
Neighborhood Preserving Kernels for Attributed Graphs

Asif Salim

Department of Mathematics,
Indian Institute of Space Science and Technology,
Thiruvananthapuram, India
asifsalim.16@res.iist.ac.in

Shiju S. S

Architect - AI & Automation,
Litmus7 Systems Consulting Pvt Ltd,
Cochin, India
shijusnair@gmail.com

Sumitra. S

Department of Mathematics,
Indian Institute of Space Science and Technology,
Thiruvananthapuram, India
sumitra@iist.ac.in

Abstract

We describe the design of a reproducing kernel suitable for attributed graphs, in which the similarity between the two graphs is defined based on the neighborhood information of the graph nodes with the aid of a product graph formulation. We represent the proposed kernel as the weighted sum of two other kernels of which one is an R-convolution kernel that processes the attribute information of the graph and the other is an optimal assignment kernel that processes label information. They are formulated in such a way that the edges processed as part of the kernel computation have the same neighborhood properties and hence the kernel proposed makes a well-defined correspondence between regions processed in graphs. These concepts are also extended to the case of the shortest paths. We identified the state-of-the-art kernels that can be mapped to such a neighborhood preserving framework. We found that the kernel value of the argument graphs in each iteration of the Weisfeiler-Lehman color refinement algorithm can be obtained recursively from the product graph formulated in our method. By incorporating the proposed kernel on support vector machines we analyzed the real-world data sets and it has shown superior performance in comparison with that of the other state-of-the-art graph kernels.

1 Introduction

Graph-structured data is used to represent information generated in fields like bioinformatics, chemoinformatics, social network, natural language processing, computer vision, etc. Hence the development of efficient methods for the analysis of such graphical data is very essential. Kernel methods [1], [2] is an efficient technique for analyzing the non-euclidean data. Applying kernel methods on such data requires a kernel function which is formulated using the characteristic properties of the data. In the case of graphs, many approaches have been developed for designing valid kernels. R-convolution kernel [3] which is a kernel design framework for structured data has been well utilized for designing graph kernels. In this approach, the graphs are decomposed into several parts and separate kernel functions are used to process these parts. Shortest path kernel [4] and Graph-hopper kernel [5] etc. are examples of this. Along with this, another approach is to design a specific feature representation process for the graphs in the form of a vector or to design

an explicit mapping of the graphs to a reproducing kernel Hilbert space. For these approaches, graph structures such as random walks [6], [7], [8], subtree patterns [9], [10], [11], etc. are utilized.

Another design paradigm associated with the graph kernel design is the valid optimal assignment kernel framework [12]. While the R-convolution framework [3] processes the subgraph structures of the argument graphs one against each other, the assignment kernel framework assigns a bijection between the subgraphs, and kernel processing between them happens in one against one as per assigned bijection. The Weisfeiler-Lehman optimal assignment kernel [12], optimal assignment kernels on molecular graphs [13], etc. are examples.

Generally, the graph data has labels and attributes associated with each of its nodes and edges, where the label is usually a single character while the attribute is a vector. For example, in the bioinformatics or chemoinformatics applications, the label on the nodes and edges can be atomic symbol and type of bonds respectively whereas the attribute information on edges and nodes can be related to that of physical and chemical properties. Thus the main challenge in designing a kernel for attributed graphs is to define a similarity function that could capture all the characteristic properties of data by making use of the label as well as attribute information along with the overall graph structure. By taking into account all these aspects, we have designed a kernel, whose description is presented in this paper.

Most of the existing graph kernels process only the label information in the nodes and edges ignoring the attributes. The kernel we propose uses the node and edge labels as well as their attribute information for finding its values. For that, we formulated the reproducing kernel as the weighted sum of two kernels of which one is an R-convolution kernel that processes the attribute information and the other is an optimal assignment kernel that processes the label information. By incorporating the proposed kernels on support vector machines, we experimentally verified its performance using real-world data sets selected from the chemoinformatics domain.

The paper is organized as follows. The related works are discussed in Section 2. The notations used are explained in Section 3. The graph kernel frameworks used in our kernel design are discussed in Section 4. We define the concept of neighborhood preserving regions and the neighborhood preserving kernel based on edges in Section 5. The extension of these concepts to the shortest paths is discussed in Section 6. Experiments and related discussions are in Section 7 and the conclusions in Section 8.

2 Related works

For processing the attributed graphs, the first attempts were based on random walk kernels. Borgwardt et.al [14] improvised the random walk kernels proposed in [7, 8] to process the attribute information. They used the direct product graph proposed by Gartner et.al [7] to identify similar walks and the kernel is formulated with the help of a modified version of the adjacency matrix of the direct product graph. This modified adjacency matrix encodes the corresponding node and edge kernels in each step of a walk in the direct product graph that corresponds to a similar walk in the argument graphs. The random walks of different lengths are taken into consideration in the higher powers of the adjacency matrix and the final kernel value is calculated using this property. But kernels based on random walks are slower in computation as the length of the walks increase and can suffer from tottering [15]. Tottering is a phenomenon that results in artificially high similarity values by the repeated visits of the nodes in the graphs.

Zhang et.al [16] used the return probabilities of random walks of fixed lengths to make node embeddings or a vector representation for the nodes. In this vector, the $i - th$ component of the vector corresponds to the probability of returning to the concerned node with an i -step random walk. This vector information is enriched with attribute information. The combined vector is then mapped to a reproducing kernel Hilbert space through kernel mean embedding. The graph kernel is then defined using these mappings. One shortcoming with kernels based on random walks is that random walks processed may not be structurally equivalent, that is, as far as the regions corresponding to these random walks in the graphs are considered, the resulting subgraphs and hence the nodes correlated need not be structurally equivalent.

Kernel designs based on shortest paths were used as instances of R-convolution kernels. Shortest path kernel [4] accounts for all shortest paths in the form of a triplet $(u, d(u, v), v)$ where u, v

are nodes, and $d(u, v)$ is the shortest path distance between the nodes. All such triplets were convolved against each other in the kernel definition. Note that in this case, only end nodes are considered for processing attribute information. Feragen et.al [5] extended these concepts to all nodes encountered in the shortest paths and designed GraphHopper kernel that process equal length shortest paths. Note that in these kernels well-defined correspondence between node pairs are absent, i.e, the source node of a shortest path can be paired with either the source or sink node of the counterpart shortest path and hence the same problem with the nodes in between as well since the kernel design does not use labels on nodes and edges. Hence the structural differences characterized by labels are getting neglected.

The subgraph matching kernel proposed by Kriege et.al [17] enumerates through subgraphs that preserves subgraph isomorphism. They proposed a product graph definition from which the subgraph isomorphisms in the argument graphs can be identified by enumerating through the cliques in this product graph. The kernel component corresponding to a clique is then defined as the product of the corresponding node and edge kernels and the final graph kernel value is the summation across all such cliques up to a predefined size. Although this kernel maintains structural equivalences, it suffers from a high runtime issue.

The ordered decomposition directed acyclic graph (ODD) kernel [18] creates a representation of the graph as a bag of well defined directed acyclic graphs (DAG). Each node pair in each DAG pairs are then processed in a similarity function to derive the kernel value. Note that in this feature extraction procedure, it is difficult to establish a correspondence between the processed regions of the argument graphs.

Graph invariant kernels[19] proposed by Orsini et.al is a framework that incorporates graph invariants with R-convolution design. This kernel is defined in terms of the summation of every possible node kernel between two graphs. Each node pair kernel value is multiplied by a weight as well. The weight is determined by the structural importance of that node pair in a particular R-convolution decompositions of the graphs. The weight function also incorporates a similarity value characterized by node invariants such as Weisfeiler-Lehman labels, spectral colors, etc. The maintenance of structural correspondences in this approach depends upon the selection of node invariants and it can have higher runtime problems depending upon the R-convolution relation design.

