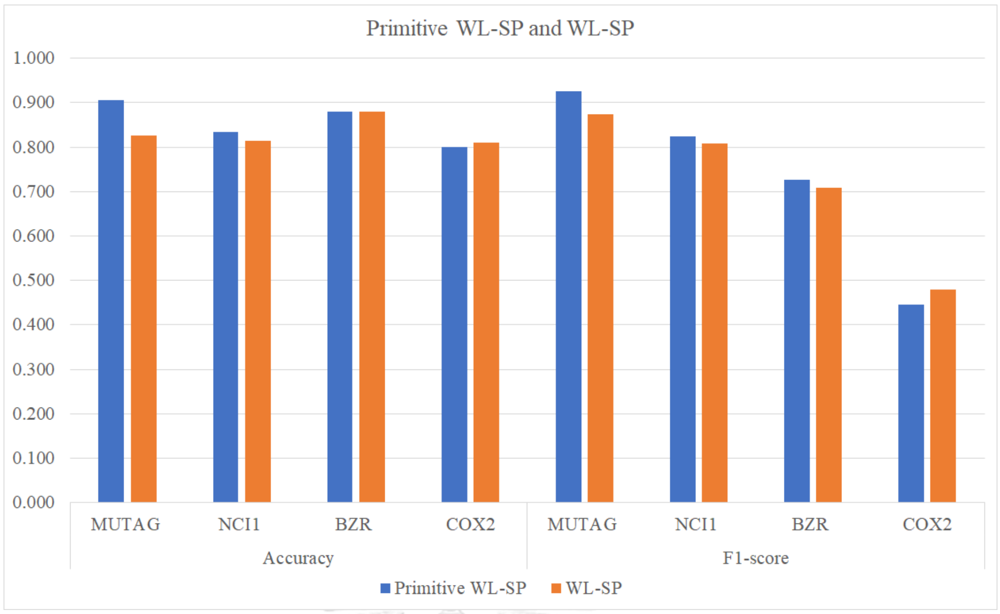


Figure 4.6: The graph shows comparison between the average accuracy and the average F1-score on each data set of Primitive WL-SP and WL-SP

The Wilcoxon signed-ranks test between these methods is shown in Table 4.5.

Table 4.5: Test statistics to comparing between WL-SP and Primitive WL-SP.

Accuracy		F1-score	
R^+	639.5	R^+	518
R^-	180.5	R^-	302
z	3.085	z	1.452
p-value	0.00102	p-value	0.07330

The null hypothesis of accuracy could be rejected at a significant level of 0.005. Thus, our method could improve the classification accuracy when combining with the Weisfeiler-lehman shortest-path kernel. The p-value of test statistics for F1-score was between 0.05 and 0.10. This provide evidence that our method's F1-score was slightly greater than the method solely used the Weisfeiler-Lehman shortest-path kernel.

4.4.2.3 SP and Primitive SP

The average accuracy and the average F1-score between the method using the shortest-path kernel on original graphs and the method using the shortest-path kernel on primitive structure graphs are shown in Figure 4.7.

