CHALMERS

Fast shortest-path kernel computations using approximate methods

Master of Science Thesis in the Programme Computer Science: Algorithms, Languages and Logic

JONATAN KILHAMN

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, March 2015

Fast shortest-path kernel computations using approximate methods
Jonatan Kilhamn

Supervisor: Fredrik Johansson

Examiner: Devdatt Dubhashi

Department of Computer Science and Engineering
LAB Research Group
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 31 772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, March 2015

Abstract

The shortest-path kernel is frequently seen in the context of graph classification, which shows up in various subjects, for example bioinformatics. However, it is not efficient enough to be applicable in practice if the graphs are too large. The purpose of this thesis is to explore the possibilities of computing the shortest-path kernel approximately, taking shorter time at the cost of a limited error.

This thesis proves a theoretical error bound for a class of kernel function approximations, applicable to the shortest-path kernel but further generaliseable as well. We also present two specific approximations of the shortest-path kernel.

Firstly, we define an approximate kernel based on the idea of sampling node pairs in a graph to approximate its shortest-path length distribution. Secondly, we define a kernel computing approximate shortest-path lengths in a graph using its graph Voronoi dual. We provide algorithms to compute both of these, and prove that their runtime complexities are better than the shortest-path kernel they approximate. Finally, we evaluate these kernel approximations empirically, comparing them to the full shortest-path kernel as well as other reference kernels.

Contents

List of Figures

Chapter 1

Introduction

A *graph kernel* is a function used to measure similarity between graphs. The problem of defining such a similarity measure is a complex one, and numerous different graph kernels based on different features and substructures of graphs have been proposed (Shervashidze et al. 2009; Borgwardt and Kriegel 2005; Borgwardt and Shervashidze 2009). Like with other types of kernel functions, a possible end goal of graph kernel research is to use kernels to classify graphs as being more similar to one group of graphs than to another.

The subject of graph kernels thus lies in the intersection between graph theory and machine learning. Since data from many real and potential applications, including bioinformatics (Borgwardt et al. 2006) and social networks (Bui and Honavar 2014), can be represented by graphs, useful kernel functions for graph classification are of growing interest to both researchers and industry. To be useful, a graph kernel must both capture relevant information about the graphs and be efficiently computable (Shervashidze et al. 2011). In recent years several papers (Shervashidze et al. 2009; Borgwardt and Kriegel 2005; Borgwardt and Shervashidze 2009) have explored different graph kernels, and different methods for computing them efficiently. One such method is the practice of sampling substructures within graphs, using statistics of the samples rather than the entire graph (Shervashidze et al. 2009).

1.1 Purpose

The purpose of this thesis is to advance the knowledge of viable graph kernels. There is both a theoretical and a practical side to this task. On the theoretical side, one can fulfil the purpose by researching new kernels, variations on existing kernels or computational methods to use with existing kernels. On the practical side, one can make progress by testing kernels on different data sets, comparing them to each others. This provides useful data on which kernels are accurate—measured by performance on classification tasks—and efficiently computable on which types of data sets.

1.2 Objectives

This thesis seeks to further the above purpose by exploring the possibilities of computing the widely used *shortest-path kernel* faster than it has previously been. In order to achieve this, we apply two approximation approaches: *sampling* and *approximate shortest-path queries*.

We present herein three new implementations of approximate shortest-path kernels, along with both theoretical and practical results and analyses. These include theoretical bounds on time complexity for all three implementations, bounds on approximation error for two of them, and empirical results from computational experiments for all of them. The theoretical bounds indicate whether the kernel computations will scale better to large graphs than the standard shortest-path kernel. The experimental data will show how the new approximate variants actually perform compared to both the standard shortest-path kernel and two other common graph kernels, with regards to classification accuracy and computation time.

1.3 Limitations

This project is limited from the outset by the focus on one family of graph kernels; the shortest-path kernel. For the comparative computational experiment part, we comnpare our contributions to the performance of two other methods as well. However, we labour to lift those methods wholesale from their proponents, making no effort to modify or improve them.

Another limitation to the applicability of the results is the choice of graph data used for testing. We have access to a few different sources of large graphs, as well as the option of generating artificial graphs of any size. However, every dataset has its drawbacks. For example, datasets from real-world applications might consist of graphs of different sizes, muddling runtime comparisons; whereas randomly generated graphs of a fixed size most likely will not share all relevant characteristics of real-world data. Of the five datasets used, two are fabricated and three have real-world origins; for more details, see section 7.1.

Moreover, shortest-path kernels can generally be defined for many different types of graphs, including graphs with node labels, edge labels, neither, or both (see chapter 2). On four of said five datasets we limit ourselves to unlabelled graphs, and the fifth includes integer-valued edge labels but no node labels. All other combinations of labels are possible future research directions, but not pursued herein.

1.4 Method

We apply two different approaches to approximating the shortest-path kernel and decrease computation times. The first is to sample a number of node pairs from each graph, and computing the kernel value based on the shortest paths between those pairs only (chapter 5). The second approach, building on the sampling results, is to also compute the shortest-path lengths approximately. This is accomplished through the use of the Voronoi graph dual (chapter 6).

Each of these approaches incorporates some theoretical groundwork in graph theory and algorithm design, a concrete implementation in MATLAB and C, and finally comparative testing. The most important feature of the theoretical part is the various time complexity and probabilistic error bound proofs. The two are interwoven, as seen in the graphlet kernel research of Borgwardt et al.: there, the bound on the approximation error affected the number of samples needed, which in turn determined the time complexity (Shervashidze et al. 2009). The same situation arises in this thesis.

Implementations

Our implementations of the speedup schemes are written in MATLAB, with particularly loop-heavy structures being placed in sub-routines written in C. Because some parts are written in a different language, there is even more reason than normally to doubt that theoretical time complexities would translate directly to actual relative runtimes. Thus, our chief conclusions are instead based on comparisons of actual operations counts, and how the runtimes scale with larger input.

