

Graph kernels

Abhishek Gupta

A Thesis Submitted to
Indian Institute of Technology Hyderabad
In Partial Fulfillment of the Requirements for
The Degree of Master of Technology



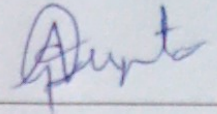
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Department of Computer Science and Engineering

June 2012

Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.



(Signature)

(Abhishek Gupta)

CS10M01

(Roll No.)

Approval Sheet

This thesis entitled Graph Kernels by Abhishek Gupta is approved for the degree of Master of Technology/ Doctor of Philosophy from IIT Hyderabad.

ch sobhan baby
ch sobhan baby

-Name and affiliation-

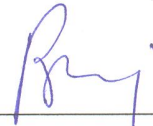
Examiner

-Name and affiliation-

Examiner

-Name and affiliation-

Adviser



-Name and affiliation-

Chairman

Acknowledgements

Many individuals contributed in many different ways to the completion of this thesis. I am deeply grateful for their support, and thankful for the unique chances they offered me.

I'm greatly thankful to my supervisor Dr. Naveen Sivadasan for his valuable guidance, constant encouragement and timely suggestions. I would like to make a special mention of the excellent facility provided by my institute IIT Hyderabad.

More than to anyone else, I owe to the love and support of my family. My father Sanjay Gupta, my mother Indira Gupta and my younger brother Abhijeet Gupta.

Abstract

Data Mining and Machine Learning are in the midst of a “structured revolution” [1]. As we can represent almost anything using graphs, learning and data mining on graphs have become a challenge in various applications. The main algorithmic difficulty in these areas, *measuring similarity of graphs*, has therefore received significant attention in recent past. Graph kernels proposes a theoretically sound and promising approach to the problem of graph comparison. These kernels should respect the information represented by the topology of the graphs, while being efficient to compute. Graph kernel are used in fields like machine learning, data mining, language processing and bioinformatics. Some of the existing graph kernel methods doesn't include topological information or have runtime issues or they do not scale to large graphs. The primary goal of this thesis is to propose a graph kernel which is efficient to compute and can work accurately on large graphs.

In this thesis we analyze existing graph kernels and their drawbacks. Then we propose a graph kernel, based on counting connected size-k graphlets [2]. We conducted experiments on various graphs to test accuracy of our graph kernel.

Contents

Declaration	ii
Approval Sheet	iii
Acknowledgements	iv
Abstract	v
1 Introduction	1
1.1 Graph comparison problem	1
1.2 kernel based graph comparison	2
1.3 Thesis outline	2
2 Preliminaries about kernels	3
2.1 Kernel	3
2.2 Positive semi-definite kernel	3
2.3 Properties of kernel	4
2.4 R-convolution kernel	4
3 Survey on existing graph kernels	6
3.1 Graph kernel	6
3.2 Random walk kernel	6
3.3 Shortest path kernel	7
3.4 Other kernels	8
3.5 A linear time graph kernel for large graphs	9
3.6 Graphlet kernel for large graph comparisons	11
3.6.1 Sampling	11
3.6.2 Bounded degree graph	12
4 Modified graphlet kernel on large graphs	13
4.1 Graphlet vector	13
4.2 Modified graphlet kernel	14
4.2.1 BFS upto depth-d	14
4.2.2 Clustering	15
5 Experimental results	17
6 Discussion	20

Chapter 1

Introduction

The efficient ways to compare graphs and then classify them into groups has been area of interest and research since long. Though graphs can represent complex data structure, they were not used as common data structure for decades because of their high degree of freedom and expressiveness. Efficient comparison of graphs has become feasible because of introduction of graph kernel. The main advantage of defining a kernel on graphs is that it allows us to calculate the similarity score between graphs without explicitly constructing feature space for graphs. Learning from graphs using the kernel methods has been one of the major topics for years, since Haussler introduced the R-convolution kernel [3]. Similarity of graphs has applications in bioinformatics, cheminformatics, language processing, machine learning, sensor network management and social network management.

1.1 Graph comparison problem

The central aim of this thesis is to compare graphs efficiently, known as graph comparison problem.

Definition 1. *Given two graphs G and G' from a graph space, ζ . The graph comparison problem is to find a function,*

$$f: \zeta \times \zeta \rightarrow \mathbb{R}$$

where \mathbb{R} represent the set of real numbers and $f(G, G')$ quantifies graph similarity (or dissimilarity) of G and G' .

Challenges: Similarity of graphs is a well studied problem in various branches of science. There are two well known strategies to compare graphs, namely *graph isomorphism* and *subgraph isomorphism*.

Definition 2. *An isomorphism of graph G and G' is a bijection between the vertex sets G and G'*

$$f: V(G) \rightarrow V(G')$$

such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in G' is known as graph isomorphism.

Subgraph isomorphism problem is a computational task in which two graphs G and H are given as input, and one must determine whether G contains a subgraph that is isomorphic to H . For graph isomorphism problem there is no known polynomial time solution and subgraph isomorphism problem is NP complete. Graph isomorphism is too restrictive way of comparing graphs. Graph isomorphism algorithm outputs *TRUE* for isomorphic graphs, *FALSE* otherwise. Hence, it is rarely used in practice, because few graphs completely match in real-world applications.

1.2 kernel based graph comparison

Most of the standard techniques for comparing graphs suffer from exponential runtime in the worst case. Kernel methods are theoretically well founded in statistical learning theory and shown good empirical results in many applications. Kernel is a similarity measure between objects. Kernel takes two objects (at a time) as input and output a real number, which signify similarity of those objects. Object can be string or vector or graphs.

Kernel which compares similarity of graphs is known as graph kernel. We give two graphs as input to kernel and it outputs similarity score between those graphs. There are several ways in which kernel for graphs can be defined, such as counting common random walks on graph [4] or counting number common subgraphs [2] or counting the number of common cyclic patterns [5], etc.

As kernel compares object in polynomial time, it is preferred over traditional ways to compare graphs. Kernel can be easily embed into support vector machine to classify graphs based on similarity score of graphs.

1.3 Thesis outline

In this thesis we study about preliminaries of kernel in chapter 2, followed by survey on existing graph kernels, which list advantages and drawbacks of existing graph kernels in chapter 3. In chapter 4 we propose modified graph kernel for large graphs based on counting of graphlets [2]. In chapter 5 we show accuracy of our kernel by comparing it with [2]. In chapter 6 we summarize our work and give an outlook to future challenges.

Chapter 2

Preliminaries about kernels

To understand graph kernel, we have to define a kernel. Afterwards, we can extend the definition of kernel to graph kernel.