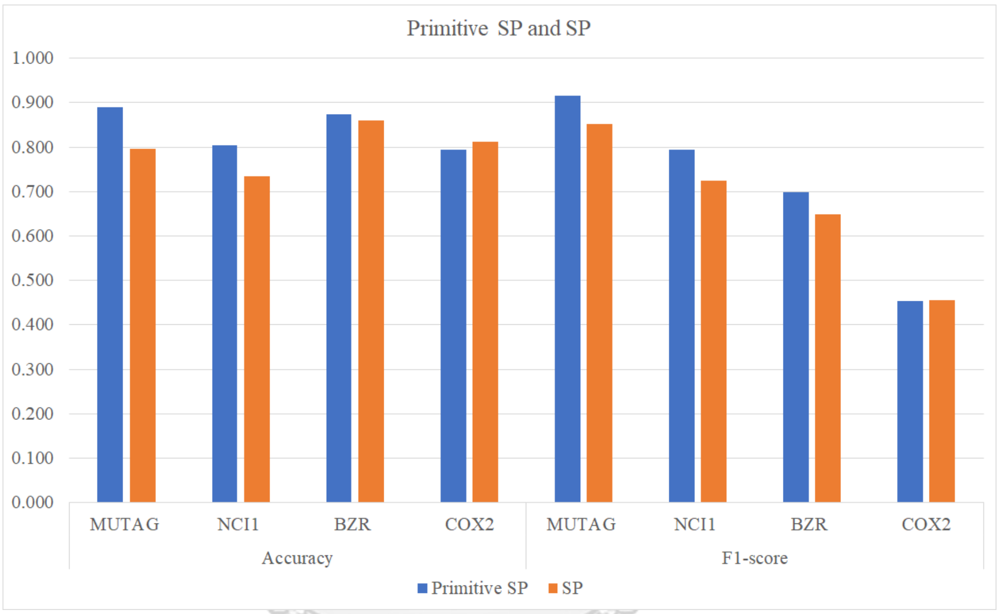


Figure 4.7: The graph shows comparison between the average accuracy and the average F1-score on each data set of Primitive SP and SP

The Wilcoxon signed-ranks test between these methods is shown in Table 4.6.

Table 4.6: Test statistics to comparing between SP and Primitive SP.

Accuracy		F1-score	
R^+	765.5	R^+	679
R^-	54.5	R^-	141
z	4.778	z	6.616
p-value	$8.836*10^{-7}$	p-value	$1.498*10^{-4}$

The null hypothesis could be rejected on both accuracy and F1-score at a significant level of 0.001. Thus, our method with the shortest-path kernel had greater performance than the method solely used the shortest-path kernel.

4.4.2.4 Edge and Primitive Edge

The average accuracy and the average F1-score between the method based on the numbers of edges between each type of vertex labels on original graphs and the method based on the numbers of edges between each type of vertex labels on primitive structure graphs are shown in Figure 4.8.

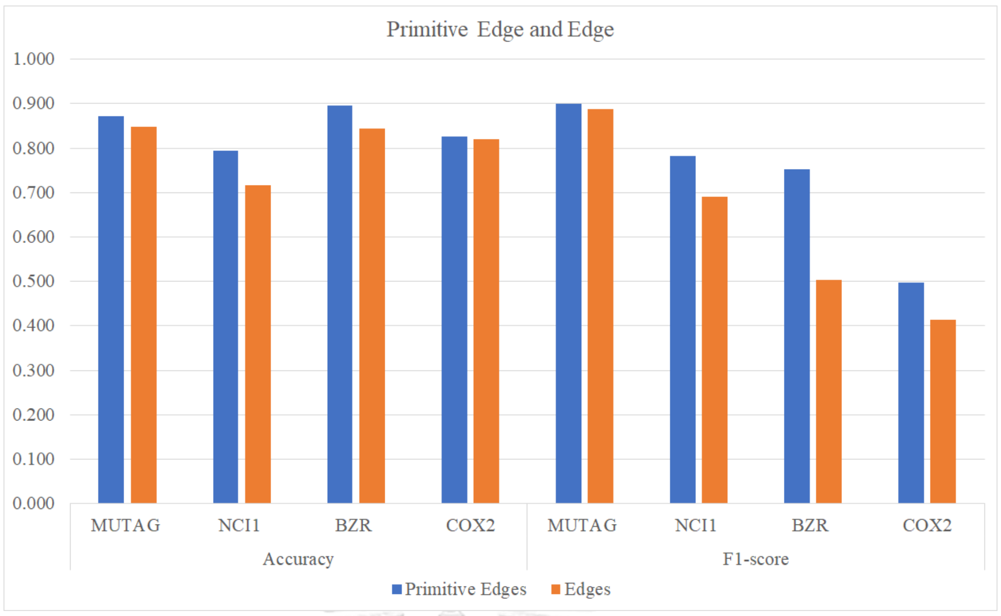


Figure 4.8: The graph shows comparison between the average accuracy and the average F1-score on each data set of Primitive Edge and Edge

The Wilcoxon signed-ranks test between these methods is shown in Table 4.7.

Table 4.7: Test statistics to comparing between Edge and Primitive Edge.

Accuracy		F1-score	
R^+	717	R^+	731
R^-	103	R^-	89
z	4.126	z	4.315
p-value	$1.842*10^{-5}$	p-value	$7.993*10^{-6}$

The null hypothesis could be rejected on both accuracy and F1-score at a significant level of 0.001. Thus, our method based on the numbers of edges between each type of vertex labels had greater performance than the method based on the numbers of edges between each type of vertex labels on original graphs.