Parts of the code used are not written entirely by us. Specifically, the code available through Shervashidze and Borgwardt (2012a) has been invaluable. It includes several MATLAB functions, used to obtain the results in chapter 8. Firstly, the Weisfeiler-Lehman kernel computation function written by Nino Shervashidze, and the graphlet kernel ditto written by Karsten Borgwardt, are used without modification. Further, the function to compute the shortest-path kernel, also by Shervashidze, is modified and extended to compute the approximate kernels defined in chapter 5 and chapter 6. Finally, Shervashidze's entire SVM testing evironment is used as-is.

Experiments

We test our implementations on five different graph datasets. The test routine consists of computing the kernel value for each pair of graphs within the set, and then using those values to classify the graphs into two categories using an SVM. In addition to the classification accuracy, we also measure the kernel value error as compared to the "baseline" shortest-path kernel, runtimes for different parts of the computations, and in some cases operation counts of key algorithms.

1.5 Overview

Chapter 2 provides a primer on graph theory, to refresh the reader's memory of the subject and provide definitions for future reference.

Chapter 3 fulfills a similar function for the subject of the SVM method in machine learning, and the notion of kernel functions closely tied to it.

Chapter 4 provides some background on the shortest-path kernel and its definition, as well as outlining its computational complexity and the basic approach to finding an approximation.

Chapter 5 and Chapter 6 develops two such approximations, proving an error bound as well as all relevant runtime complexities.

Chapter 7 describes the experimental setup used to evaluate the approximations presented in earlier chapters; outlining both the specifics of which algorithms, parameters etc. were used as well as desciribing the graph datasets used for testing.

Chapter 8 presents and visualises the most important and interesting results of the empirical evaluation in some detail; however, the full set of results is available in section 9.3.

Finally, Chapter 9 includes our conclusions and discussion arising from the results presented in the previous chapter.

Chapter 2

Graph theory

This chapter provides a primer on graph theory: the basic definitions and concepts, as well as some background on shortest-path algorithms including Dijkstra's algorithm and the Floyd-Warshall algorithm.

2.1 Definitions

A *graph* G consists of a set of *nodes* V and a set of *edges* E. Each edge $e = (u, v)$ is an ordered pair of nodes, and represents a connection between nodes $u, v \in V$. The total number of nodes $|V|$ is denoted n, and the number of edges $|E|$ is denoted m. In any graph with no duplicate edges, we have $m \leq n^2$.

If the number of edges is close to the maximum, i.e. $m \in O(n^2)$,the graph is called *dense*. Conversely, if $m \in O(n)$, it is called *sparse*.

A graph can be *labelled* in several ways, or completely unlabelled. Real-valued edge labels are common, and each edge label (*weight*) w_{uv} can thus represent a distance or cost of travelling between the connected nodes u and v. In this context, a graph with no edge labels can represent a network with only distance-1 edges.

If every edge $(v, w) \in E$ has a corresponding reverse edge $(w, v) \in E$ (with the same label), the graph is *undirected*; otherwise it is *directed*.

A *walk* W is a sequence of nodes $v_1, v_2, ..., v_k$ where $(v_{i-1}, v_i) \in E$ for $1 < i \leq k$. A *path* is a walk with no repeated nodes.

A walk or path as per above has a *length* of $k - 1$, corresponding to the number of edges it traverses. Alternatively, in graphs with weighted edges, the length is the sum of the weights of those edges. This thesis makes extensive use of the concept of a *shortest path* from v_s to v_t, which is the (possibly non-unique) minimum-length path $W_{\mathrm{SP}} = v_s, ..., v_t$.

The *diameter* $\mathrm{diam}(G)$ of a connected graph is the longest shortest-path distance between any pair of nodes.

2.2 Representations

We can represent the data contained in the object $G = (V, E)$ in several ways. One way is through an *adjacency matrix*, denoted A, which is defined by its elements

$$A_{ij} = \begin{cases} w_{v_i v_j}, \text{ if } (v_i, v_j) \in E \\ 0, \text{ otherwise} \end{cases} \tag{2.1}$$

An adjacency matrix encodes the entire graph as long as it has no node labels.

An alternate representation is the *adjacency list* L. The i:th element of L is the set of nodes adjacent to (i.e. connected to) v_i. This corresponds to the notion of a *neighbourhood* $N(v_i) = \{v_j : (v_i, v_j) \in E\}$. In a graph with edge weights, the set L_i instead contains pairs (v_j, c) such that $(v_i, v_j) \in E$ and $c = w_{v_i v_j}$.

These two representations can encode the same information, but are favoured in different circumstances. The adjacency matrix provides easy lookup for any node pair whether connected or not, while the adjacency list is suitable for searching through all edges. On sparse graphs, the $O(m)$ space complexity of the adjacency list may also be favourable over the adjacency matrix, which requires space in $O(n^2)$ regardless of sparseness.

2.3 Shortest-path algorithms

There are several well-known problems related to finding shortest paths in graphs, and algorithms which solve them. Three problems relevant to this thesis are the *single-pair*, *single-source* and *all-pairs* shortest path problems. The solutions to these problems are, in a fashion, supersets to each other: the shortest path between a pair of nodes is contained in the set of shortest paths from a single source node to all other nodes, which is in turn contained in the set of shortest paths between all possible node pairs in the graph.

Single-pair

For the single-pair problem, the A^* algorithm (Hart, Nilsson, and Raphael 1968) has been championed as the superior method (Zeng and Church 2009). However, it relies on the existence of a distance heuristic; a method of approximating the total distance from any node to the target. In e.g. graphs where nodes represent spatial locations this poses no problem, but for some types of graphs, including ones where we shall later see the need to solve the single-pair shortest path problem, we have found no such heuristic.