2.1 Kernel

The idea of finding some non-linear mapping $\Phi : \chi \rightarrow H$, such that we map $x \in \chi$ and $x' \in \chi$, in lower dimensional space, to $\Phi(x)$ and $\Phi(x')$ respectively, in some higher dimensional space and then comparing them by taking dot product of $\Phi(x)$ and $\Phi(x')$ is known as kernel.

Definition 3. Suppose $\langle x, x' \rangle$ denote dot product of x and x' then kernel function k is defined as,

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle$$

Note that as kernel function is defined as dot product of x and x' in some higher dimensional space, it is a symmetric function.

Kernel trick: It is a way of mapping observations from a general set χ into an inner product space H (equipped with its natural norm), *without ever having to compute the mapping explicitly*, in the hope that the observations will gain meaningful linear structure in H .

By using kernel trick we save the computation cost of finding explicit mapping function. It can easily be embedded into standard classification methods which uses dot product to classify objects like SVM (support vector machine).

2.2 Positive semi-definite kernel

To understand which kernel functions can be used to compare objects we need to define few terms.

Gram matrix: Let \mathbb{R} denote set of real numbers and \mathbb{C} denote set of complex number. Given a function $k : \chi^2 \rightarrow \mathbb{K}$ (where $\mathbb{K} = \mathbb{C}$ or $\mathbb{K} = \mathbb{R}$) and patterns $x_1, x_2, \dots, x_m \in \chi$, the $m \times m$ matrix K with elements,

$$K_{ij} = k(x_i, x_j)$$

is called the gram matrix (or kernel matrix) of k with respect to x_1, x_2, \dots, x_m .

Positive semi-definite matrix: A real $m \times m$ matrix K satisfying

$$\sum_{i,j=1}^m r_i \bar{r}_j K_{ij} \geq 0$$

for all $r_i \in \mathbb{R}$ is called positive semi-definite matrix, where \mathbb{R} denote set of complex numbers.

Positive semi-definite kernel: Let \mathbb{N} be the set of natural numbers and χ be a nonempty set. A symmetric function k on $\chi \times \chi$ which for all $m \in \mathbb{N}$ and all $x_1, x_2, \dots, x_m \in \chi$ gives rise to a positive definite Gram matrix is called a positive semi-definite (psd) kernel. From now, often we will refer psd kernel as a *kernel*.

If k is a psd kernel function then only we can construct Hilbert space H with

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle.$$

A Hilbert space is a inner product space, which is complete/closed with respect to inner product. The Hilbert space associated with a kernel is referred to as a *Reproducing Kernel Hilbert Space* (RKHS). It is well known fact that every kernel function is associated with a RKHS and every RKHS is associated with a kernel function. So, we needn't find explicit mapping of objects in higher dimension when we have psd kernel or simply kernel.

Theorem 1. (Mercer's Theorem) *A symmetric function $k(.,.)$ is a kernel if and only if for any finite sample ς the kernel matrix for ς is psd [7].*

2.3 Properties of kernel

Every psd kernel satisfies properties mentioned in [6]. Consider any space ς of samples and kernels $k_1(.,.)$ and $k_2(.,.)$ over ς . Then $k(.,.)$ is a kernel with,

- $k(x, y) = k_1(x, y) + k_2(x, y)$, that is sum of two kernels is a kernel.
- $k(x, y) = k_1(x, y) \cdot k_2(x, y)$, that is product of kernel is a kernel.
- $k(x, y) = \alpha k_1(x, y)$ where $\alpha > 0$
- $k(x, y) = f(x) \cdot f(y)$ for any function f on x
- $k(x, y) = \frac{k_1(x, y)}{\sqrt{k_1(x, x)} \sqrt{k_1(y, y)}}$

Note that this is not exhaustive list, as you can always combine one or more kernels based on above listed properties to get new kernel.

2.4 R-convolution kernel

R-convolution kernels provide a generic way to construct kernels for discrete compound objects. Let $x \in \chi$ be such an object, and $x := (x_1, x_2, x_3, \dots, x_D)$ denote decomposition of x , with each $x_i \in \chi_i$. Define boolean predicate,

$$R: \chi \times \chi \rightarrow \{\text{TRUE}, \text{FALSE}\}$$

where $\chi = \chi_1 \times \chi_2 \times \dots \times \chi_D$ and $R(\hat{x}, x)$ is TRUE whenever \hat{x} is a valid decomposition of x . R-convolution kernel is defined as,

$$k(x, \hat{x}) := \sum_{\substack{x \in R^{-1}(\hat{x}) \\ \hat{x} \in R^{-1}(x)}} \mu(x, \hat{x}) \prod_{i=1}^D k_i(x_i, \hat{x}_i)$$

where, $R^{-1}(x) := \{ \hat{x} | R(\hat{x}, x) = \text{TRUE} \}$, that is all valid decomposition of x , assuming that $R^{-1}(x)$ is countable and μ is a finite measure on $\chi \times \chi$ which ensures that above sum converges.

Chapter 3

Survey on existing graph kernels

3.1 Graph kernel

A Kernel that operates on graphs to measure their similarity is known as graph kernel. These graph kernels belong to larger class of R-convolution kernel defined by [3] (discussed in earlier chapter under R-convolution kernel). In the next section, we discuss various approaches which were proposed in the past.

3.2 Random walk kernel

Random Walk Kernel [4], abbreviated as RWK, compares number of common random walks in given graphs. The trick to calculate number of common random walk of length k is by taking direct product graphs (random walk on this graph is equivalent to simultaneous random walk in the two graphs). After direct product raise the adjacency matrix A , of resultant graph, by power of k to get number of common walk.

Direct product graph: Gartner proposed calculating random walk on direct product of graphs is equivalent to common random walk on graphs individually. Direct product graph is also known as tensor product or categorical product.

Definition 4. *The direct product of two graphs $G(V,E)$ and $G' (V',E')$ shall be denoted as $G_x = G \times G'$. The node and edge set of the direct product graph are respectively defined as,*

$$V_x = \{ (v_i, v'_r) : v_i \in V, v'_r \in V' \}$$

$$E_x = \{ ((v_i, v'_r), (v_j, v'_s)) : (v_i, v_j) \in E \wedge (v'_r, v'_s) \in E' \}$$

An example of direct product graph is shown in figure 3.1. Random walk kernel is defined as follows using the definition of direct product graph.