The approaches discussed above are based on certain graph substructures and each having a specific feature extraction process from them. Another fundamental approach applied to attributed graphs is based on the discretization of attributes. Propagation kernel [20] by Neumann et.al, Hash graph kernels by Morris et.al [21] etc. are examples. In Propagation kernels, the attributes are discretized using a hashing mechanism. The information in the graphs is propagated at each iteration across the nodes. Each iteration gives a unique instance of the graph that carries structural information and the final kernel value is defined as the summation of individual kernels at these iterations. For efficient computation, it makes use of locality sensitive hashing. It is also helpful in handling missing attributes as well. In Hash graph kernels, the idea is to convert the attributes to discrete ones using well-defined hashing functions. The hashing mechanism is done in several iterations and kernels values at these iterations are summed up. Although both of the above approaches are scalable, it is coming at the cost of transformation of the original attributes.

3 Notation

We define an undirected graph G as (V, E) , where V is the set of nodes and $E = \{(u, v) : u \text{ is incident on } v\}$ is the set of edges. Define a mapping $l : V \cup E \rightarrow \Sigma$, such that $l(v)$ is the label associated with a node v , $l((u, v))$ is the label associated with an edge (u, v) , and (Σ, \leq) is a total ordered set that consists of node and edge labels.

The shortest path between the node u_1 and u_n is represented as $\Pi(u_1, u_n)$ and its length as $|\Pi(u_1, u_n)|$ where, the length of the path is defined as number of edges involved. The graph density is defined as the ratio of number of edges in the graph to the maximum possible edges where parallel edges are not assumed. \mathcal{X} is assumed to be a separable metric space consisting of discrete structures like string, trees or graphs and (Σ_C, \leq) be a total ordered set containing Weisfeiler-Lehman (WL) refined labels.

4 Methods

The discussion about the kernel frameworks we used in our design is given below.

4.1 R-convolution kernels

The R-convolution kernel formulated by Haussler [3] is a generalized framework for processing structured data. This framework involves the decomposition of data into its constituent parts and a kernel is defined in terms of such decomposition. The shortest path kernel [4], subgraph matching kernel [17], etc. are examples of R-convolution kernels.

Let $x \in \mathcal{X}$. Assume that x can be decomposed to D number of components or parts. Let $\tilde{x} = \{x_1, \dots, x_D\}$ be this decomposition, where $x_i \in X_i$ and $X_i, 1 \leq i \leq D$, be a non empty, separable metric space. Define a relation R as

$$R = \{(\tilde{x}, x) \in X_1 \times \dots \times X_D \times \mathcal{X} \mid \tilde{x} \text{ are parts of } x\}$$

That is, R is true iff \tilde{x} is a valid decomposition of x . Now it is possible to define the inverse relation, $R^{-1}(x) = \{\tilde{x} : R(\tilde{x}, x) = 1\}$. Note that R is finite if R^{-1} is finite $\forall x \in \mathcal{X}$. Then the R-convolution kernel $K_{R_{conv}} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is defined as

$$K_{R_{conv}}(x, y) = \sum_{\tilde{x} \in R^{-1}(x)} \sum_{\tilde{y} \in R^{-1}(y)} \prod_{i=1}^D k_i(x_i, y_i)$$

where (x_i, y_i) is the i^{th} component of (\tilde{x}, \tilde{y}) and $k_i(.,.) : X_i \times X_i \rightarrow \mathbb{R}, 1 \leq i \leq D$ is the kernel corresponding to i^{th} component.

4.2 Valid optimal assignment kernel framework

The optimal assignment kernel is defined as follows: Let $\mathcal{X}^n, n \in \mathbb{N}$ denote the set of all n -element subsets of \mathcal{X} and $B(X, Y)$, the set of all bijections between X, Y , where $X, Y \in \mathcal{X}^n$. The optimal assignment kernel $K_B^k : \mathcal{X}^n \times \mathcal{X}^n \rightarrow \mathbb{R}$ is defined as

$$K_B^k(X, Y) = \max_{\beta \in B(X, Y)} W(\beta) \quad \text{where} \quad W(\beta) = \sum_{(x, \beta(x))} k(x, \beta(x)), \quad (1)$$

where k is a strong kernel defined on $\mathcal{X} \times \mathcal{X}$ [12]. The above concept can be extended to the case where the underlying domain consists of sets with unequal cardinality by adding dummy objects d to the smaller set, where $k(d, x) = 0, \forall x \in \mathcal{X}$. The concept of strong kernel is discussed below.

4.2.1 Strong kernels and hierarchies

The strong kernel can be explained using two different concepts:

1. In terms of an inequality constraint on the kernel values.

A function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^+$ is called strong kernel if $k(x, y) \geq \min \{k(x, z), k(z, y)\} \quad \forall x, y, z \in \mathcal{X}$. That is, once we consider x and y , there is no other element z in \mathcal{X} where both x and y are more similar to z than themselves.

2. In terms of hierarchy defined on the domain of the kernel.

The hierarchy can be constructed in the form of a rooted tree, T as follows. We assume that the leaves of T corresponds to elements in \mathcal{X} . It has to be noted that tree forms a nested structure, that is, each inner node in T apart from the leaves corresponds to a subset of elements in \mathcal{X} . For example, in an example hierarchy in Figure 1, the node v is an inner vertex that corresponds to nodes a, b in \mathcal{X} and node r corresponds to node c in \mathcal{X} .

A weight is defined to each of the nodes in T . Let $w : V(T) \rightarrow \mathbb{R}^+$ be a weight function such that $w(v) \geq w(p(v))$ for all v in T where $p(v)$ denotes parent node of the node v , and $V(T)$ is the set of nodes in T . The tuple (T, w) is referred as hierarchy. It has to be noted that base kernels which are strong in $\mathcal{X} \times \mathcal{X}$ mentioned earlier can be derived from the hierarchy structure [12].

With the above concepts, Kriege et.al[12] proved that a kernel k on $\mathcal{X} \times \mathcal{X}$ is strong if and only if it is induced by a hierarchy on \mathcal{X} .

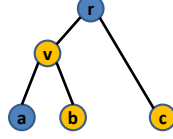


Figure 1: Example for hierarchy

It has to be noted that the hierarchy corresponding to a base kernel k is not unique. Considering this, to prove that a kernel is an optimal assignment on a structured data, it is enough to prove that the underlying base kernel is strong and there exists a hierarchy which induces the corresponding base kernel from which equation 1 can be calculated.

Examples for optimal assignment kernels are kernels defined on vertices and edges, WL-optimal assignment kernel by Kriege et.al [12], and optimal assignment kernels on molecular graphs by Fröhlich et.al [13].

5 Neighborhood Preserving Kernel

In this section we discuss the design aspects of the neighborhood preserving kernel we formulated.

5.1 Neighborhood preserving regions

The neighborhood preserving (NP) kernels process the structurally similar regions of the argument graphs. In our design, the concept of structural similarity is defined using neighborhood preserving property, whose concept is given below.

To formalize the neighborhood preserving regions, 1-dimensional WL color refinement [22] is done on the graphs. The WL color refinement algorithm creates a representation for each node in the graph in the form of a string. The node label of the node is the first element of the string and the remaining elements are node labels of the neighboring nodes in lexicographic order. This string representation is assigned a new label through a hashing function and the above process is repeated with the new assigned label. If the graphs are devoid of any node labels, the same label for all nodes can be assumed using some strategy: for example, the node label can be taken as the degree of the node in the first iteration. A product graph [7] is then constructed to find the structural similar regions, where the product graph is defined as follows :

Definition 5.1. [Direct product graph]: Let $G = (V, E)$ and $G' = (V', E')$ be two graphs under consideration. The direct product graph, $G_P = G \times G'$, represented as (V_P, E_P) is defined as

$$\begin{aligned}
 V_P &= \{(u, u') \in V \times V' : l_C(u) = l_C(u')\} \\
 E_P &= \left\{ ((u, u'), (v, v')) : (u, v) \in E \wedge (u', v') \in E' \wedge \right. \\
 &\quad \left. l((u, v)) = l((u', v')) \right\}
 \end{aligned}$$

An example is shown in figure 2(a),(b),(c) of two sample graphs, their WL coloring, and product graph formation. From the product graph the set of neighborhood preserving or structurally similar edges of the two given graphs based on preserving its structure and node/edge labels information can be deduced as per the definition given below.