Other than the A^* approach, we have not been able to find any single-pair shortest path algorithm that does not also solve the single-source shortest path problem. Intuitively, this makes sense: if we do not have domain-specific knowledge about the graph (a distance heuristic), we must be prepared to, in the worst case, search all nodes before we reach the goal node. When that happens, we have found the shortest paths from the source node to all other nodes, i.e. we have solved the single-source problem while looking for the single-pair solution.

Single-source

Dijkstra's (1959) algorithm is the definitive single-source shortest path algorithm (Cormen et al. 2001). It has a runtime in $O(n^2)$ in its original formulation (ibid.), $O(m+n\log n)$ in a commonly cited implementation using a Fibonacci heap (Fredman and Tarjan 1984), and as low as $O(m+n\log\log n)$ in certain specialised settings (Thorup 2003).

A simpler version (in terms of implementation) which falls between the original and the Fibonacci heap formulation is one based on a binary heap. Mehlhorn and Sanders (2008) proves that binary-heap Dijkstra's computes single-source shortest-path distances in expected $O(m + n\log\frac{m}{n}\log n)$ time. This is "expected" in the sense that the proof defines a way to randomise edge labels for a given graph, and finds the expected runtime over graphs randomised this way; for details see Mehlhorn and Sanders (ibid.). It is worth noting that for the specific case of unweighted graphs, the randomisation process has only one possible value, and thus the expected result holds generally.

In subsequent chapters, when we speak of using Dijkstra's algorithm in general, we will assume that it is possible to use the Fibonacci heap formulation and attain a runtime in $O(m + n\log n)$. In practice, we use the binary heap formulation for our implementations, giving a slightly worse runtime. We motivate this choice in section 5.5.

All-pairs

For the third problem, the all-pairs shortest path problem, there is the specialised Floyd-Warshall algorithm (after Floyd (1962) and Warshall (1962)). It uses a dynamic programming approach to find all shortest paths in a graph in $O(n^3)$ runtime. Asymptotically, this is worse than the algorithm constructed by performing Dijkstra's algorithm for each node, which runs in $O(nm + n^2\log n)$. However, there are two points in favour of the Floyd-Warshall algorithm worth noting.

The first one is that while the asymptotic runtime is worse (for sparse graphs), the hidden constant factors may differ. An immediate difference is that, in an undirected graph, the Floyd-Warshall algorithm simply computes all shortest paths – while Dijkstra's algorithm run for all n choices of source node actually computes each path twice.

Secondly, Floyd-Warshall is simpler and seems to us easier to implement. A specific instance of this, responsible for its use in this thesis, is that it is used to compute the shortest-path kernel in the code provided by Shervashidze and Borgwardt (2012a).

Chapter 3

SVMs and kernels

This section provides an introduction to the machine learning problem of graph classification by a *support vector machine* (SVM), and the related notion of a kernel function. It will not go into more detail regarding either kernels or SVMs than is necessary for the understanding of the rest of this thesis. The results in this section, unless otherwise noted, come from Schölkopf and Smola (2001).

3.1 Graph classification

To motivate the introduction of the SVM, consider the following problem: given a set of *training data*

$$x_1, ..., x_M \in \mathcal{X}$$

with corresponding *labels*

$$y_1, ..., y_M \in \{\pm 1\}$$

we wish to construct a *decision function*

$$f : \mathcal{X} \to \{\pm 1\}$$

which can accuractely predict the label y_{M+1} of a new data point x_{M+1}. The setup assumes that the training data actually tells us something about the new data; i.e. there is a common distribution of (x_i, y_i) pairs which we wish to, in some sense, learn.

The set $\mathcal{X}$ can be virtually anything. In our case, it will be (some subset of) graphs. As an example of a graph classification task, the dataset from Dobson and Doig (2003) consists of graphs representing molecules, along with labels in $\{\pm 1\}$ representing whether each molecule is an enzyme or not.

3.2 The support vector machine

Moving back a little, consider the more constrained problem of classifying data $\mathbf{x} \in \mathcal{H}$, where $\mathcal{H}$ is an inner product space, i.e. we can define an inner product $\langle \mathbf{x}, \mathbf{x}' \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$. In this setting, one option is to search for the *optimal hyperplane* dividing the space $\mathcal{H}$ into one subspace for each of the two labels $+1$ and -1. A hyperplane in $\mathcal{H}$ is defined by a choice of $\mathbf{w} \in \mathcal{H}$ and $b \in \mathbb{R}$, and can be used to construct a decision function

$$f(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

As an example, fig. 3.1 shows the case where $\mathcal{H} = \mathbb{R}^2$. Here, a hyperplane becomes a line, and the decision function assigns different labels to points falling on different sides of the line.

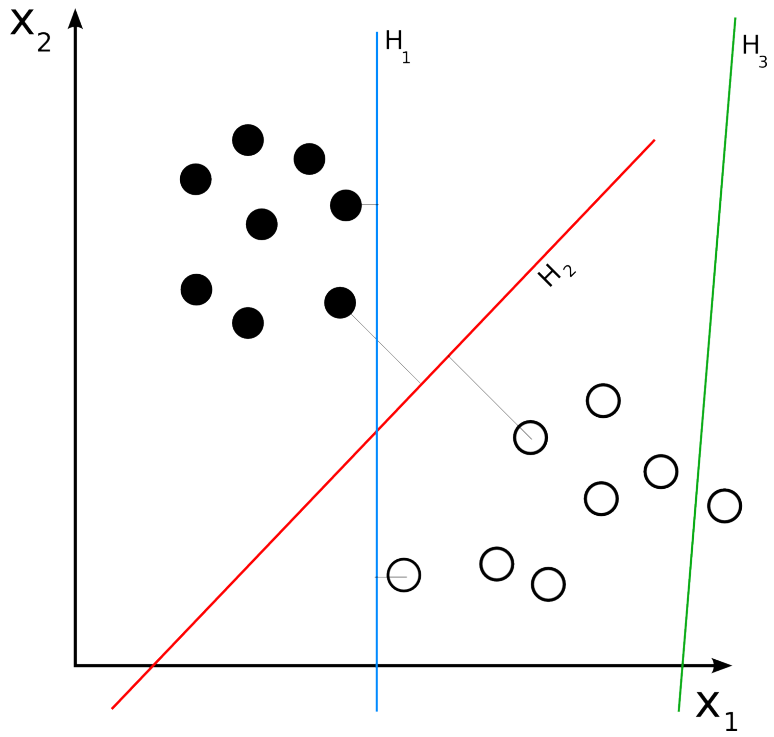

Figure 3.1: A set of training data in $\mathbb{R}^2$, and three hyperplanes (lines), two of which separate the data correctly.
Source: Zack Weinberg, via Wikimedia Commons.