Definition 5. *Let G and G' be two graphs, let A_x denote the adjacency matrix of their product graph G_x , and V_x denote the node set of the product graph G_x . With a sequence of weights (also known as decaying factor) $\lambda = \lambda_1, \lambda_2, ..(\lambda_i \in \mathbb{R}, \lambda_i \geq 0$ for all $i \in \mathbb{N})$ the product graph kernel is defined as,*

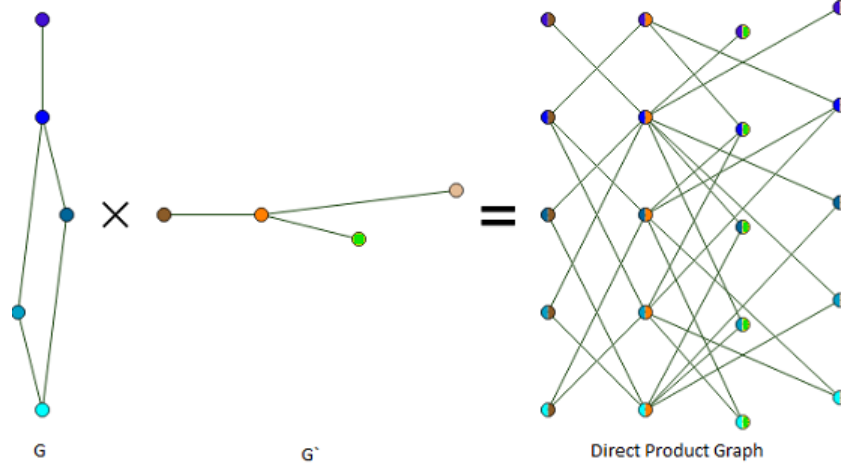


Figure 3.1: Pictorial Representation of Direct product graph

$$k_x(G, G') = \sum_{i,j=1}^{|V_x|} \left(\sum_{k=1}^{\infty} \lambda_k A_x^k \right)_{ij}$$

Drawbacks:

High Runtime: $O(n^6)$

If we set $\lambda_k = \lambda^k$ then, the kernel equation, $k_x(G, G') = \sum_{i,j=1}^{|V_x|} \left(\sum_{k=1}^{\infty} \lambda_k A_x^k \right)_{ij}$ will be reduced to,

$$k_x(G, G') = \sum_{i,j=1}^{|V_x|} [(I - \lambda A_x)^{-1}]_{ij}$$

As A_x is n^2 by n^2 matrix (assuming G and G' are of size n) the inverse would take $O(n^6)$.

Tottering

Some vertex might get visited over and over again or walk going back and forth, as walk allows repetition of nodes. This can result in high similarity between two graph because of one edge in common.

Halting

We have to choose decaying factor too small (in most of the cases) for convergence. By choosing small λ we are down-weighting the random walk of length more than 1 significantly, and by that I mean we are almost neglecting the random walk of length more than 1.

Fast computation of RWK: In [8] time complexity of RWK was reduced from $O(n^6)$ to $O(n^3)$. They used Sylvester equation to speed-up the process of finding the inverse of matrix, the term in Summation in above equation.

3.3 Shortest path kernel

Graph kernels based on walk suffer from tottering and halting. Unlike walks, path don't suffer from tottering as there is no repetition of nodes in path. So, [9] proposed a new kernel based on path. The challenge they faced while proposing a kernel on path was computing all paths is a NP-hard

problem. To overcome this difficulty they came up with a kernel based on shortest path distance (as it is unique for a graph and can be computed in polynomial time).

Definition 6. (Shortest path distance matrix) Let $G = (V, E)$ be a graph of size $|G| = n$. Let $d(v_i, v_j)$ be the length of the shortest path between v_i and v_j . The shortest path matrix D of G is then a $n \times n$ matrix defined as

$$D_{ij} = \left\{ \begin{array}{ll} d(v_i, v_j) & \text{if } v_i \text{ and } v_j \text{ are connected} \\ \infty & \text{otherwise} \end{array} \right\}$$

Definition 7. (Shortest path graph) Let $G = (V, E)$ be a graph, and let D be its shortest path distance matrix. Then the shortest-path graph S of G has the same set of nodes V as G , and its set of edges is defined via the adjacency matrix $A(S)$

$$A(S)_{ij} = \left\{ \begin{array}{ll} 1 & \text{if } D(v_i, v_j) < \infty \\ 0 & \text{otherwise} \end{array} \right\}$$

where $D(v_i, v_j)$ is edge label of edge (v_i, v_j) in S .

To compute SPK we have to transform graphs G and G' into shortest path graphs S and S' using Floyds algorithm. After transformation (we refer it as Floyd-transformation), we can now define SPK.

Definition 8. Let G and G' be two graphs that are Floyd-transformed into S and S' . We can then define our shortest-path graph kernel, $k(G, G') = k(S, S')$, on $S = (V, E)$ and $S' = (V', E')$ as

$$k_{\text{shortestpath}}(S, S') = \sum_{v_i, v_j \in G} \sum_{v'_k, v'_l \in G'} k_{\text{length}}(d(v_i, v_j), d(v'_k, v'_l))$$

where $k_{\text{length}}(., .)$ is any psd kernel (can be a delta kernel or linear kernel) to compare lengths of shortest path.

Linear kernel can be defined as the product of $d(v_i, v_j)$ and $d(v'_k, v'_l)$. Delta kernel can be a kernel whose value is Equal to 1 if and only if $d(v_i, v_j) = d(v'_k, v'_l)$ otherwise 0. It can be easily proved SPK is a psd kernel.

Advantages: SPK doesn't suffer from tottering, serves as better classification kernel (accuracy wise) and time complexity is $O(n^4)$.

Drawback: $O(n^4)$ is too slow for large Graphs.

3.4 Other kernels

- **Cyclic Pattern Kernel [5]** decompose a graph into cyclic patterns, then count the number of common cyclic patterns which occur in both graphs. It is computationally expensive as computing the cyclic pattern kernel on a general graph is NP-hard.
- **Edit Distance based kernel [10]** computes the minimum cost c incurred over all sequences S of edit operations that transform graph G into G' . It is not psd kernel and worst case time complexity can be exponential and hence not used in practice.

- **Optimal assignment kernel** [11] finds a best match, an optimal assignment between the substructures from G and G' . It is not a psd kernel, hence can't be used in machine learning problems.
- **Subtree-Pattern Kernel** [12] compares subtree-patterns. It suffers from tottering and can have exponential time complexity with respect height of the sub-tree patterns considered.
- **Weighted decomposition kernel** [13] decompose a graph into small subparts and reward similarity of these subparts with different weights. It works efficiently only for highly simplified representation of graph.

Note that the kernels discussed/listed till now work efficiently for smaller graph and are quite useful in cheminformatics, that is useful for molecule comparison. As comparing two large graphs has wide range of applications, like we can compare two groups in social networking, friend circle, etc. we need efficient graph kernel for large graphs. So, from now we discuss graph kernel based on large graphs.