Definition 5.2. [Neighborhood preserving edges]:

The set of neighborhood preserving edges in graph G with respect to G' , is

$$\mathcal{S}_{G,G'} = \{(u, v) \in E : ((u, u'), (v, v')) \in E_P\}$$

An example of the extraction of neighborhood preserving edges of the graphs is shown in figure 2 (d). Note that the WL color refinement algorithm basically processes the information about neighborhood of the nodes. Now, corresponding to each $(u, v) \in \mathcal{S}_{G,G'}$ there exists atleast one $(u', v') \in E'$ such that u, u' as well as v, v' has the same label and neighborhood information. Hence the edges are termed as *neighborhood preserving edges*. This helps to process kernel

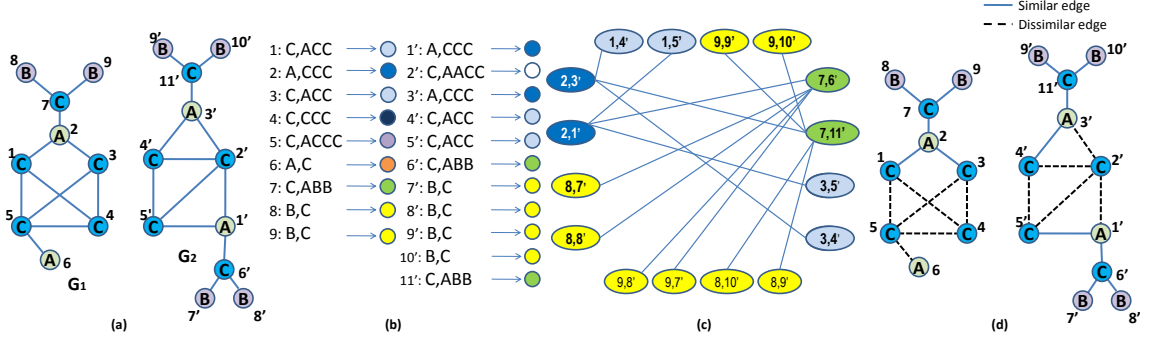


Figure 2: (a) Two sample graphs G_1 and G_2 , (b) WL color refinement, (c) direct product graph, (d) separation of neighborhood preserving or structurally similar edges (bold lines) and dissimilar edges (dashed lines)

computation with subgraphs having well-defined correspondences in terms of the neighborhood of the component nodes and it also gives a visualization of the structural similarity between the graphs.

We designed a kernel that defines the similarity of the argument graphs with the aid of neighborhood preserving edges, whose description is given below.

5.2 Neighborhood preserving edge kernel

Let κ_V and κ_E be positive semi definite kernels defined on $V \times V'$ and $E \times E'$ respectively and $\lambda \in \mathbb{R}^+$ be a weight function. Then the neighborhood preserving edge kernel, K_{NPE} , is defined as

$$K_{NPE}(G, G') = \lambda(G, G') \sum_{(u,v) \in E} \sum_{(u',v') \in E'} \kappa_V(u, u') \cdot \kappa_E((u, v), (u', v')) \cdot \kappa_V(v, v') \quad (2)$$

where κ_V is defined in terms of the attribute information of argument nodes and κ_E is defined as

$$\kappa_E((u, v), (u', v')) = \delta(l(u, v), l(u', v')) \cdot \delta(l_C(u), l_C(u')) \cdot k((u, v), (u', v')) \cdot \delta(l_C(v), l_C(v'))$$

where λ is a normalization constant, δ is the Kronecker delta function and k is a valid kernel defined in terms of attributes on edges. If no attributes are present on edges, the value of k can be taken as 1. Note that delta functions ensure that the edges processed for the kernel computation are from neighborhood preserving edges only and hence the name for the kernel. The function of weight λ is to give a normalizing effect to avoid artificially high similarity value if the argument graph sizes vary unevenly and if some WL refined labels in a graph are very high compared to its counterpart resulting in a large number of edges of a particular type.

Theorem 1. *The neighborhood preserving edge kernel is a valid kernel.*

Proof. We prove K_{NPE} as a R-convolution kernel [3]. Let Σ^3 the set of strings of length three formed by members of Σ_C and Σ , that is, $\Sigma^3 = \{xyz | x, z \in \Sigma_C, x \leq z, y \in \Sigma\}$. Consider two graphs G and G' and $\Lambda \subseteq \Sigma^3$ where

$$\Lambda = \bigcup_{(u,v) \in E} l_C(u) \odot l(u, v) \odot l_C(v) \cap \bigcup_{(u',v') \in E'} l_C(u') \odot l(u', v') \odot l_C(v')$$

where \odot is a concatenation operator, $l_C(u) \odot l(u, v) \odot l_C(v)$, $l_C(u') \odot l(u', v') \odot l_C(v') \in \Sigma^3$. We call elements of Λ as *WL refined edge address*.

Now we define a relation R corresponding to each member $\epsilon \in \Lambda$. Let $R(e, G^\dagger, G; \epsilon)$ be a relation where $R(e, G^\dagger, G; \epsilon) = 1$ iff G^\dagger is the graph obtained from G if an edge e corresponding to a member

ϵ in Λ is removed. Now $R^{-1}(G; \epsilon) = \{(e, G^\dagger) : R(e, G^\dagger, G; \epsilon) = 1\}$ is the decomposition of a graph into an edge with *WL refined edge address* ϵ and rest of the graph. Now K_{NPE} is defined as

$$K_{NPE}(G, G') = \sum_{\epsilon \in \Lambda} \frac{1}{|E_\epsilon \times E'_\epsilon|} \sum_{\substack{(e, G^\dagger) \in R^{-1}(G; \epsilon) \\ (e', G'^\dagger) \in R^{-1}(G'; \epsilon)}} k_{edge}(e, e') \times k_{triv}(G^\dagger, G'^\dagger) \quad (3)$$

where E_ϵ, E'_ϵ are set of edges in G, G' corresponding to *WL refined edge address* ϵ , $E_\epsilon \times E'_\epsilon$ being their cartesian product, k_{triv} is a trivial kernel whose value is always 1 and

$$k_{edge}(e, e') = \kappa_V(u, u') \times k(e, e') \times \kappa_V(v, v') \quad (4)$$

where $e = (u, v)$ and $e' = (u', v')$. Now k_{edge} is a valid kernel [23]. Also, $|E_\epsilon \times E'_\epsilon|$ is a well defined function and hence K_{NPE} is a R convolution kernel. \square

The effect of the cardinality of Λ has to be considered and hence included the weight $\frac{1}{|E_\epsilon \times E'_\epsilon|}$ in equation (3). The λ in equation (2) can be considered as the counterpart of this term.

By constructing a product graph, the information used in K_{NPE} kernel can be extracted as explained below.

Theorem 2. *Each edge in the direct product graph corresponds to a product calculation in convolution operation defined in equation (3) except for a few edges of the form $((u, u'), (v, v')) \in E_P$ where $l_C(u) = l_C(v) = l_C(u') = l_C(v')$ and $l(u, v) = l(u', v')$.*

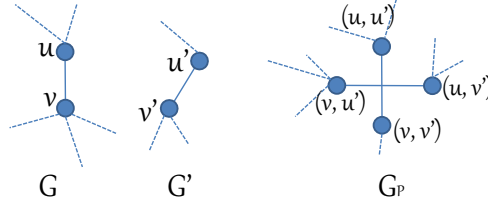


Figure 3: Two nodes u, v in the graph G and two nodes u', v' in the graph G' with the same WL refined labels and corresponding edges having a similar label. The product graph G_P containing an additional pair of nodes and an additional edge.