The optimal hyperplane is the one where the *margin*, i.e. the distance between the hyperplane and the closest training point—marked in fig. 3.1 for hyperplanes H_1 and H_2 by faint lines—is maximised. It can be found by solving the following minimisation problem:

$$\min_{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}} \tau(\mathbf{w}) = \frac{1}{2} ||\mathbf{w}||^2 \tag{3.1}$$

$$\text{subject to } y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, i = 1, ..., M. \tag{3.2}$$

Through the use of the Lagrangian—the full derivation is presented by Schölkopf and Smola (2001)—the problem can be converted to its dual:

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{3.3}$$

$$\text{subject to } \alpha_i \geq 0, i = 1, ..., M \text{ and} \tag{3.4}$$

$$\sum_{i=1}^{M} \alpha_i y_i = 0 \tag{3.5}$$

where α is a vector of elements α_i.

This yields a decision function of

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{M} y_i \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right) \tag{3.6}$$

where b can be computed from α and the training data (ibid.).

The dual problem is efficiently solvable, and eq. (3.6) provides a good start for tackling classification problems. There are two large improvements worth mentioning, however.

Firstly, the separating hyperplane may not exist – imagine for example an anomalous outlier in the training data which is labeled "+1" yet lies in the middle of a cluster of "−1"-labeled points. This can be solved by introducing *slack variables* $\chi_i, i = 1, ..., M$ in the primal problem's constraints, changing eq. (3.2) into

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \chi_i, i = 1, ..., M \tag{3.7}$$

With this change, it is possible to formulate a *soft margin* classifier whose decision function has the exact same form as eq. (3.6); the difference lies in the values of α_i and b.

Secondly, this method is only defined for input $\mathbf{x}_i \in \mathcal{H}$, and not the more general $x_i \in \mathcal{X}$; the difference being that $\mathcal{H}$ requires a well-defined inner product. Returning to our example of the space of all graphs, there is no clear way to define the inner product of two graphs. To generalise the SVM, we need to use the *kernel trick* as described in the next section.

3.3 Kernels

In order to use the SVM method described in the previous section when there is no inner product defined for the input space $\mathcal{X}$, we can map the input to another space $\mathcal{H}$ which has an inner product. In other words, we need a mapping $\Phi : \mathcal{X} \to \mathcal{H}$ which conserves the important features of the input data.

Given such a mapping, the decision function from eq. (3.6) turns into

$$f(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{M} y_i \alpha_i \langle \Phi(x), \Phi(x_i) \rangle + b\right) \tag{3.8}$$

The key to the kernel trick lies in observing that the mapping Φ only appears as part of expressions on the form $\langle \Phi(x), \Phi(x') \rangle$. By denoting this function

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \tag{3.9}$$

the need to directly evaluate Φ, or even to explicitly define it, disappears.

Symmetry and positive definiteness

In order to ensure applicability of the SVM results above, we need to prove that the chosen kernel function actually does correspond to a mapping to an inner product space. Fortunately, Aronszajn (1950) proves that every *symmetric*, *positive definite* kernel function has a corresponding mapping in a reproducing kernel Hilbert space. Being a Hilbert space, such a space has an inner product.

The symmetry condition means that for any x, x', we have

$$k(x, x') = k(x', x)$$

Positive definiteness of a kernel k means the following condition holds for any nonzero coefficient vector $\mathbf{c} = [c_1, c_2, ..., c_M]$:

$$\sum_{i=1}^{M} \sum_{j=1}^{M} c_i c_j k(x_i, x_j) > 0 \tag{3.10}$$

In other words: if a kernel function is symmetric and positive definite, it implicitly defines a mapping Φ to an inner product space, and can be used freely in an SVM setting.

A common example of the kernel trick is the RBF kernel (Chang et al. 2010), defined as:

$$k_G(x, x') = \exp\left(\frac{-||x - x'||^2}{2\sigma^2}\right)$$

It is positive definite, but the Hilbert space to which its corresponding mapping maps the input is infinite-dimensional and thus impractical to define explicitly.

Graph kernels

To return to the application central to this thesis, let $\mathcal{X}$ be a suitable subspace of the space of all graphs (for instance, all connected, undirected, unlabeled graphs). Numerous *graph kernels* have been defined on such sets, aiming to extract important features from the larger graph data structure.

The next chapter will go into detail about one such kernel, the *shortest-path kernel*. It is, however, only one among many graph kernels used in various settings. Other illustrative examples are the random walk kernel (Gärtner, Flach, and Wrobel 2003), the graphlet kernel (Shervashidze et al. 2009), and the Weisfeiler-Lehman kernels (Shervashidze et al. 2011). We will describe them informally here, in order to show different approaches to the problem of constructing an informative mapping of graph pairs to real numbers.