3.5 A linear time graph kernel for large graphs

Linear time graph kernel [14] is also named as neighbourhood hash kernel (NHK) was proposed for labelled nodes, simple, unweighted and undirected graph. Later on it can be extended to other types of graphs. As name suggest the similarity score is calculated in $O(n)$ time. In NHK we map each label of a node to a D-bit array (same labels will have same representation in D-bit array) randomly, typically D is chosen as 16. To compute NHK we compute neighbourhood hash (NH) value for every node and then run simple algorithm to find similarity score.

Bit rotation: We need to rotate bits in specific manner to find NH value of a node and for that we define Bit rotation (ROT) operation. A ROT_o operation for $B = \{b_1, b_2, \dots, b_D\}$ shifts the last o bits to the left by o bits, and moves the first o bits to the right end, that is,

$$ROT_o(B) = \{b_{o+1}, b_{o+2}, \dots, b_D, b_1, b_2, \dots, b_o\}.$$

Definition 9. Suppose for a node v , $l(v)$ denote bit mapped label of v . If $x \oplus y$ represent XOR of bit array x and y then, NH of a node v is defined as,

$$NH(v) = ROT_1(l(v)) \oplus l(v_1^{adj}) \oplus l(v_2^{adj}) \oplus \dots \oplus l(v_d^{adj})$$

where v_i^{adj} is i^{th} neighbour of node v and d is number of neighbours of v

Figure 3.2 shows an example of how the neighbourhood hash works for a node given a set of connected nodes and labels.

But when we use NH technique there is a possibility of collision, depicted in figure 3.3 To reduce possibility of collision they came up with another technique, known as *Count-sensitive Neighbourhood Hash* (CS-NH). We get better accuracy in terms of similarity between two graphs when we use CS-NH instead of NH.

Definition 10. Suppose for a node v , $l(v)$ denote bit mapped label of v and $l'(v) = ROT_o(l(v) \oplus o)$, where o is number of times a label is repeated then CSNH for a node v is defined as,

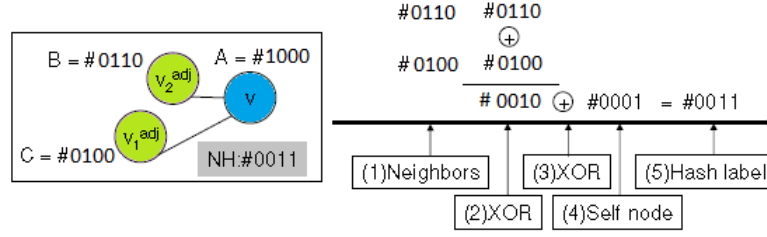


Figure 3.2: A example of neighbour hashing

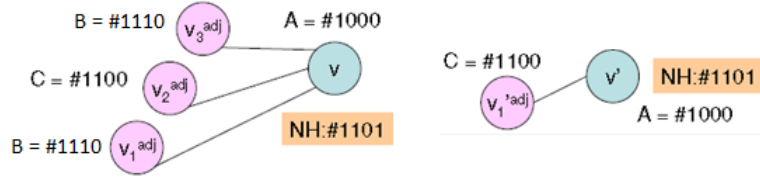


Figure 3.3: Collision of NH value for different graphs

$$CSNH(v) = ROT_1(l(v)) \oplus (l'(v_1^{adj}) \oplus l'(v_2^{adj}) \oplus \dots \oplus l'(v_{d'}^{adj}))$$

where v_i^{adj} is i^{th} neighbour of node v and d' is number of distinct neighbours of v .

After calculating NH or CSNH value for nodes algorithm compares all node pair labels and based on it it gives a similarity score. Similarity score is equal to number times a node label is matched with another node in other graph. If you are looking for similarity upto some depth, R then repeat above procedure for R times.

Time Complexity: $O(DR\bar{d}n)$

where D is a fixed constant (as described above), R is maximum order of neighbourhood to compare and \bar{d} is average number of neighbours in a graph.

Space Complexity: $O(n^2)$

Other types of Graphs

For *edged labelled graph* convert each edge (which is labelled) into a node and connect the nodes to which that label was connect by unlabelled edges. For *directed graph*, rotate 2-bit of neighbour for incoming edge and for outgoing edge do 3-bit rotation of neighbour to distinguish between them to find $NH(v)$ or $CSNH(v)$. For *unlabelled graph* apply label enrichment technique like Morgan index to make it a labelled graph. For *weighted graph* discretize the weights by binning which segments the continuous values into set of bins corresponding to different ranges of values, as long as the orders between the ranges can be ignored. This kernel is *scalable and efficient but doesn't accurately compare* large graphs.

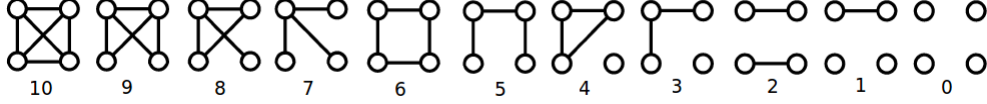


Figure 3.4: All size-4 modulo isomorphic graphlets

3.6 Graphlet kernel for large graph comparisons

Graphlet kernel for large graph [2] is based on counting size-k graphlets. Graphlets are small induced subgraphs of a large graph. The basic idea of graphlet kernel is to count number of matching graphlets of size-k in both graphs. No. of graphlets of size-k are exponential with respect to k. But number of modulo isomorphic graphlet of size-k will be far less than exponential.

Example: No. of possible graphlet of size-4 = 64 but no. of modulo isomorphic graphlets are only 11 (we call it as N_4), as seen in figure 3.4.

Frequency vector: For a given graph G, define a N_4 dimensional vector f_G whose i^{th} component corresponds to the frequency of occurrence of graphlet(i), shown in figure 3.4, in graph G.

Definition 11. Given two graphs G and G' , and their frequency vectors f_G and $f_{G'}$, we can compute the value of graphlet kernel as,

$$K_g(G, G') = f_G^T f_{G'}$$

In brief, it means to compute a graph kernel matrix on a set of graphs, we have to determine the frequency vector of each graph by enumerating its graphlets. To obtain the similarity score for two graphs, we take dot product of their frequency vectors.

To find f_G exactly using naive algorithm we have to consider ${}^n C_k$ graphlets of size-k for a graph of size n, that is $O(n^k)$. $O(n^k)$ is too expensive in practice, therefore they came up with two speed-up schemes, sampling and specifically designed scheme for bounded degree graph.

3.6.1 Sampling

We do sampling of graphlets from graph with hope that if sufficient samples are drawn from a graph then the approximate similarity score is close to actual similarity score. We sample some graphlets and with some confidence in a error bound we can calculate kernel value.