Proof. Consider a particular edge $(u, v) \in E$ in a graph G which is neighborhood preserving. Note that the nodes in the direct product graph contain all combination of nodes from graphs G and G' whose label and neighborhood information is identical. Hence the edge (u, v) is a part of set of edges of the form $((u, u'), (v, v'))$ in direct product graph where $u' \in \{x \in V' : l_C(x) = l_C(u)\}$, $v' \in \{y \in V' : l_C(y) = l_C(v)\}$, $(u', v') \in E'$ and $l(u, v) = l(u', v')$. Hence these set of edges correspond to a convolution of edge $(u, v) \in E$ with its counterpart edges in E' . Similarly we can find out other convolutions corresponding to every edge in G .

But a duplication arises in a special case when $l_C(u) = l_C(u') = l_C(v) = l_C(v')$ and $l(u, v) = l(u', v')$. Consider such a case as shown in the figure 3. We can see that in the direct product graph, G_P , four nodes are created with two edges. But as per convolution operation, the edge pair (u, v) and (u', v') shall be counted only once. Hence avoiding this duplication, K_{NPE} can be calculated from G_P . \square

The NPE kernel can be found out for multiple iterations of WL color refinement. In this case the final kernel is taken as the sum of kernels in the individual iteration.

The neighborhood preserving edge kernel processes node and edge attributes with the labels on them acting as a guidance for the R-convolution process. For processing the node and edge labels alone, another kernel named neighborhood preserving optimal edge assignment kernel is formulated, whose description is given in the next section. The objective is to analyze the importance of the label information.

5.3 Neighborhood preserving optimal edge assignment kernel

The neighborhood preserving optimal edge assignment kernel K_{NPO} processes only node labels and defined as

$$K_{NPO}(G, G') = \sum_{e \in \Lambda} \text{minimum}(|E_e|, |E'_e|) \quad (5)$$

The minimum value is taken as it represents a normalized score of structural similarity. Note that for K_{NPO} computation also, the edges processed are only neighborhood preserving ones. Note that K_{NPO} can also be defined for multiple WL color refinement steps where kernel value in individual iteration is summed up.

With the aid of an optimal assignment kernel $K_B(E, E')$, we prove $K_{NPO}(G, G')$ as a valid kernel. We define $K_B(E, E')$ as,

$$K_B(E, E') = \max_{\beta \in B(E, E')} \sum_{(u, v), \beta(u, v)} \sum_{i=1}^h k_b^i((u, v), \beta(u, v)) \quad (6)$$

where $B(E, E')$ is the set of all bijections between the members of the set E and E' corresponding to the base kernel k_b^i at iteration: i of WL color refinement, $(u, v) \in E$, and $\beta(u, v) \in E'$. Here we assume that the sets among E and E' that contain the least cardinality have enough dummy members to make up for the difference between their cardinality and WL iteration is done h times.

The base kernel k_b^i is defined as,

$$k_b^i((u, v), (u', v')) = \begin{cases} 1, & \text{if } l_C^i(u) = l_C^i(u') \wedge l_C^i(v) = l_C^i(v') \\ & \wedge l((u, v)) = l((u', v')) \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

where $l_C^i(.)$ indicates WL color of the concerned node at iteration: i of the WL color refinement. The base kernel defined above is strong as the cardinality of its range set is two. [12].

The hierarchy tree corresponding to base kernel k_b can be constructed in the following way. Consider the application of the WL color refinement algorithm on the given graphs. Corresponding to each WL refined edge address obtained in the first iteration, a root node is created. The elements of WL refined edge addresses obtained for iteration $i > 1$, forms the nodes for the level i of the hierarchy tree. As the nodes in the higher level are generated from the nodes in the lower level, a parent-child relationship between two subsequent levels can be established. Thus a tree structure T can be built by adding edges between parent and children.

Lemma 1. $K_B(E, E')$ can be calculated from the hierarchy.

Proof. With the help of the hierarchy T , K_B can be calculated as a histogram intersection kernel [12] as follows. Corresponding to each graph G , a histogram vector G_V of length $V(T)$ is created, where $V(T)$ is the set of nodes of the tree. The i^{th} element of G_V is the number of times it produces the i^{th} element of $V(T)$ during the WL color refinement algorithm procedure. The histogram vector calculation is explained in Algorithm 1.

Note that the above algorithm corresponds to a tree traversal for each edge in a graph starting from the leaf nodes. Since the hierarchy contains all possible occurrences of *WL refined edge addresses* in h iterations of WL color refinement, each visit of a node in T at level i with respect to the edge under consideration correspond to an occurrence of that particular *WL refined edge address* characterizing the node at iteration: i . Hence the tree traversals as per the algorithm give the number of occurrences of *WL refined edge addresses* in the entire h number of WL iterations, that is, G_V . Let G_V and G'_V be the histogram vectors of graphs G and G' . Then $K_B(E, E')$ in equation (6) can be calculated as histogram intersection kernel of G_V and G'_V [12], that is,

$$K_B(E, E') = \sum_{i=1}^{|G_V|} \text{minimum}(G_V(i), G'_V(i)) \quad (8)$$

Algorithm 1: Computation of histogram vector G_V

Input : The hierarchy T for h iterations of WL color refinement algorithm and a graph $G = (V, E)$.

Output: Histogram vector G_V

```
1 Assign a unique location starting from 1 to  $|V(T)|$  to all the nodes in  $T$ 
2 Initialize  $G_V$  as  $|V(T)| \times 1$  vector
3 for every edge  $e$  in  $E$  do
4   Find the leaf node  $n$  in  $T$  corresponding to WL refined edge addresses of edge  $e$  at WL
   iteration:  $h$ 
5    $i = h$ 
6   while  $i \geq 1$  do
7      $G_V(\text{location}(n)) = G_V(\text{location}(n)) + 1$ 
8      $n = \text{parent}(n)$ 
9      $i = i - 1$ 
10  end
11 end
```

It is straightforward now to establish that $K_B(E, E')$ for h iterations of WL color refinement steps is the summation of K_{NPO} kernels in individual refinement steps. That is,

$$K_B(E, E') = \sum_{i=1}^h K_{NPO}^i(G, G')$$

where K_{NPO}^i is the NPO kernel at iteration i . □

On the basis of above discussion, it is clear that neighborhood preserving optimal edge assignment kernel is a valid optimal assignment kernel.

It has to be noted that K_{NPO} in actual experiments is calculated as explained in Section 5.8.

5.4 Neighborhood preserving kernel definition

The neighborhood preserving (NP) kernel for h iterations of WL color refinement algorithm is defined as

$$K(G, G') = \alpha \sum_{i=1}^h K_{NPE}^i(G, G') + (1 - \alpha) \sum_{i=1}^h K_{NPO}^i(G, G') \quad (9)$$

where $\alpha \in (0, 1)$ is a tuning parameter, K_{NPE}^i is the NPE kernel and K_{NPO}^i is the NPO kernel defined for i^{th} iteration. The purpose of α is to have a trade off between the two components of NP kernel.

5.5 Other neighborhood preserving kernels

It can be seen that to ensure a graph kernel to process only on the neighborhood preserving regions, it is enough to take features from the edges that have a common *WL refined edge address* in the graphs. The application of WL refinement iteration on WL-edge kernel [10] results in the process of the neighborhood preserving regions. However, this kernel does not process attribute information. Hence the neighborhood preserving edge (NPE) kernel can be considered as a generalization of WL-edge kernel.

In the case of WL-subtree kernel and WL-shortest path kernel [10], the regions processed may include non-neighborhood preserving edges as well. However, WL-shortest path kernel can be made to a neighborhood preserving kernel by imposing an additional constraint such that the shortest paths considered for feature extraction shall contain only neighborhood preserving edges.