The *random walk kernel* is based on counting random walks of a given length. The idea is that the structure of the graph—clusters, degree distribution and so on—inform the number of possible unique walks of a given length. Each graph G is mapped to a feature vector $\mathbf{x}$ where each element x_i is the number of unique walks of length i in the graph. A kernel value can then be computed by taking $k(G, G') = \mathbf{x}\mathbf{x}'^{\top}$. The random walk kernel can be extended to the set of node-labeled graphs by sorting walks of the same length into different categories based on the labels of the nodes in the walk.

The *graphlet kernel* (of order h) is similar, in that it constructs a feature vector $\mathbf{x}$ where each element is the count of some substructure found in the graph G. In the case of the graphlet kernel, that substructure is not a random walk but a *graphlet*, which is one of the possible sub-graphs with h nodes. For example, there are 2 (undirected) graphlets of size 2: the one with 2 nodes connected by an edge, and the one with 2 nodes without any edge. For size 3, the number of graphlets is 4; for size 4 it becomes 11.

The *Weisfeiler-Lehman family* of graph kernels is based on the graph isomorphism test with the same name. In short, the isomorphism test works by constructing new labels for each node based on the labels of its neighbours. In each iteration of the algorithm, these labels are compressed, and the process is repeated. After n iterations, every node label will in a sense contain information from the entire graph. Based on this algorithm, any algorithm which takes node labels into account can be augmented by applying it to a graph with labels according to a given iteration h of the Weisfeiler-Lehman isomorphism test. For example, the Weisfeiler-Lehman random walk kernel is defined by applying the random walk kernel to graphs as they are labelled after a given number of iterations of the isomorphism test.

Chapter 4

Shortest-path kernels

The shortest-path kernel is based on the idea that graphs can be compared by looking only at the distribution of shortest-path lengths within each graph. Intuitively, a densely connected graph will have very short paths available between most pairs of nodes. In the same way, a sparse graph will have several node pairs between which the shortest possible path is comparatively long.

Alternately, we can view the shortest-path kernel as a special case of the walk kernel (section 3.3). The walk kernel is computed by counting similar walks in both graphs, with "similar" being defined as "of the same length". Going from general to more specific, we can decide to only consider paths (i.e. non-repeating walks), and then restrict ourselves further to the shortest paths between each node-pair.

This also removes a potential problem with walk kernels: as the length of the walk grows, the number of possible walks also grows, regardless of graph structure. This necessitates some sort of decay factor or similar, to ensure convergence (Borgwardt and Kriegel 2005). The number of shortest paths, in contrast, is limited to one per distinguishable node pair, i.e. n^2 (or $n(n+1)/2$, in an undirected graph).

Based on Borgwardt and Kriegel (ibid.), we define the general shortest-path kernel $k_{\text{sp}}(G, G')$ as follows:

$$k_{\text{sp}}(G, G') := \sum_{v_i, v_j \in V} \sum_{v'_k, v'_l \in V'} k_{\text{length}}(d(v_i, v_j), d(v'_k, v'_l)) \tag{4.1}$$

Here, $d(u, v)$ is the length of the shortest path from u to v in the relevant graph. Furthermore, k_{length} is an inner kernel which compares two lengths. (Recall that in an unlabelled graph, the length is the number of edges on the shortest path; in a weighted one it is the total edge cost.) The choice of this inner kernel will influence the behaviour of the outer graph kernel.

The two simplest choices for the inner kernel are the delta kernel k_δ and the linear kernel k_{linear}. The former is the well-known delta function, comparing for equality, and the latter returns the product of the distances. Since the delta function compares for equality, general real-valued path lengths would pose a problem. In this thesis, we assume that all edge labels are integers, resulting in integer-valued path lengths. A corresponding shortest path kernel for graphs with real-valued edge labels can be defined by binning path lengths, but we do not pursue this possibility in this thesis.

Using either the linear or the delta inner kernel, the expression for k_{sp} simplifies greatly.

The linear kernel Using k_{linear}, the expression for the graph kernel becomes

$$k_{\text{sp-linear}}(G, G') = \sum_{v_i, v_j \in V} \sum_{v'_k, v'_l \in V'} d(v_i, v_j) \times d(v'_k, v'_l) =$$
$$= \left(\sum_{v_i, v_j \in V} d(v_i, v_j) \right) \left(\sum_{v'_k, v'_l \in V'} d(v'_k, v'_l) \right)$$

The kernel can then be computed only from the sum of all shortest-path lengths within each graph, also known as the Wiener index $W(G)$ (Wiener 1947).

The delta kernel Using k_δ, the expression instead becomes

$$\begin{aligned} k_{\text{sp-}\delta}(G, G') &= \sum_{v_i, v_j \in V} \sum_{v'_k, v'_l \in V'} \delta(d(v_i, v_j), d(v'_k, v'_l)) = \\ &= \sum_{s=0}^{n} \sum_{v_i, v_j \in V} \sum_{v'_k, v'_l \in V'} \delta(d(v_i, v_j), s) \times \delta(d(v'_k, v'_l), s) = \\ &= \sum_{s=0}^{n} \left(\sum_{v_i, v_j \in V} \delta(d(v_i, v_j), s) \right) \left(\sum_{v'_k, v'_l \in V'} \delta(d(v'_k, v'_l), s) \right) = \\ &= F F'^{\top} \end{aligned}$$

In the final equality, we employ the dot product of the frequency vectors F and F', defined as follows: the element F_i is the number of node pairs within G whose shortest-path length is equal to i. How to treat the paths of length 0 is a minor issue; with the above definition the frequency vectors have to be indexed starting at 0.

Presenting the kernel in this form also makes it natural to explore a normalised version, where the frequency vector F is replaced by the distribution vector $\varphi = F/\text{sum}(F)$.

4.1 Information value

In terms of information, $k_{\text{sp-}\delta}$ has the advantage over $k_{\text{sp-linear}}$. As shown above, the latter can be reduced to a single value per graph, which is multiplied to give the kernel value. No matter how clever this value is, it is still one-dimensional; as a similarity measure it is very simple. It is included in this thesis to show that there is more than one kernel in the shortest-path family.