Theorem: Let D be a probability distribution on finite set $\mathcal{A} = \{1, 2, \dots, a\}$. Let $X = \{X_j\}_{j=1}^m$ be a iid multiset, with $X_j \sim D$. For a given $\epsilon > 0$ and $\delta > 0$,

$$m = \left\lceil \frac{2(a \log 2 + \log(\frac{1}{\delta}))}{\epsilon^2} \right\rceil$$

samples suffice to ensure that, $P \left\{ \left\| D - \hat{D}^m \right\|_1 \geq \epsilon \right\} \leq \delta$, where $\hat{D}^m(i) = \frac{1}{m} \sum_{j=1}^m \delta(X_j = i)$ for

$$i \in \mathcal{A} \text{ and } \delta(X_j = i) = \begin{cases} 1, & X_j = i \\ 0, & \text{otherwise} \end{cases}$$

Explanation: For comparing graphs, assume \mathcal{A} to be set of all size-k graphlets and are distributed according to an unknown distribution D. Let m be the number of graphlets randomly sampled from graph. Then from above formula, the value of m gives number of randomly selected samples needed

to ensure that \hat{D}^m is at most ϵ distance away from the actual distribution D with confidence of $1 - \delta$.

Example: Consider size-4 graphlets then we have, $a = 11$ (No. of distinct isomorphic graphs). And if we set,

- $\epsilon = 0.05$ and $\delta = 0.05$ then $m = 8,497$
- $\epsilon = 0.01$ and $\delta = 0.01$ then $m = 244,596$

3.6.2 Bounded degree graph

There is a large portion of graphs where complete counting of graphlets can be performed efficiently, that is the class of graphs with bounded degree d . In other words, a BDG (bounded degree graph) with bounded degree d will not have any node which have degree more than d . They proposed two algorithms for counting graphlets efficiently in low degree graph and those are as follows,

- Counting all connected graphlets
- Counting all graphlets for a fixed node v_1

Counting all connected graphlets

Here the assumption is that we are given graphs in adjacency list representation. They constructed a data structure that supports checking of existence of edge in $O(1)$.

Theorem: Let G be a BDG, and let d denote maximum degree. Then all connected graphlets of G with size $k \in \{3,4,5\}$ can be enumerated in $O(nd^{k-1})$ [2].

Counting all graphlets

Theorem: For a fixed node v_1 , we can compute the distribution of subgraphs of size 3 in time $O(d^2)$, d is the maximal degree of any node [2].

Advantages: Graphlet kernel is scalable and time Complexity is $O(nd^{k-1})$, where d is the maximum degree of a graph and k is the size of the graphlet considered for comparing graphs.

Drawbacks:

Maximum degree of a graph can be at most $n - 1$, implies worst case time complexity will be $O(n^{k-1})$, which is expensive.

Undesired similarity

This kernel considers non-connected components as well for calculating kernel value, which can unwantedly increase the similarity score between graphs.

The algorithms/techniques discussed till now are already developed. We came up with a new technique for comparing large graphs which is found to be more accurate than existing graph kernels in classifying graphs. We named it as *modified Graphlet Kernel* or *sub-graph kernel (SGK)*. In the next chapter we explain about the new technique which we proposed for measuring similarity of graphs.

Chapter 4

Modified graphlet kernel on large graphs

As existing kernels based on counting number of graphlets has few drawbacks, we proposed a new graph kernel based on counting of graphlets to overcome those drawbacks. After conducting experiment on various graphs our kernel is found to be more efficient, accuracy-wise, than existing graph kernels. Before understanding how modified graphlet graph kernel works, we need to know few aspects which will be used in further sections.

4.1 Graphlet vector

Graphlet vector is synonymous to frequency vector, but it is calculated for every node in a graph.

Definition 12. For a given k let N_k denote number of modulo isomorphic graph of size- k . Let G_1, G_2, \dots, G_{N_k} denote the subgraphs of size k modulo isomorphism. Let G be a graph. For vertex v of G let the corresponding graphlet vector $\bar{v} = \langle F_0, F_1, \dots, F_{N_k-1} \rangle$ is a frequency vector of the size N_k where i^{th} component F_i is the number of subgraphs in G containing v and isomorphic to G_i .

Graphlet vector for all the nodes in a graph are calculated in pre-processing step of modified graphlet kernel. We calculate graphlet vector for every node based on *counting all graphlets* mentioned in sub-section 3.6.2. For size-4 graphlet, $N_k = 11$ and in vector component F_i , where $0 \leq i \leq 10$ suffix i denote number edges in that graphlet. The vector component, F_i is pictorial shown in figure 4.1

Averaging effect: Existing graphlet kernel for large graphs [2] does averaging of graphlet vectors to compare similarity score. For two graphs $G(V, E)$ and $G'(V', E')$ graphlet kernel is expressed as

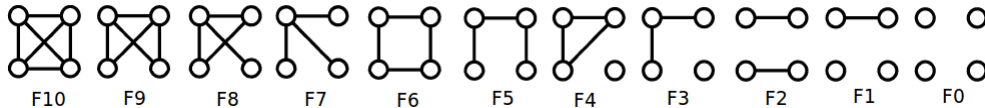


Figure 4.1: Graphlet vector dimensions

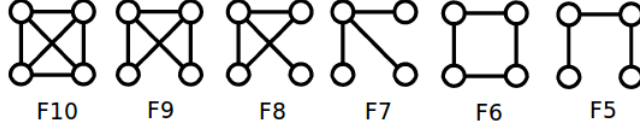


Figure 4.2: All connected size-4 graphlets

$$K_g(G, G') = f_G^T f_{G'} = c \sum_{\bar{v} \in V} \bar{v} \sum_{\bar{v}' \in V'} \bar{v}'$$

where, c is some constant, \bar{v} and \bar{v}' are graphlet vectors of nodes for graph G and G' respectively. As it does averaging of graphlet vector by simply aggregating them and then multiplying it with some constant, we named it as averaging effect. It is one of the drawbacks of [2]. Our graphlet kernel neither violate the topology of graph nor does simple averaging of frequency vector of nodes. Instead it does a weighted average of graphlet vectors.

Dimensions Considered: Before defining sub-graph kernel, we observed considering only connected components of graphlet vector are important for computing better similarity score. Considering non-connected components for similarity can distort/disturb similarity score, as they can be huge in number for large graphs. Pictorially, we are considering only graphlets shown in figure 4.2 for calculating kernel value of two graphs.