From the results, using the information from primitive structure graphs can improve the performance of graph classification in 3 out of 4 similarity measures in the experiment.

4.4.3 CPU Runtime

The runtime for our method in the first and the second step (the primitive structure selection algorithm, and the primitive structure extraction algorithm) was high, due to the time complexity of a sub-graph isomorphism problem which was one of NP-class problems, especially on the large data set such as NCI1 such that total runtime was 6157 seconds (about 1 hour and 42 minutes).

Table 4.8: CPU runtime in second for primitive structure selection and primitive structure extraction on each data set.

Methods	Data sets			
	MUTAG	NCI1	BZR	COX2
Primitive structure selection	50	3338	471	643
Primitive structure extraction	21	2819	103	107
Total runtime	71	6157	574	750

However, most of the CPU runtime for similarity measures and classification on our method was lower than the runtime of the method based on original graphs. Because the concept of our algorithm was to group common structures and considered them as vertices, then the primitive structure graphs were usually smaller than the original graphs. Thus, it needed less time to calculated the similarity measure. The runtime of the method based on the numbers of edges between each type of vertex labels on primitive structure graphs (Primitive Edges) was higher than the method with the same idea of information extraction (Edges). Since the number of vertex labels on the primitive structure graph, which was a type of primitive structures, was always larger than the number of vertex labels (type of atoms) on the original graph.

Table 4.9: Average CPU runtime in second for similarity measures and classification on each data set.

Methods	Data sets			
	MUTAG	NCI1	BZR	COX2
WL-ST	0.43	51.93	1.23	1.68
WL-SP	3.34	57.57	26.84	31.23
SP	0.23	22.55	1.29	1.44
Edges	0.07	36.87	0.19	0.18
Primitive WL-ST	0.35	40.17	0.95	0.97
Primitive WL-SP	0.64	68.31	7.36	8.15
Primitive SP	0.10	13.95	0.47	0.55
Primitive Edges	0.08	655.00	0.80	0.81

CHAPTER V

CONCLUSIONS AND FUTURE WORK

This chapter provides conclusion of this thesis and the possibility of the future work.

5.1 Conclusions

The algorithm to extract the information from the data of biochemical compounds encoded as graph-structured data by analyzing their common sub-graphs called "primitive structure" and creating a new graph called "primitive structure graph" was proposed. The primitive structure graph was a graph where its vertices were primitive structures and its edges were connections between each primitive structure. This new graph obtained from our approach could be used with any previously proposed graph kernels or other similarity measures to extract their information. The information obtained by this method could be used with the support vector machine to classify a graph. Primitive structures and their connection in the original graph were the primary concern for this study. They were a piece of essential information for measuring the similarity of biochemical compounds. The algorithm was tested with four benchmark data sets and four similarity measures. The result shows that this algorithm can improve the accuracy, and the F1-score of classification methods when combined with suitable similarity measures.

5.2 Future work

Directions for the future work are to apply this algorithm on the labeled graph with high-dimensional vertex labels (graph with multidimensional and real value vertex labels) which provide more information about the structure of biochemical compounds. Moreover, we need to find a better kernel method to handle the problem of imbalanced data set as well as a multi-class problem. Since the real-world data set of biochemical compounds mostly has non-interested class data more than interested class data and some data sets have more than two classes.

REFERENCES

[1] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, “Weisfeiler-lehman graph kernels,” *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, 2011.

[2] T. Gärtner, P. A. Flach, and S. Wrobel, “On graph kernels: Hardness results and efficient alternatives,” in *COLT*, 2003.

[3] K. M. Borgwardt and H.-P. Kriegel, “Shortest-path kernels on graphs,” *Fifth IEEE International Conference on Data Mining (ICDM’05)*, pp. 8 pp.–, 2005.

[4] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, “Efficient graphlet kernels for large graph comparison,” in *AISTATS*, 2009.

[5] J. Ramon and T. Gärtner, “Expressivity versus efficiency of graph kernels,” 2003.

[6] M. Niepert, M. Ahmed, and K. Kutzkov, “Learning convolutional neural networks for graphs,” *ArXiv*, vol. abs/1605.05273, 2016.

[7] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. P. Liu, and S. Jaiswal, “graph2vec: Learning distributed representations of graphs,” *ArXiv*, vol. abs/1707.05005, 2017.