We instead consider the version with the delta kernel as the inner kernel. Here too we can talk about information: by normalising the frequency vector F to obtain φ, we throw away one piece of information. It is the number of node pairs whose shortest-path lengths are counted. This number does not depend on the graph's structure, however: for any connected graph of size n, the sum of F will be $n(n+1)/2$. This means that when comparing only graphs of the same size n, the normalised shortest-path kernel contains all relevant information.

We will use the normalised version; thus from this point onwards we will use the definition

$$k_{\text{sp-}\delta} = \varphi\varphi'^{\top} \tag{4.2}$$

where φ_i is the *proportion* of node pairs in G whose shortest-path lengths are equal to i.

4.2 Time complexity

To compute the delta-based $k_{\text{sp-}\delta}$, we need to find the frequency vectors. In section 2.3, we introduced the Floyd-Warshall algorithm which solves the all-pairs shortest-path problem in $O(n^3)$ time. Its output is a distance matrix S such that $S_{ij} = d(v_i, v_j)$. Given this, what remains is to create the two frequency vectors and multiply them; both tasks can be done in $O(n^2)$. The full algorithm is presented algorithm 1 in for future reference.

Algorithm 1 SP-KERNEL; for computation of $k_{\text{sp-}\delta}(G, G')$

1: Solve the all-pairs shortest path problem for G and G'; store distances.
2: Construct φ and φ' by tallying the distances of each integer length and normalising the resulting vectors.
3: Compute $k_{\text{sp-}\delta} = \varphi\varphi'^{\top}$.

The total time complexity of computing $k_{\text{sp-}\delta}(G, G')$ is in $O(n^3)$. However, the Floyd-Warshall pre-processing step can be performed independently for each graph. This means that for a set of N graphs $\{G_i\}_{i=1}^{N}$, each of size n, computing $k_{\text{sp-}\delta}(G_i, G_j)$ for all $i, j \leq N$ only takes $O(Nn^3 + N^2n^2)$, and not $O(N^2n^3)$ as the pairwise $O(n^3)$ suggests.

4.3 Approximating the kernel

All kernels explored in this thesis are approximations of the delta-based shortest-path kernel. Further, they all arrive at the final kernel value specifically by approximating the shortest-path-length distributions φ and φ'. In other words, we seek to find a $\hat{\varphi}$ which approximates the true distribution φ. We can then define an approximate kernel

$$\hat{k} = \hat{\varphi}\hat{\varphi}'^{\top} \tag{4.3}$$

In the following chapters, we explore two approaches to approximating φ. This first is to *sample a limited number of node pairs*, and compute the shortest-path length for each sampled pair only. The second uses sampling as well, but further computes only an approximate shortest-path length for each sampled node pair, using a precomputed *Voronoi dual graph*. Both approaches are further described and motivated below.

The chief purpose of any approximation scheme is to improve computational time (or possibly space) at the cost of a (hopefully negligible) loss of performance. Thus, an important part of the following chapters' analyses is the quantification of the time complexity improvement and the error, i.e. distance to the original kernel $k_{\text{sp-}\delta}$.

Chapter 5

A sampling-based approximation

The delta kernel version of the shortest-path kernel computes a similarity measure on graphs based on information from all possible node pairs within each graph. However, we use a normalised version which only uses the proportions of pairs with a given shortest-path length, not the absolute ones. (As an aside, we could find the absolute counts from the relative ones through multiplication by the number of node pairs.) For a large graph, we can approximate this shortest-path length distribution by sampling node pairs from the graph. A similar method was used by Shervashidze et al. (2009), who approximate the graphlet kernel by sampling subgraphs of a given size h to approximate the graph's distribution over size-h graphlets.

In this section, we define a sampling approximation of the shortest-path kernel and analyse its error (compared to the original $k_{\text{sp-}\delta}$). Moreover, we present two algorithms to compute it, analyse their respective time complexity, and discuss their differences and the importance of each.

5.1 The approximate kernel $k_{\text{sp-sampled}}$

As established in section 4.3, we define our approximate kernel in relation to an approximate shortest-path length distribution:

$$\hat{k}_{\text{sp-}\delta} = \hat{\varphi}\hat{\varphi}'^{\top} \tag{5.1}$$

Now, let

$$\hat{\varphi}_i = \frac{1}{M}\sum_{j=1}^{M} \delta(d(v_{j1}, v_{j2}), i) \tag{5.2}$$

where v_{j1}, v_{j2} constitute the j:th pair, out of $M \ll n^2$ node pairs sampled from V. Simply put, we sample M node pairs, tally the lengths of their respective shortest paths, and normalise the resulting distribution by dividing by M. Denote the approximate kernel computed from this choice of $\hat{\varphi}$ by $k_{\text{sp-sampled}}$.

5.2 An error bound on sampled distributions

To find an error bound for our kernel, we first look at a theorem from Weissman et al. (2003). They establish a relationship between the number of samples taken from a general probability distribution φ over a discrete set, and the distance from the distribution $\hat{\varphi}$ of those samples to φ. In our case, φ is the distribution of shortest-path lengths in a graph.