By considering only connected components we overcome one of the drawback of graphlet kernel [2]. After calculating all the graphlet vectors of the graphs, which are to be compared, we have to deal only with comparison of *set of vectors*. As those set of vectors implicitly capture topology of graph, we defined modified graphlet kernel on them.

4.2 Modified graphlet kernel

In modified graphlet kernel we divide set of vectors into set of groups or clusters. We can divide set of vectors (a graph) into groups based on any criteria. The comparison of graphs using modified graphlet kernel is independent of the method used for dividing a graph into sub-graph. We tried dividing graph into sub-graphs based on following approaches,

- BFS upto depth-d
- Clustering

We will discuss each of these technique in detail in following sections. For simplicity we will assume the graphs which are to be compared are of same size, that is they have equal number of nodes, say n . The methods which we proposed can easily be extended for graphs of different sizes. We also assume that we are given with two graphs, $G(V, E)$ and $G'(V', E')$ in adjacency matrix format and graphlet vectors of both the graphs G and G' , that is n graphlet vectors for each graph (as both graphs has n nodes).

4.2.1 BFS upto depth-d

As we are given with 'n' graphlet vector and adjacency matrix of graphs G and G' , we can easily traverse graph using standard BFS from a node. We group nodes using this approach as follows.

Start from a node and traverse graph using bfs upto depth d. Then create a group A_i ($1 \leq i \leq n$), which consists of nodes captured in traversal. Continue this process for all the nodes in a graph. After grouping process is completed, we will have ‘n’ groups/sub-graphs (A_1, A_2, \dots, A_n) for a graph G and n groups/sub-graphs (B_1, B_2, \dots, B_n) for graph G' .

The advantage of this method is that the node with high degree will appear in many groups/sub-graphs. Thus that node will contribute to similarity score those many times. Another advantage of this technique is that, it will help us capturing local similarity between two graphs efficiently.

Definition 13. (Group Vector) Let $\bar{v}_1, \bar{v}_2, \dots, \bar{v}_d$ be the graphlet vectors in group \mathcal{A} then group vector for \mathcal{A} denoted as \bar{A} , is defined as

$$\bar{A} = \sum_{i=1}^d \bar{v}_i$$

In simple words sum of all the graphlet vectors in a group is known as group vector.

Definition 14. Let A_1, A_2, \dots, A_n and B_1, B_2, \dots, B_n denote groups of graph G and G' respectively. Let $\bar{A}_1, \bar{A}_2, \dots, \bar{A}_n$ and $\bar{B}_1, \bar{B}_2, \dots, \bar{B}_n$ denote group vectors of graph G and G' respectively. Sub-graph kernel based on BFS upto depth-d is defined as,

$$K_{sgk}(G, G') = D_G^T D_{G'}$$

where, $D_G^T = \sum_{i=1}^n \bar{A}_i$ and $D_{G'} = \sum_{i=1}^n \bar{B}_i$.

In other words graphlet kernel based on BFS upto depth-d is pair-wise dot product of group vectors from G and G' .

4.2.2 Clustering

As we are given with ‘n’ graphlet vector, we can easily cluster nodes based on euclidean distance of graphlet vector by using any of the standard clustering algorithm, like k-means or k-median.

Definition 15. After clustering assume we have ‘k’ groups (A_1, A_2, \dots, A_k) in graph G and ‘l’ groups (B_1, B_2, \dots, B_l) in graph G' . Let $\bar{A}_1, \bar{A}_2, \dots, \bar{A}_k$ and $\bar{B}_1, \bar{B}_2, \dots, \bar{B}_k$ denote group vectors of graph G and G' respectively, n_i denote number of nodes in i^{th} group. Sub-graph kernel based on clustering is defined as

$$K_{sgk}(G, G') = D_G^T D_{G'}$$

where, $D_G^T = \sum_{i=1}^k \frac{n_i}{n} \bar{A}_i$ and $D_{G'} = \sum_{j=1}^l \frac{n_j}{n} \bar{B}_j$

Above kernel definition makes sure that we are not doing simple averaging of graphlet vector. Instead we are doing weighted average of graphlet vectors.

Clustering technique is faster than BFS upto depth-d, as we over count many nodes while grouping them in BFS. As you don’t need to store graph adjacency matrix for clustering nodes, you save memory in clustering technique. In general BFS technique is more accurate in comparing graphs. In clustering technique for number of clusters $k = \lfloor \log n \rfloor$ the results are found to be much better.

Lemma: Sub-Graph kernel is a psd Kernel.

Proof: As positive semi-definite kernel indicate dot product of vectors in high dimensional space and as our kernel function is defined as dot product of two vectors, sub-graph kernel is a positive semi-definite kernel. The class of p.s.d kernels is closed under non-negative linear combinations and point wise limits [15]. Positive definiteness of this kernel follows from this as well.

Above discussion can easily be extended for other size graphlets, that is for size-3 and size-5 graphlets.

Chapter 5

Experimental results

In this chapter we evaluate performance of our kernel in terms of scalability and accuracy of classification or comparing graphs.

Dataset: We performed experiments on $G(n,p)$ model [16], a graph $G(V,E)$ is constructed by connecting nodes randomly, where $|V| = n$. Each edge is included in the graph with probability p independent from every other edge. We tested our kernel using different variants of graphs and graphlet size, that is for graphlet of sizes 3 and 4. In this chapter we show results obtained when we performed detailed experiments on 1000 node graphs and size-4 graphlets.

Experimental Setup: We generated a 1000 node graph, G_1 , using $G(n,p)$ model [16], and then we removed some edges randomly from that graph, and named the resultant as graph G_2 , and then we removed few more edges from G_2 in similar fashion to get G_3 and continued similar process upto G_{11} . As we generated graphs such that G_1 is more similar to G_2 than G_3 or G_4 or any other graph, the similarity score should be in decreasing order. Based on these graphs, we generated gram matrix after running modified graphlet kernel on these 11 graphs and we observed that the similarity score in rows are in decreasing order, after diagonal entry, as expected. Note that we took depth, $d=2$ for BFS upto depth- d and number of clusters $k=9$ in our experiments.

Results: In following figures, *X-axis* denotes graphs (G_1, G_2, \dots, G_{11}) numbered as 1, 2, 3, ..., 11 and *Y-axis* denote kernel value or similarity score between graph G_1 and other graphs (that is values of first row in gram matrix).

When we performed sub-graph kernel experiments on 1000 node graph (generated using $G(n,p)$ model) taking only connected graphlet of size-4 into consideration with edge probability as 0.005 and 0.1 we get the graph pattern as shown in figure 5.1.

In figure 5.2, we show the values of first row in gram matrix generated using graphlet kernel [2], when we took only connected graphlets. Though the actual paper doesn't talk about considering only connected components, we plotted graph to show when we take only connected components the results are better than considering all components.