It is evident that the walks in the formulated product graph corresponds to neighborhood preserving walks in the argument graphs. Hence random walk kernel [14] can be made neighborhood preserving. It is possible to distinguish the shortest paths from the neighborhood preserving walks. Hence it is possible to make GraphHopper kernel neighborhood preserving and WL-shortest path

kernel [10] can be generalized to attributed graphs in the same way as the proposed NPE kernel generalizes the WL edge kernel.

If we insert d -edges in the direct product graph definition, as explained in [17], the cliques corresponds to neighborhood preserving subgraph isomorphism and hence subgraph matching kernel [17] can be made neighborhood preserving.

5.6 Recursive computation of NPE kernel from product graph

We prove with the following theorem that the neighborhood preserving edge kernel can be computed at each WL iteration from the product graph defined for the previous iteration which is obtained in a recursive fashion, i.e, the initial product graph formulation alone is enough to find out the product graphs in proceeding WL iterations without explicitly finding them.

Theorem 3. *The product graph at any iteration >1 of the WL color refinement algorithm is the subgraph of the product graph defined for the previous iteration.*

Proof. The product graph (G_{P_i}) in iteration : $i > 1$ can be obtained from the product graph $(G_{P_{i-1}})$ in iteration: $(i-1)$ by deleting certain edges. Suppose G_{P_i} is the product graph obtained by deleting the whole edges of the form $((u, v), (u', v'))$ in $G_{P_{i-1}}$ where $l_{C_i}(u) \neq l_{C_i}(u')$ and $l_{C_i}(v) \neq l_{C_i}(v')$ where l_{C_i} is the WL alphabet in iteration : i . Our argument is that the edges that can occur in G_{P_i} is already embedded in $G_{P_{i-1}}$ except the deleted edges based on the above rule.

Suppose there exists an edge $((y, y'), (z, z'))$ in G_{P_i} that does not exist in $G_{P_{i-1}}$. That is label and neighborhood of y, y' and z, z' with respect to WL labels $l_{C_{i-1}}$ at the iteration $h = i - 1$ are identical. If there neighborhood are identical at the iteration $h = i - 1$, they would have formed nodes in $G_{P_{i-1}}$ and so the edge between them. Hence the edge $((y, y'), (z, z'))$ in G_{P_i} is already embedded in $G_{P_{i-1}}$. Hence G_{P_i} formed with the above edge deletion rule is the subgraph of $G_{P_{i-1}}$. □

5.7 Computational complexity analysis

With efficient sorting techniques, WL color refining for one iteration can be done in $\mathcal{O}(m)$ operations, where m is the number of edges. NPE kernel defined in equation 2 requires $\mathcal{O}(m^2)$ operations. Considering d be the dimension of attributes, the base kernel computation is of $\mathcal{O}(d)$. Hence NPE kernel is computed in $\mathcal{O}(h \times N^2(m^2 \times d))$ where N is the dataset size.

The computational complexity of K_{NPO} is $\mathcal{O}(N^2 \times |\Sigma^3|^3)$, where $|\Sigma^3|^3$ is the maximum possible size of Λ bounded by m^2 for a pair of graphs. This makes the overall computational complexity of NP kernel to be $\mathcal{O}(h \times N^2(m^2 \times d))$.

5.8 Algorithms of the pairwise and global kernel computations of NP kernel

We can compute K_{NPE} and hence $K(G, G')$ in two ways.

1. In pairwise manner with product graph.

Although the kernel computation with product graph using recursion property can have n^2 nodes in the worst case where n is the number of nodes, the number of edges are bounded by $c \times m^2$ where c is a factor that accounts for the edges that result in duplication as explained in Theorem 2. The computation steps are detailed in Algorithm 2.

2. In global manner.

We can create a list of *WL refined edge addresses* derived from $\Lambda \subseteq \Sigma^3$ that occurs across the dataset and a pre-computed feature information data can be formed for each graph listing the edges corresponding to each *WL refined edge address*. Then the kernel can be computed by iterating through this precomputed feature information data. This method by default provides the provision to calculate K_{NPO} because it only requires processing the cardinality of individual refined addresses in the feature information data. The computation steps are detailed in Algorithm 3.

Algorithm 2: Pairwise computation

Input : Two graphs $G = (V, E)$ and $G = (V', E')$, number of WL refinement iterations h **Output**: $K_{NPE}(G_i, G_j)$ and $K_{NPO}(G_i, G_j)$

```
1 Initialize  $K_{NPE}(G_i, G_j) = 0$ 
2 while  $i \leq h$  do
3   Do WL color refinement on the graphs
4   Create direct product graph  $G_P = (V_P, E_P)$  for  $h = i$  avoiding the duplicate edges.
5   Initialize  $\Lambda(key, value)$  as empty dictionary,  $k = 1$ 
6   Initialize  $k_v$  and  $\omega$  as empty lists with values zero,  $K_{NPE}(G_i, G_j) = 0$ 
7   for every node  $n_i = (u_i, u'_i)$  in  $|V_P|$  do
8     for every node  $n_j = (v_j, v'_j)$  in  $|V_P|$  do
9       if  $(u_i, v_j) \in V$  and  $(u'_i, v'_j) \in V'$  then
10        Find WL refined edge address,  $\epsilon$ 
11        if  $\epsilon \notin \Lambda$  then
12           $\Lambda = \Lambda \oplus \epsilon$ 
13           $key(\epsilon) = k$ 
14           $k_v(k) = k_v(k) + \kappa_V(u_i, u'_i) \times \kappa_E((u_i, v_j), (u'_i, v'_j)) \times \kappa_V(v_i, v'_i)$ 
15           $\omega(k) = \omega(k) + 1$ 
16        else
17           $k_v(key(\epsilon)) = k_v(key(\epsilon)) + \kappa_V(u_i, u'_i) \times \kappa_E((u_i, v_j), (u'_i, v'_j)) \times \kappa_V(v_i, v'_i)$ 
18           $\omega(key(\epsilon)) = \omega(key(\epsilon)) + 1$ 
19        end
20      end
21    end
22  end
23  for  $i = 1$  to  $length(k_v)$  do
24     $K_{NPE}(G_i, G_j) = K_{NPE}(G_i, G_j) + k_v(i) / \omega(i)$ 
25  end
26  Initialize  $\Lambda(key, value)$  as empty dictionary,  $k = 1$  and  $K_{NPO}(G_i, G_j) = 0$ 
27  for every graphs  $G_i = (V_i, E_i)$  under consideration do
28    for every edge  $e = (v_p, v_q)$  do
29      Find WL refined edge address,  $\epsilon$ 
30      if  $\epsilon \notin \Lambda$  then
31         $\Lambda = \Lambda \oplus \epsilon$ 
32         $key(\epsilon) = k$ 
33         $F(G_i)\{k\} = 1$ ,  $F(G_i)$  is a vector (initialized to zero) which has  $|\Lambda|$  elements and in
           each such element, number of occurrence of the concerned WL refined edge
           address  $\epsilon$  is stored
34         $k = k + 1$ 
35      else
36         $F(G_i)\{key(\epsilon)\} = F(G_i)\{key(\epsilon)\} + 1$ 
37      end
38    end
39  end
40  for every component  $k$  in  $\Lambda$  do
41     $K_{NPO}(G_i, G_j) = K_{NPO}(G_i, G_j) + minimum(|F(G_i)\{k\}|, |F(G_j)\{k\}|)$ 
42  end
43   $i = i + 1$ 
44 end
```