Theorem 5.1. *(Adapted from Weissman et al. (ibid.))*

Given $\phi(p) := \frac{1}{1-2p} \log \frac{1-p}{p}$ *for* $p \in [0, 1/2)$*;*

and given π_P *as a function of a probability distribution* P*, such that* $\pi_P \leq 1/2$ *for all* P*;*

let φ *be a probability distribution on the set* $\mathcal{A} = \{1, ..., a\}$*. Let* $\mathbf{X}^M = X_1, X_2, ..., X_M$ *independent identically distributed random variables distributed according to* φ*. Then, for all* $\gamma > 0$*,*

$$Pr(||\varphi - \hat{\varphi}_{\mathbf{X}^M}||_1 \geq \gamma) \leq (2^a - 2)e^{-M\phi(\pi_\varphi)\gamma^2/4} \tag{5.3}$$

Starting from this, we denote the probability in the LHS of eq. (5.3) by ξ, and set out to derive a bound for the number of samples M needed to achieve a given ξ and γ:

$$\xi \leq (2^a - 2)e^{-M\phi(\pi_P)\gamma^2/4}$$

which leads to

$$M \leq \frac{4}{\gamma^2 \phi(\pi_P)} \left(\log (2^a - 2) + \log \frac{1}{\xi} \right)$$

Using the two properties $\frac{4}{\phi(\pi_P)} \leq 2$ and $\log (2^a - 2) \leq a \log 2$, we arrive at the final inequality

$$M \leq \frac{2}{\gamma^2} \left(a \log 2 + \log \frac{1}{\xi} \right) \tag{5.4}$$

which ties the number of samples to the two error measures ξ and γ.

In our setting, the number of elements a in the set $\mathcal{A}$ is the length of the longest shortest-path in the graph; i.e. the graph's diameter. The important implication of this bound is that the number of samples M only depends on the graph size in this indirect fashion; we shall expand on this in the next section.

5.3 An error bound on the approximate kernel value

The error bound derived above applies to a specific empirical distribution $\hat{\varphi} = \hat{\varphi}_{\mathbf{X}^M}$, sampled from a true distribution φ. We are interested in a similar bound applied to the entire kernel function, i.e. bounding the probability δ that the error $|k_{\text{sp-sampled}} - k_{\text{sp-}\delta}|$ is larger than a given ϵ. Based on the previous bound, we know that the following holds with probability $1 - \xi$:

$$||\varphi - \hat{\varphi}||_1 \leq \gamma \tag{5.5}$$

This is true for both graphs, which means that it holds for both φ and φ' separately.

The error we are interested in can be written

$$\begin{aligned}
|k_{\text{sp-sampled}} - k_{\text{sp-}\delta}| &= |\hat{\varphi}\hat{\varphi}'^{\top} - \varphi\varphi'^{\top}| = \\
&= \left| \left(\sum_{i=1}^{a} \hat{\varphi}_i \hat{\varphi}'_i \right) - \left(\sum_{i=1}^{a} \varphi_i \varphi'_i \right) \right| \leq \\
&\leq \sum_{i=1}^{a} |\hat{\varphi}_i \hat{\varphi}'_i - \varphi_i \varphi'_i| = \\
&= \sum_{i=1}^{a} |\hat{\varphi}_i (\hat{\varphi}'_i - \varphi'_i) - \varphi'_i (\hat{\varphi}_i - \varphi_i)| \leq \\
&\leq \sum_{i=1}^{a} |\hat{\varphi}_i (\hat{\varphi}'_i - \varphi'_i)| + \sum_{i=1}^{a} |\varphi'_i (\hat{\varphi}_i - \varphi_i)| \leq \\
&\leq \sum_{i=1}^{a} |\hat{\varphi}'_i - \varphi'_i| + \sum_{i=1}^{a} |\hat{\varphi}_i - \varphi_i| \leq \\
&\leq \left[\text{with probability } (1-\xi)^2\right] \leq 2\gamma = \epsilon
\end{aligned}$$

The penultimate step uses the fact that $\varphi'_i \leq 1$ and $\hat{\varphi}_i \leq 1$ for all i, since both are probability distributions, each summing to 1. The last step uses eq. (5.5) for both φ and φ'. However, eq. (5.5) only holds with probability $1 - \xi$, and our result requires it to hold for two independent empirical distributions at the same time. This means our result only holds with probability $1 - \delta = (1 - \xi)^2$. In conclusion, we have:

Corollary 5.1.1. *To achieve an error $|k_{sp\text{-}sampled} - k_{sp\text{-}\delta}| \leq \epsilon$ with probability $1 - \delta$, we must sample M node pairs from each graph, where*

$$M = \left\lceil \frac{8 \left(a \log 2 + \log \frac{1}{1 - \sqrt{1-\delta}} \right)}{\epsilon^2} \right\rceil \tag{5.6}$$

Like the error bound on the distribution itself, this bound does not depend directly on the size of either graph but instead on their diameters. (Eq. (5.6) only features a single a, which is the largest of the two graphs' diameters.) For a given error tolerance setting (ϵ, δ), this ensures that our criterion of $M \ll n^2$ is met by large margin, since $M \in O(a)$. In the (degenerate) worst-case where all nodes in the graph lie along a single chain, we have $a = n - 1$, and for real-life graphs we believe M will be much smaller. For some classes of graphs the bound $a = \operatorname{diam}(G) \leq n - \Delta(G) + 1$ (Goddard and Oellermann 2011), where $\Delta(G)$ is the maximum vertex degree in G, might be relevant.

5.4 Algorithms to compute $k_{\text{sp-sampled}}$

Computing the value $k_{\text{sp-sampled}}(G, G')$ entails two things: sampling M node pairs, and finding the lengths of the shortest paths between each of them. The first task is not very complicated – virtually every programming language features a pseudo-random number generator that will do the trick. Theorem 5.1 defines the empirical distribution as sampled with replacement.

The second task—finding the shortest-path lengths between pairs of nodes—is where the real work of the algorithm takes place. It can be done in several ways. A sort of "brute force" solution is to take a page from the original kernel $k_{\text{sp-}\delta}$, and compute all shortest-path lengths beforehand. If all shortest-path lengths are available, the entire kernel computation can be done in $O(M)$ time, since each sampled pair only requires an $O(1)$ lookup. This version, which we denote SMP-LAST, is presented more formally in algorithm 2.