In figure 5.3, we show comparison of our kernel (sub-graph kernel) with graphlet kernel of [2]. We took graphlet vector of size 11, that is considering all the components of graphlet vector. We took edge probability for generating G_1 as 0.005, number clusters as 9 for clustering and depth $d=2$ in

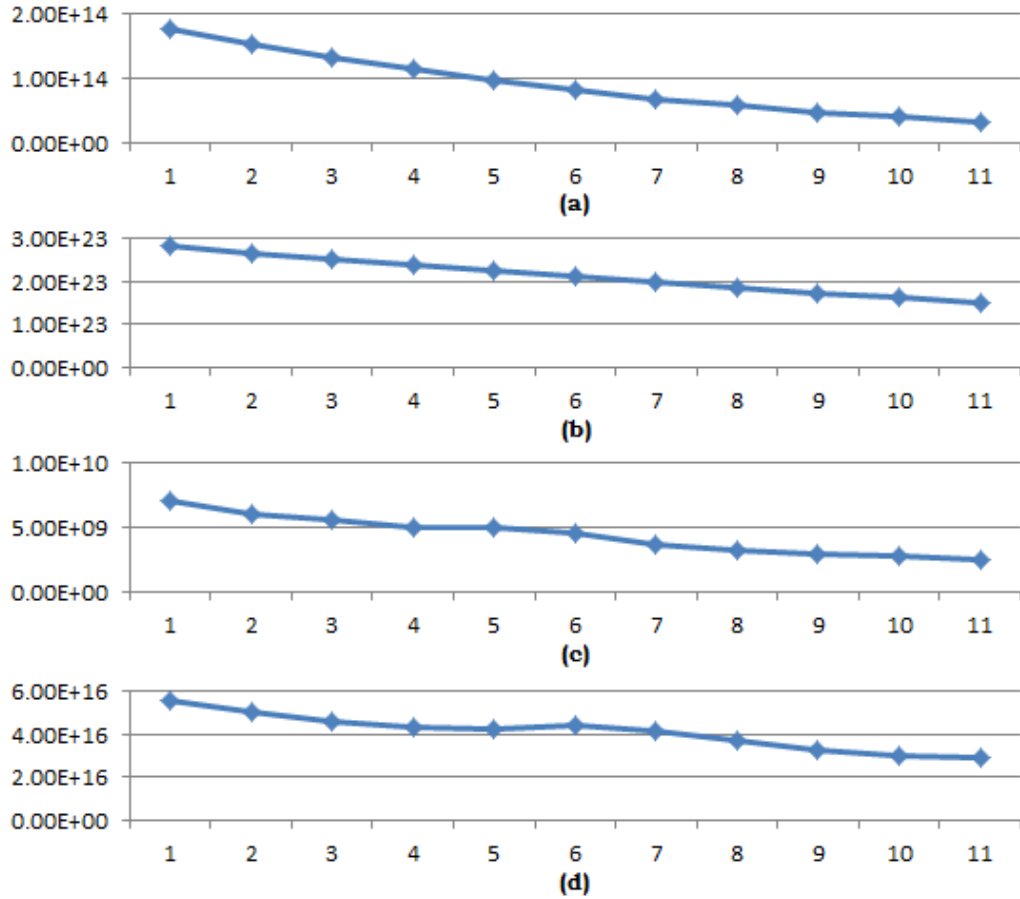


Figure 5.1: In (a),(b) we used BFS upto depth-d technique and in (c),(d) we used clustering technique to calculate kernel value. In (a), (c) the edge probability for generating G1 is 0.005 and in (b), (d) the edge probability for generating G1 is 0.1

bfs upto depth-d.

Clearly from comparison shown in figure 5.3, we can state that for $G(n,p)$ model our kernel outperform existing graph kernel in terms of accuracy in measuring similarity of large graphs.

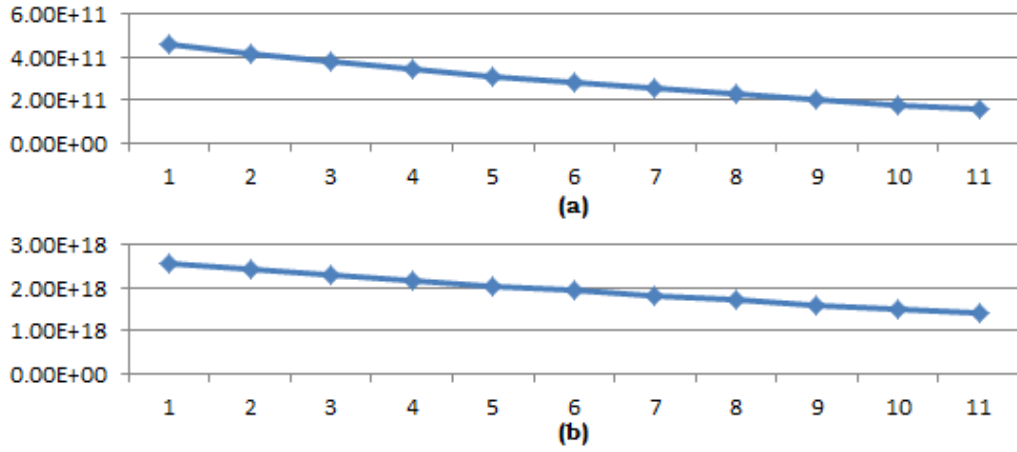


Figure 5.2: In (a) edge probability is 0.005 and in (b) edge probability is 0.1 for generating graph G1