Algorithm 3: Global computation for one iteration of WL color refinement

Input : The graph dataset D

Output: The neighborhood preserving edge kernel matrix K_{NPE} and optimal edge assignment kernel matrix K_{NPO}

```
1 Do WL color refinement algorithm on the graphs
2 Initialize  $\Lambda(key, value)$  as empty dictionary and  $k = 1$ 
3 for every graph,  $G_i = (V_i, E_i)$  in  $|D|$  do
4   for every edge  $e = (v_p, v_q)$  in  $E_i$  do
5     Find WL refined edge address,  $\epsilon$ 
6     if  $\epsilon \notin \Lambda$  then
7        $\Lambda = \Lambda \oplus \epsilon$ 
8        $key(\epsilon) = k$ 
9        $F(G_i)\{k\} = (v_p, v_q)$ ,  $F(G_i)$  is a tensor which has  $|\Lambda|$  elements and in each such
        element, end vertices of the edges are stored as a list corresponding to the
        concerned WL refined edge address  $\epsilon$ 
10       $k = k + 1$ 
11    else
12       $F(G_i)\{key(\epsilon)\} = F(G_i)\{key(\epsilon)\} \oplus (v_p, v_q)$ 
13    end
14  end
15 end
16 Initialize  $K_{NPE} = 0$  and  $K_{NPO} = 0$ 
17 for every graph,  $G_i = (V_i, E_i)$  in  $|D|$  do
18   for every graph,  $G_j = (V_j, E_j)$  in  $|D|$  do
19     if  $i \leq j$  then
20       Initialize  $k_v = 0$ 
21       for every element  $k$  in  $\Lambda$  do
22         if  $F(G_i)\{k\} \cap F(G_j)\{k\} \neq \emptyset$  then
23           for every members  $e = (v_p, v_q)$  in  $F(G_i)\{k\}$  do
24             for every members  $e = (v'_p, v'_q)$  in  $F(G_j)\{k\}$  do
25                $k_v = k_v + \kappa_V(v_p, v'_p) \times \kappa_E((v_p, v_q), (v'_p, v'_q)) \times \kappa_V(v_q, v'_q)$ 
26             end
27           end
28            $K_{NPE}(i, j) = K_{NPE}(i, j) + \frac{k_v}{|F(G_i)\{k\}| \times |F(G_j)\{k\}|}$ 
29            $K_{NPO}(i, j) = K_{NPO}(i, j) + \text{minimum}(|F(G_i)\{k\}|, |F(G_j)\{k\}|)$ 
30         end
31       end
32        $K_{NPE}(j, i) = K_{NPE}(i, j)$ 
33        $K_{NPO}(j, i) = K_{NPO}(i, j)$ 
34     end
35   end
36 end
```

6 Neighborhood preserving shortest path kernel

It is straight forward to extend the concepts of neighborhood preserving edge kernel defined in Section 5.2. One disadvantage with NPE kernel is that the information that gets processed is in terms of edges only, even though the neighborhood preserving regions are connected. Hence if we process larger subgraphs, the kernel measure is much more accurate. Shortest paths are a good candidate for analyzing these larger connected regions.

Let κ_V be a positive semi definite kernel defined on $V \times V'$. We assume P, P' as the sets that contain shortest paths in graphs G, G' respectively. Then the neighborhood preserving shortest path kernel, (K_{NPS}) , is defined as,

$$K_{NPS}(G, G') = \sum_{\Pi(u_1, u_n) \in P} \sum_{\Pi(u'_1, u'_n) \in P'} \kappa_V(u_1, u'_1) \times k_\delta(\Pi(u_1, u_n), \Pi(u'_1, u'_n)) \times \kappa_V(u_n, u'_n) \quad (10)$$

k_δ can be defined as a psd kernel as follows.

$$k_\delta(\Pi(u_1, u_n), \Pi(u'_1, u'_n)) = \begin{cases} 1, & \text{if } |\Pi(u_1, u_n)| = |\Pi(u'_1, u'_n)| \wedge \\ & \sum_{i=1}^n l_C(u_i) = l_C(u'_i) \wedge \\ & \sum_{i=1}^{n-1} l(u_i, u_{i+1}) = l(u'_i, u'_{i+1}) \\ 0, & \text{otherwise} \end{cases} \quad (11)$$

In the above definition edges involved in the shortest paths are compared for neighborhood preserving property. Note that the nodes in shortest paths are fixed based on the total order in Σ_C and for kernel computation only attributes in source and sink nodes are considered. We can modify this to have kernel value computation of attributes of the nodes in between as well.

The above kernel can be proved as a positive semi-definite kernel in the same way as the NPE kernel. We can formulate the R-convolution relation as a decomposition of the shortest path and the rest of the graph. For each shortest path, we can assign an address like we assign an edge with a *WL refined edge address*. The address can be a string where WL labels of the nodes and edge labels involved in the concerned shortest path are concatenated. With this setting, as in the case of NPE kernel in Section 5.2, we can define an R-convolution kernel corresponding to these addresses.

6.1 Computational complexity analysis

The complexity associated with finding shortest paths for a single graph is of $\mathcal{O}(V^2)$. For kernel computation, the shortest paths have to be compared against each other which is of $\mathcal{O}(V^2)$ and along with each comparison, the base kernel has to be computed twice for source and sink nodes. If the attributes are of dimension d , this makes the kernel computation $\mathcal{O}(V^2 \times d)$. So the overall computational complexity for a set of N graphs is $\mathcal{O}(NV^2 + hN^2V^2d)$.

7 Experiments

The efficiency of the proposed neighborhood preserving kernels was analyzed by subjecting them on real-world data sets and compared its performance with state-of-the-art graph kernels namely: Shortest path(SP) kernel [4], GraphHopper(GH) kernel [5], RetGK kernels [16], Graph invariant kernel (GIK) [19], Propagation kernels [20] and Hash graph kernels [21].

The components of the proposed NP kernel is analyzed separately, i.e, when $\alpha = 1$ in Equation 9 it corresponds to the NPE kernel component that processes the attributes and when $\alpha = 0$ in Equation 9 it corresponds to the NPO kernel component that processes the labels. This helps in evaluating the role of attributes and labels separately and also the effectiveness of NP kernel where both information are utilized.

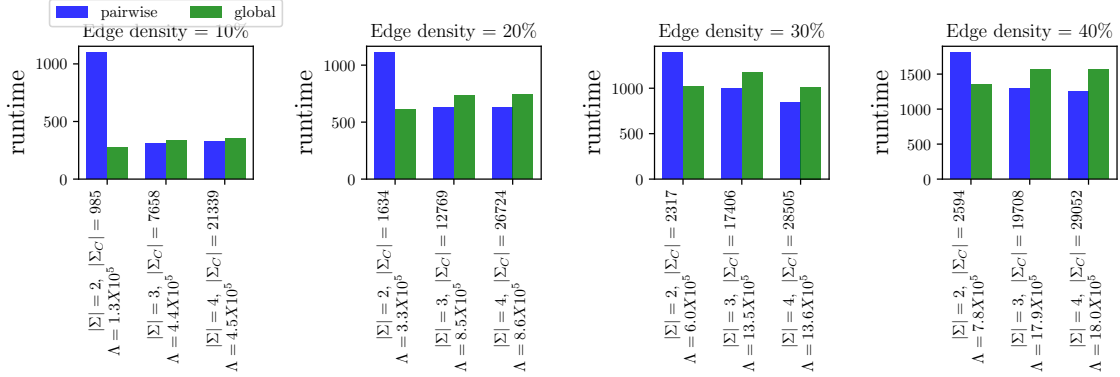


Figure 4: Runtime comparison of pairwise and global computation of NP kernel for 100 synthetic graphs at graph density = 10%, 20%, 30% and 40% respectively.

Table 1: Classification accuracy of the proposed kernels with state-of-the-arts.