[8] A. J.-P. Tixier, G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, “Classifying graphs as images with convolutional neural networks,” *ArXiv*, vol. abs/ 1708.02218, 2017.

[9] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," *KDD : proceedings. International Conference on Knowledge Discovery and Data Mining*, vol. 2016, pp. 855–864, 2016.

[10] G. Nikolentzos, G. Siglidis, and M. Vazirgiannis, "Graph kernels: A survey," *ArXiv*, vol. abs/1904.12218, 2019.

[11] D. Haussler, "Convolution kernels on discrete structures," 1999.

[12] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[13] B. E. Boser, I. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *COLT '92*, 1992.

[14] H.-C. Ehrlich and M. Rarey, "Systematic benchmark of substructure search in molecular graphs - from ullmann to vf2," *Journal of Cheminformatics*, vol. 4, pp. 13 – 13, 2012.

[15] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 1367–1372, 2004.

[16] V. Carletti, P. Foggia, and M. Vento, "Vf2 plus: An improved version of vf2 for biological graphs," in *GbRPR*, 2015.

[17] J. Gasteiger, "Handbook of chemoinformatics," 2003.

[18] B. Mishra, "Molecular (graph) characteristics of some hydrocarbons through graph theory,"

[19] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, "Benchmark data sets for graph kernels," 2016.

[20] A. Debnath, R. L. L. de Compadre, G. Debnath, A. Schusterman, and C. Hansch, “Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds,” 1991.

[21] N. Wale, I. A. Watson, and G. Karypis, “Comparison of descriptor spaces for chemical compound retrieval and classification,” *Knowledge and Information Systems*, vol. 14, pp. 347–375, 2006.

[22] J. J. Sutherland, L. A. O’Brien, and D. F. Weaver, “Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships,” *Journal of chemical information and computer sciences*, vol. 43 6, pp. 1906–15, 2003.

[23] G. Siglidis, G. Nikolentzos, S. Limnios, C. Giatsidis, K. Skianis, and M. Vazirgiannis, “Grakel: A graph kernel library in python,” *arXiv preprint arXiv: 1806.02193*, 2018.

[24] T. P. Peixoto, “The graph-tool python library,” *figshare*, 2014.

[25] A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using networkx,” 2008.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] F. Wilcoxon, “Individual comparisons by ranking methods,” 1945.

[28] J. Demsar, “Statistical comparisons of classifiers over multiple data sets,” *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.

APPENDIX

APPENDIX A : Primitive structure selection

```
import networkx as nx
def primitive_selection(candidate_list):
'''
Input: candidate_list (set of candidate sub-graph)
Output: primitive_structure (set of primitive structure)
'''
all_primitive_structure = []
#Search for all candidate graph that appear in data set
for candidate in candidate_list:
    for g in MUTAG_graph:
        for i in  nx.algorithms.isomorphism.GraphMatcher(g,
            candidate).subgraph_isomorphisms_iter():
            sub = nx.Graph.subgraph(g,i)
            all_primitive.append(sub)
#Delete duplicate primitive structure
primitive_structure = []
nm = nx.algorithms.isomorphism.categorical_node_match('label',
    node_label_set)
for g in all_primitive:
    dif_g = True
    for base_g in primitive_structure:
        if nx.algorithms.isomorphism.GraphMatcher(g,base_g,nm).
            is_isomorphic():
        dif_g = False
        break
    if dif_g:
        primitive_structure.append(g)
return primitive_structure
```