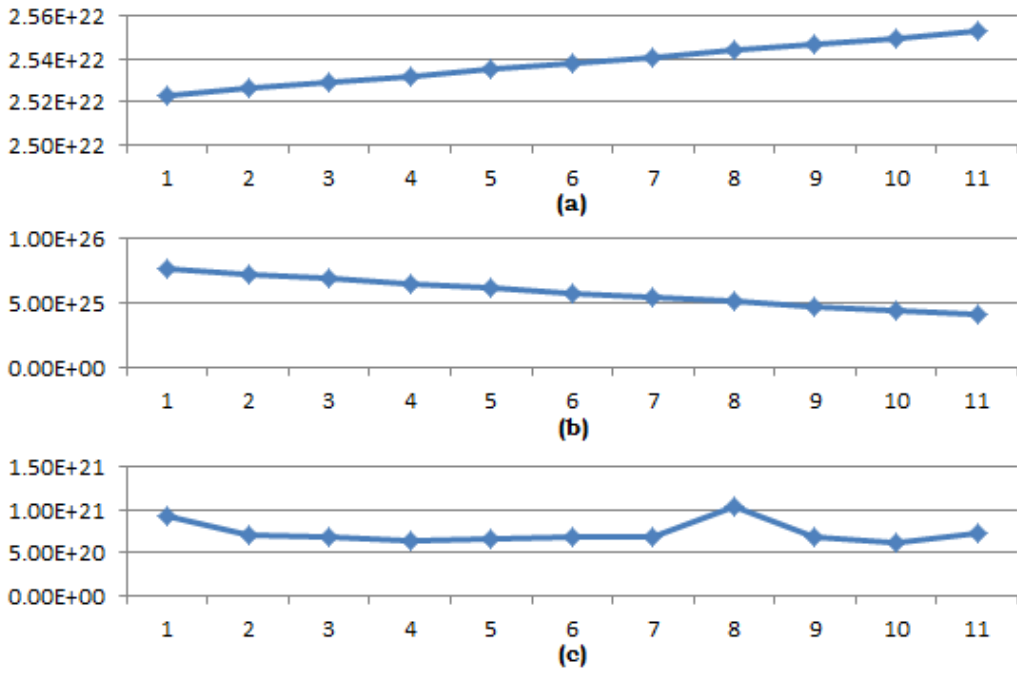


Figure 5.3: Graph kernel Comparison on size-11 graphlet vector. (a) shows kernel values based on [2], (b) shows kernel values based on modified graphlet kernel when we use bfs upto depth-d and (c) shows kernel values based on modified graphlet kernel when we use clustering technique.

Chapter 6

Discussion

We proposed a graph kernel for large graphs which is more accurate than existing graph kernels on randomly generated graphs, generated using $G(n,p)$ model. We have overcome the problems like averaging effect by taking weighted average and problem of counting non-connected graphlets by considering only connected components in our graph kernel. Our graph kernel is more accurate in classification of large graphs, can capture local similarity, scalable and doesn't disturb topology of graph.

Though our graph kernel is more accurate for large graphs, the central challenge for future is to *speed-up pre-processing step* by calculating graphlet vector for all nodes approximately. Another challenge is to check for the efficiency of the graph kernel if the graphlet of size more than 5 are considered for comparing graphs. We have to try for accuracy and efficiency of our graph kernel on real world classification problem on real data. As the modified graphlet kernel is based on *set of vectors* we can use other kernels, like [17] or [18] for measuring similarity of graph. Then we can compare the results of these kernel with sub-graph kernel to check accuracy of our kernel.

References

- [1] Mining and Learning with Graphs 2007.
- [2] N. Shervashidze, S. V. N. Vishwanathan, T. H. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient Graphlet Kernels for Large Graph Comparison. In D. van Dyk and M. Welling, eds., Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS), volume 5 of *JMLR: Workshop and Conference Proceedings*. CSAIL, Clearwater Beach, Florida, USA, 2009 488–495.
- [3] D. Haussler. Convolution Kernels on Discrete Structures. Technical Report 1999.
- [4] T. Gartner, P. A. Flach, and S. Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. In B. Scholkopf and M. K. Warmuth, eds., COLT, volume 2777 of *Lecture Notes in Computer Science*. Springer, 2003 129–143.
- [5] T. Horvath, T. Gartner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel, eds., KDD. ACM, 2004 158–167.
- [6] B. Scholkopf and A. J. Smola. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge, MA, USA, 2001.
- [7] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society* .
- [8] S. V. N. Vishwanathan, N. N. Schraudolph, R. I. Kondor, and K. M. Borgwardt. Graph Kernels. *Journal of Machine Learning Research* 11, (2010) 1201–1242.
- [9] K. M. Borgwardt and H.-P. Kriegel. Shortest-Path Kernels on Graphs. In ICDM. IEEE Computer Society, 2005 74–81.
- [10] M. Neuhaus and H. Bunke. Edit Distance Based Kernel Functions for Attributed Graph Matching. In L. Brun and M. Vento, eds., GBRPR, Lecture Notes in Computer Science. Springer, 2005 352–361.
- [11] H. Frohlich, J. K. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In In Proceedings of the 22nd international conference on Machine learning. ACM Press, 2005 225–232.
- [12] J. Ramon and T. Gartner. Expressivity versus efficiency of graph kernels. Technical Report 2003.

- [13] S. Menchetti, F. Costa, and P. Frasconi. Weighted decomposition kernels. In L. D. Raedt and S. Wrobel, eds., ICML, ACM International Conference Proceeding Series. ACM, 2005 585–592.
- [14] S. Hido and H. Kashima. A Linear-Time Graph Kernel. In Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, ICDM '09. IEEE Computer Society, Washington, DC, USA, 2009 179–188.
- [15] C. Berg, J. P. R. Christensen, and P. Ressel. Harmonic Analysis on Semigroups. Springer, Berlin, 1984.
- [16] B. Bollobas. Random Graphs. Cambridge University Press, 2001.
- [17] F. Desobry, M. Davy, and W. J. Fitzgerald. A class of kernels for sets of vectors. In In Proceedings of the 13th European Symposium on Artificial Neural Networks. 2005 .
- [18] R. Kondor and T. Jebara. A Kernel between Sets of Vectors. In In International Conference on Machine Learning (ICML. 2003 .
- [19] K. M. Borgwardt. Graph kernels. Ph.D. thesis, Ludwig Maximilians University Munich 2007.
- [20] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society* 68.
- [21] K. M. Borgwardt, H. peter Kriegel, S. V. N. Vishwanathan, Nicol, and N. Schraudolph. Graph kernels for disease outcome prediction from protein-protein interaction networks. In In Proc. of Pacific Symposium on Biocomputing (PSB 2007), Maui. World Scientific, 2007 .
- [22] X. Wang, A. Smalter, J. Huan, and G. H. Lushington. G-hash: towards fast kernel-based similarity search in large graph databases. In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, EDBT '09. ACM, New York, NY, USA, 2009 472–480.
- [23] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM '01. IEEE Computer Society, Washington, DC, USA, 2001 313–320.
- [24] K. M. Borgwardt and H. peter Kriegel. An Efficient Sampling Scheme For Comparison of Large Graphs.
- [25] N. Pržulj, D. G. Corneil, and I. Jurisica. Efficient estimation of graphlet frequency distributions in protein–protein interaction networks. *Bioinformatics* 22, (2006) 974–980.
- [26] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In Proceedings of the 20th International Conference on Machine Learning. AAAI Press, 2003 321–328.
- [27] N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. In NIPS. 2009 1660–1668.
- [28] S. V. N. Vishwanathan and A. Smola. Fast Kernels for String and Tree Matching 2004.