Kernel	PROTEINS	ENZYMES	BZR	COX2	DHFR	SYN.new
SP	73.42 ± 1.11	66.58 ± 4.06	85.76 ± 2.35	79.88 ± 1.73	79.52 ± 2.40	86.72 ± 3.68
GH	73.19 ± 1.79	66.33 ± 2.78	82.90 ± 2.73	79.66 ± 1.17	76.65 ± 3.21	88.81 ± 3.41
RetGK-I	75.94 ± 1.79	65.67 ± 3.07	85.58 ± 2.20	78.19 ± 0.47	80.83 ± 2.10	97.08 ± 1.54
RetGk-II	74.69 ± 1.60	62.34 ± 2.94	85.76 ± 1.61	78.25 ± 0.35	81.66 ± 2.41	96.94 ± 1.47
GIK	72.36 ± 2.17	52.32 ± 3.89	86.31 ± 2.25	79.68 ± 2.41	81.25 ± 2.56	91.73 ± 2.22
Prop-diff	72.60 ± 2.32	38.19 ± 2.27	78.46 ± 0.59	77.94 ± 0.47	72.58 ± 0.78	48.59 ± 1.17
Prop-WL	74.23 ± 1.80	44.85 ± 1.63	78.92 ± 0.41	78.25 ± 0.35	73.41 ± 0.53	46.38 ± 1.39
HGK-WL	74.69 ± 1.98	64.30 ± 3.37	81.48 ± 1.84	78.50 ± 0.57	75.47 ± 2.40	81.00 ± 3.69
HGK-SP	75.57 ± 1.89	62.60 ± 3.00	82.38 ± 1.79	78.55 ± 0.65	76.61 ± 2.72	96.38 ± 1.92
NPE _(α=1)	71.08 ± 2.80	64.93 ± 2.97	87.13 ± 1.98	82.36 ± 2.02	82.70 ± 2.35	99.51 ± 0.67
NPO _(α=0)	72.74 ± 2.02	45.67 ± 3.24	88.81 ± 1.70	80.71 ± 2.92	81.07 ± 2.89	97.81 ± 1.53
NP	73.66 ± 2.55	58.10 ± 3.16	89.12 ± 2.03	81.16 ± 2.13	83.98 ± 2.35	99.74 ± 0.64
NPS	73.87 ± 2.47	58.94 ± 3.58	89.24 ± 2.18	81.37 ± 2.26	84.16 ± 2.43	99.85 ± 0.71

7.1 Datasets

The classification datasets used for analysis were ENZYMES, PROTEINS, BZR, COX2, DHFR and SYNTHETICnew. ENZYMES is a data set of 600 protein tertiary structures obtained from the BRENDA enzyme database [24]. PROTEINS is a dataset of chemical compounds with two classes (enzyme and non-enzyme) introduced in [25]. BZR, COX2, and DHFR [26] are taken from the Graph data repository [27]. SYNTHETICnew is a synthetic dataset introduced in [5]. The datasets are all of the binary class except ENZYMES which has six classes.

7.2 Experimental setup

The validation process was carried out in the following way. Using the hold-out technique 70% of the data points were assigned for training and the remaining for testing. 10 fold cross-validation was done on training data for selecting the hyperparameters. A model was then built using the entire training data and its performance was tested on the testing data. The above process was repeated 30 times and the results reported were averaged over these 30 iterations to nullify the effects of fold assignments.

The classification algorithm used was SVM (with Libsvm implementation [28]). The penalty parameter C of SVM was searched in the range $[2^{-7}, 2^{-5}, \dots, 2^{15}]$. The performance parameter used was accuracy. The number of iterations of WL color refinement algorithm was chosen from $\{1, 2, 3\}$ which is fixed through cross validation. The experiments were done in machine with Intel Xeon i5 2.4 GHz CPU with 80 GB RAM.

We wrote the code for SP and GIK kernel while GH, RetGK kernels, and Hash graph kernel were done with the codes published by authors. Propagation kernel implementation and (its hyper-parameter selection) was done by the codes published by authors of the GH kernel. The linear and Gaussian kernel ($k(x, y) = e^{-\beta \|x-y\|^2}$) where $\beta = 1/d$, (d the dimension of attribute information) were used as the base kernels within the state-of-the-arts. The coding was done in Matlab except for Hash graph kernel which is in Python.

For GIK kernels, WL coloring was taken as vertex invariant. Two variants of the propagation kernel was implemented. In the 'Prop-diff' variant, the propagation scheme used is diffusion [20]. In the 'Prop-WL' variant labels of the nodes are first hashed and the WL propagation scheme is used. Total variance distance was used as the hashing function in both propagation schemes. The bin width of the hash function was set to 10^{-5} and the number of propagation steps for both variants was fixed through cross validation from $\{1, 2, \dots, 5\}$. RetGK kernels used also have two variants. RetGK-I is the one with an explicit feature map in RKHS and RetGK-II is the one with approximated mapping. For both approaches, 50 steps of random walks are assumed. For Hash graph kernels, WL subtree (HGK-WL) and shortest path (HGK-SP) kernels [10] were employed as the base kernels and hashing function used is 2-stable Locality sensitive hashing (LSH) with bin width 1 and label and hashed attribute information was propagated separately following the practice of authors. Number of WL refinement steps was fixed through cross validation from $\{1, 2, \dots, 5\}$.

For the SYNTHETICnew dataset, the node labels were given identical labels discarding the original continuous type values, and attributes are used as such. For the proposed kernels and GH kernel, the result reported is the best among the case between Linear and Gaussian kernels where they are employed as base kernels. Runtime reported for these kernels is done when Gaussian kernel is employed as the base kernel.

7.3 Runtime experiments with pairwise and global computation of NP kernel

This experiment is done to evaluate the pairwise and global computation schemes in calculating the NP kernel as explained in Section 5.8. For this experiment, 4 artificial datasets of 100 graphs with 300 nodes and graph density 10%, 20%, 30%, and 40% respectively were created. Each dataset has experimented 3 times with the nodes being selecting a random label out of an alphabet Σ of size 2, 3, and 4 labels respectively, and edge labels are assumed to be identical. The experiment is done for 2 steps of WL color refinement where calculation for pairwise computation is done as per Theorem 2 and Theorem 3. The runtime (in seconds) for both approaches is plotted against the size of WL alphabet Σ_C and size of Λ for the label size, $|\Sigma| = 2, 3$, and 4 respectively. It is assumed that unit time is taken for both approaches for base kernel computations. The variation in graph density is studied since it is the number of edges that affects the computation time as explained in Section 5.7.

It can be seen from Figure 4 that when $|\Sigma_C|$ is small global computation takes less amount of time. In this case, more nodes are sharing common WL labels and this results in smaller $|\Lambda|$. But the product graph involves lots of nodes and hence larger computation time for pairwise computation scheme. But as $|\Sigma_C|$ becomes larger, pairwise computation time is much better compared to the global. In this case, the nodes sharing common WL labels are relatively less and it will result in a larger $|\Lambda|$. In comparison with the smaller number of nodes in the product graph, global computation of these larger Λ feature information takes more time than pairwise computation. Hence it is actually $|\Lambda|$ that influences more than $|\Sigma_C|$ because as $|\Sigma_C|$ increases, there is no corresponding increase in $|\Lambda|$ as seen from the experiments especially for cases where $|\Sigma| = 3, 4$.

In the case of datasets used global computation is used. But from the experiments, it is evident that in the case of datasets with larger $|\Lambda|$ pairwise computation may take lesser time than the global computation.

7.4 Results and discussion

The accuracy with standard deviation obtained are tabulated in Table 1 and runtime (wall-clock time) in Table 2. We did experiments with the proposed kernels with NPE kernel alone ($\alpha = 1$) and NPO kernel alone ($\alpha = 0$) as well as with the formal NP kernel definition, NPS kernel and compared the results with that of state-of-the-arts. The best results are given in bold letters.

Table 2: Runtime of the proposed kernels with state-of-the-arts.