APPENDIX B : Primitive structure extraction

```python
import graph_tool.all as gt
def primitive_extraction(graph_db,primitive_structure,primitive_structure_label):
'''
Input:  graph_db (set of considered graph)
       primitive_structure (set of primitive structure)
       primitive_structure_label (set of primitive structure label)
Output: graph_primitive (set of graph created by primitive structure)
'''
graph_primitive = []
#Scan all graph in data set to create the new one
for g in graph_db:
    all_unique_set = []
    #Identify all primitive structure in considered graph
    for idx, base_g in enumerate(primitive_structure):
        iso_list = []
        for i in gt.subgraph_isomorphism(base_g,g,vertex_label=(base_g.vertex_properties['label'],g.vertex_properties['label']),generator=True):
            iso_list.append(frozenset(i))
        all_unique_set.append(set(iso_list))
    graph_obj = []
    #Select only primitive_structure that not overlay with other
    for idx_obj, set_obj in enumerate(all_unique_set):
        for obj_i in set_obj:
            check_overlay = False
            for obj_j in graph_obj:
            i_intersect_j = obj_i.intersection(obj_j[0])
            if len(i_intersect_j) > len(obj_i)/2 or len(i_intersect_j) == len(obj_i):
                check_overlay = True
        if not(check_overlay):
            graph_obj.append([obj_i,primitive_structure_label[idx_obj
                ]])
    #Create new graph from primitive structures and their connection
    go = nx.empty_graph()
    for i,[obj_i,label_i] in enumerate(graph_obj):
        go.add_node(i)
        go.nodes[i]["label"] = label_i
        for j,[obj_j,label_j] in enumerate(graph_obj):
            if i!= j and obj_i.intersection(obj_j) != set():
                go.add_edge(i,j)
    graph_primitive.append(go)
return graph_primitive
```

APPENDIX C : Import benchmark data set

```
from grakel import datasets
import networkx as nx
def Load_graph(names):
'''
Input:  names (name of a benchmark dataset in TUD Datasets)
Output: graph_db (list of graph in networx format)
       dataset.target (list of target class for each graph)
'''
dataset = datasets.fetch_dataset(names, verbose=False,
    prefer_attr_nodes=False)
dataset_attr = datasets.fetch_dataset(names, verbose=False,
    prefer_attr_nodes=True)
graph_data = dataset.data
graph_data_attr = dataset_attr.data
graph_db = []
#Convert graph data from dict to networkx.graph
for i,g_el in enumerate(graph_data):
    g = nx.from_edgelist(g_el[0])
    if g_el[1] != {}:
        for v in g.nodes:
            g.nodes[v]["label"] = g_el[1][v]
    if graph_data_attr[i][1] != {}:
        for v in g.nodes:
            g.nodes[v]["attribute"] = graph_data_attr[i][1][v]
    if g_el[2] != {}:
        for e in g.edges:
            g.edges[e]["label"] = g_el[2][e]
    graph_db.append(g)
return graph_db, dataset.target
```

APPENDIX D : Example code for classification with SVM by using the Weisfeiler-Lehman sub-tree kernel as a similarity measure.

```python
from grakel import GraphKernel
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score
from sklearn.model_selection import train_test_split, GridSearchCV,
    StratifiedKFold
import numpy as np
'''
Input:  grakel_db (list of graph in grakel dictionary format)
        classes_db (list of target class for each graph)
        rs (list of random_state seed)
Output: acc (list of accuracy for each experiment)
        pre (list of precision for each experiment)
        rec (list of recall for each experiment)
        f1  (list of f1-score for each experiment)
'''
acc = []
pre = []
rec = []
f1 = []
for i in rs:
    best_score = 0
    for iter in [1,2,3,4,5]:
        X_train, X_test, y_train, y_test = train_test_split(np.array(
            grakel_db), np.array(classes_db) , test_size = ts,
            random_state = i)
        wl_kernel = GraphKernel(kernel=[{"name": "weisfeiler_lehman",
             "n_iter": iter}, {"name": "subtree_wl"}], normalize=True
            )
        K_train = wl_kernel.fit_transform(X_train)
```

```
            parameters = {'C':np.logspace(-2,3,10)}
            clf = GridSearchCV(SVC(kernel='precomputed'), parameters, cv=StratifiedKFold(n_splits=5))
            clf.fit(K_train, y_train)
            if clf.best_score_ > best_score:
                best_model = clf
                best_h = iter
                best_kernel = wl_kernel
                best_score = clf.best_score_
                best_test = X_test
        K_test = best_kernel.transform(best_test)
        y_pred = best_model.predict(K_test)
        acc.append(accuracy_score(y_test, y_pred))
        pre.append(precision_score(y_test, y_pred,pos_label=1))
        rec.append(recall_score(y_test, y_pred,pos_label=1))
        f1.append(f1_score(y_test, y_pred,pos_label=1))
```

BIOGRAPHY

Name	Mr. Peemapat Wongsriphisant
Date of Birth	10 January 1996
Place of Birth	Bangkok, Thailand
Education	B.S. (Mathematics), Kasetsart University, 2018