Kernel	PROTEINS	ENZYMES	BZR	COX2	DHFR	SYNnew
SP	>5 day	>3 day	>2 day	>2 day	>3 day	>2 day
GH	13' 1"	3' 4"	58"	1' 22"	3' 7"	4' 2"
RetGK-I	2' 57"	37"	17"	24"	1' 15"	30"
RetGk-II	2.5"	1"	0.7"	0.9"	1.4"	13.3"
GIK	22' 34"	8' 43"	7' 16"	12' 49"	30' 39"	35' 11"
Prop-diff	7"	5"	3.5"	3.7"	7.5"	4"
Prop-WL	12"	5.3"	4"	5.2"	9"	8"
HGK-WL	3' 42"	1' 25"	1' 2"	1' 9"	1' 55"	2' 11"
HGK-SP	3' 8"	1' 6"	48"	52"	1' 28"	1' 38"
NPE _($\alpha=1$)	56' 51"	7' 16"	3' 18"	5' 25"	11' 37"	17' 33"
NPO _($\alpha=0$)	6' 37"	41"	13"	19"	37"	45"
NP	56' 42"	7' 10"	3' 12"	5' 20"	11' 31"	17' 27"
NPS	>1 day	65' 10"	23' 37"	47' 1"	151'	198'

The NP and NPS kernels have a significant improvement over state-of-the-arts in the case of BZR, COX2, DHFR, and SYNTHETICnew datasets, and in the case of PROTEINS and ENZYMES datasets their performance is reasonably good. Note that NPE kernel which processes attribute information and NPO kernel which processes only labels where α tuning is not required performs better than state-of-the-arts in BZR, COX2, and SYNTHETICnew datasets. In the case of the DHFR dataset, the NPE kernel outperforms the state of the arts whereas the performance of the NPO kernel is on par with them. For datasets except for ENZYMES and COX2, NP kernel augmented with NPO kernel performs significantly better than NPE kernel. This validates our argument about the need for processing the label and attribute information independently. This also gives evidence to the fact that in the case of graph data analysis, if attributes are present it cannot be neglected. This is important since most of the kernels developed in this regard could only process labels. In comparison with the performance of the proposed kernels with the discretization algorithms (Propagation and HGK kernels), the proposed designs perform better although the runtime of discretization based approaches are low.

The runtime of NP and NPS kernels are reported for the global computation approach with Gaussian kernel being the base kernel and it is reasonable compared with state-of-the-arts. Considering the runtime, propagation kernels are the fastest. But they use discretized attribute information and hence their performance is lower than state-of-the-arts. Although the runtime of RetGK kernels is better, NP kernel processes the label and attribute information independently which makes the performance of those better. Since NP and NPS kernel establishes a well-defined correspondence between subgraphs, its performance is better than GraphHopper kernel despite the difference in runtime. In comparison with the shortest path kernel and Graph invariant kernels, the runtime for NP kernels is better. The performance of the GIK kernel are close with that of the NP kernel while the performance of NP kernels compared to the shortest path kernel is better in the datasets, an exception being ENZYMES.

Note that the runtime of the NPS kernel can be improvised with the efficient computation approaches like the strategies introduced in [5]. The difference in the runtime of NPE and NP kernels are negligible. The reason is because of the global computation scheme. When we calculate the NPE kernel, the steps involved by default provides a way to calculate the NPO kernel as well, as described in line: 29 of Algorithm 3. But once we are only concerned with the computation of NPO kernel, the runtime is significantly reduced since the processing of attributes is not required.

8 Conclusion

We designed kernels based on the neighborhood preserving property where the similarity between argument graphs is well defined and visualized. The neighborhood preserving concept helps to define a well-defined correspondence between subgraphs computed during kernel computations. The kernels can be recursively computed from the product graph that helps in an efficient computation procedure for large alphabets occurring in the WL color refinement algorithm. The proposed kernel which independently processes the attribute and label information is found to

be very effective in the graph classification tasks. Their performance is found to be superior in comparison with the state-of-the-arts.

References

- [1] A. J. Smola and B. Schölkopf, *Learning with kernels*. Citeseer, 1998.
- [2] J. Shawe-Taylor and N. Cristianini, *Kernel methods for pattern analysis*. Cambridge university press, 2004.
- [3] D. Haussler, “Convolution kernels on discrete structures,” tech. rep., Citeseer, 1999.
- [4] K. M. Borgwardt and H.-P. Kriegel, “Shortest-path kernels on graphs,” in *Fifth IEEE International Conference on Data Mining (ICDM’05)*, pp. 8–pp, IEEE, 2005.
- [5] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt, “Scalable kernels for graphs with continuous attributes,” in *Advances in Neural Information Processing Systems*, pp. 216–224, 2013.
- [6] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, “Graph kernels,” *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.
- [7] T. Gärtner, P. Flach, and S. Wrobel, “On graph kernels: Hardness results and efficient alternatives,” in *Learning Theory and Kernel Machines*, pp. 129–143, Springer, 2003.
- [8] H. Kashima, K. Tsuda, and A. Inokuchi, “Marginalized kernels between labeled graphs,” in *ICML*, vol. 3, pp. 321–328, 2003.
- [9] J. Ramon and T. Gärtner, “Expressivity versus efficiency of graph kernels,” in *First international workshop on mining graphs, trees and sequences*, pp. 65–74, Citeseer, 2003.
- [10] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, “Weisfeiler-lehman graph kernels,” *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.
- [11] L. Bai, L. Rossi, Z. Zhang, and E. Hancock, “An aligned subtree kernel for weighted graphs,” in *International Conference on Machine Learning*, pp. 30–39, 2015.
- [12] N. M. Kriege, P.-L. Giscard, and R. Wilson, “On valid optimal assignment kernels and applications to graph classification,” in *Advances in Neural Information Processing Systems*, pp. 1623–1631, 2016.
- [13] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell, “Optimal assignment kernels for attributed molecular graphs,” in *Proceedings of the 22nd international conference on Machine learning*, pp. 225–232, ACM, 2005.
- [14] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, “Protein function prediction via graph kernels,” *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.
- [15] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, “Extensions of marginalized graph kernels,” in *Proceedings of the twenty-first international conference on Machine learning*, p. 70, ACM, 2004.
- [16] Z. Zhang, M. Wang, Y. Xiang, Y. Huang, and A. Nehorai, “Retgk: Graph kernels based on return probabilities of random walks,” in *Advances in Neural Information Processing Systems*, pp. 3964–3974, 2018.
- [17] N. Kriege and P. Mutzel, “Subgraph matching kernels for attributed graphs,” in *Proceedings of the 29th International Conference on Machine Learning (ICML-12)* (J. Langford and J. Pineau, eds.), ICML ’12, (New York, NY, USA), pp. 1015–1022, Omnipress, July 2012.
- [18] G. Da San Martino, N. Navarin, and A. Sperduti, “Tree-based kernel for graphs with continuous attributes,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, pp. 3270–3276, July 2018.
- [19] F. Orsini, P. Frasconi, and L. De Raedt, “Graph invariant kernels,” in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- [20] M. Neumann, N. Patricia, R. Garnett, and K. Kersting, “Efficient graph kernels by randomization,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 378–393, Springer, 2012.

- [21] C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel, "Faster kernels for graphs with continuous attributes via hashing," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 1095–1100, Dec 2016.
- [22] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI, Series*, vol. 2, 1968.
- [23] C. Cortes, P. Haffner, and M. Mohri, "Rational kernels: Theory and algorithms," *Journal of Machine Learning Research*, vol. 5, no. Aug, pp. 1035–1062, 2004.
- [24] I. Schomburg, A. Chang, C. Ebeling, M. Gremse, C. Heldt, G. Huhn, and D. Schomburg, "Brenda, the enzyme database: updates and major new developments," *Nucleic acids research*, vol. 32, no. suppl_1, pp. D431–D433, 2004.
- [25] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of molecular biology*, vol. 330, no. 4, pp. 771–783, 2003.
- [26] 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, no. 6, pp. 1906–1915, 2003.
- [27] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, "Benchmark data sets for graph kernels," 2016.
- [28] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.