Algorithm 2 SMP-LAST computation of $k_{\text{sp-sampled}}(G, G')$

1: Solve the all-pairs shortest path problem for G and G'; store distances.
2: Sample M node pairs (v_{i1}, v_{i2}) from each graph.
3: Look up the distances $d(v_{i1}, v_{i2})$ and $d(v'_{i1}, v'_{i2})$.
4: Construct $\hat{\varphi}$ and $\hat{\varphi}'$ by tallying the distances of each integer length and dividing by M.
5: Compute $k_{\text{sp-sampled}} = \hat{\varphi}\hat{\varphi}'^{\top}$.

However, this method does not fully leverage the advantages of sampling. It computes all shortest paths, even the ones that are never used! A more efficient version, called SMP-FIRST, is presented in algorithm 3. It differs from SMP-LAST on one point: the shortest paths are computed after sampling, rather than before.

Algorithm 3 SMP-FIRST computation of $k_{\text{sp-sampled}}(G, G')$

1: Sample M node pairs (v_{i1}, v_{i2}) from each graph.
2: Compute shortest paths to find distances $d(v_{i1}, v_{i2})$ and $d(v'_{i1}, v'_{i2})$.
3: Construct $\hat{\varphi}$ and $\hat{\varphi}'$ by tallying the distances of each integer length and dividing by M.
4: Compute $k_{\text{sp-sampled}} = \hat{\varphi}\hat{\varphi}'^{\top}$.

Mathematically, the two algorithms compute the same kernel. If the random selection of nodes is performed in a given way with a given random seed, both algorithms would arrive at the exact same kernel value for the same input. The above error analysis holds for either algorithm, and the interesting difference lies in their respective time complexities.

5.5 Time complexity

The analysis for SMP-LAST depends on which algorithm is used for the all-pairs shortest path step, but is apart from that fairly straightforward. The sampling, lookup, empirical distribution construction and dot product steps can each be performed in $O(M)$. Given an all-pairs shortest path algorithm running in $p(n)$ time, the total complexity becomes $O(p(n) + M)$. Since $M < n$, this becomes $O(p(n))$ for any all-pairs shortest path algorithm we can imagine. For example, we can use the Floyd-Warshall algorithm running in $p(n) = n^3$ for a total time complexity in $O(n^3)$.

SMP-FIRST has no preprocessing, but instead we have two shortest-path queries for each sample (since the number M denotes the number of sampled pairs from each graph). As seen in section 2.3, without some domain-specific knowledge, for example in the form of a distance heuristic, our best option for pairwise shortest-path queries is Dijkstra's algorithm. This could give us a runtime in $O(Mn \log n)$. Our implementation uses the slightly worse binary heap formulation of Dijkstra's algorithm, yielding an expected runtime in $O(Mn \log \frac{m}{n} \log n)$.

The justification for not using a state-of-the-art version of Dijkstra's algorithm is about fairness. The chief purpose of this thesis is to compare our kernel approximations with the original kernel $k_{\text{sp-}\delta}$, which relies on $O(n^2)$ single-pair shortest-path queries; or $O(n)$ single-source ones. The thrust of our argument for using $k_{\text{sp-sampled}}$ by way of SMP-FIRST is that we only need $O(M)$ single-pair queries. Any progress with regards to Dijkstra's algorithm's performance would improve the runtime of both the original kernel and the approximation by the same factor, if it were fully implemented. Improving one by using state-of-the-art versions of Dijkstra's algorithm without doing so for the other would make for a very unfair comparison.

With this in mind, the specific version of Dijkstra's algorithm used is not of great importance to this thesis, as long as it is used consistently. The motivation for choosing the binary heap version is twofold. Firstly, more improved versions such as those by Thorup (2003) require specific representations and models, and that complexity would detract from the presentation of our new approximations. Secondly, the binary heap version of Dijkstra's algorithm was easy to use for us, but still allowed the test runs to finish in shorter time than would an even simpler priority queue.

5.6 A note on positive definiteness

In order to be a proper kernel function, usable in an SVM setting, our kernel must be positive definite. The full kernel $k_{\text{sp-}\delta}$ is proven to be positive definite by Borgwardt and Kriegel (2005). However, it is hard to prove positive definiteness for the sampled kernel without the following adjustment: for any graph G, the set of pairs sampled from that graph must be the same in the computation of $k(G, G')$ for all G'. If we ensure this, then our kernel defines a unique vector for every possible graph, and the kernel value is simply a dot product of the two graphs' vectors.

One possible way to ensure this while still maintaining the idea that the samples are "random" (anything computed by computers will of course actually be pseudorandom), is as follows. Define some unique representation of graphs, encoded as a string of bits of a fixed size. (This will limit the maximum graph size the kernel can handle, but that maximum size can be set arbitrarily large.) To determine the "randomly sampled pairs" shortest-path length distribution vector, use the graph's bit string as random seed to a specific pseudorandom number generator. Without prior knowledge of the encoding scheme, there is no reason to assume that the random sampling will be biased in any particular way; yet each graph will only ever receive a single set of sampled pairs.

The above is only a sketch, and in the experimental setup described in chapter 7 does not include this complex insurance of positive definiteness. Instead, we sample node pairs from each graph only once within each run. This means that technically, a different (positive definite) kernel is used for each run.

Chapter 6

A Voronoi-based approximation

In section 4.2, we briefly raised the point of precomputation versus query-time computation, noting the advantage of the Floyd-Warshall algorithm in that it need only be computed once for each graph to produce the shortest paths for all node pairs. When computing SMP-FIRST, however, we wanted to compute only a small ($M < n$) number of shortest-path queries. Dijkstra's algorithm, which does all the work at query time, was suitable for this task. The Floyd-Warshall algorithm, working solely with precomputation, was not.

Might there be a middle ground; an algorithm which performs some general precomputation in order to reduce the query-time runtime later? Honiden et al. (2010) present an *approximate* algorithm which does just that, by using a method based on the concept of the *graph Voronoi dual*. Apart from providing the inspiration for this entire line of inquiry, Honiden et al. (ibid.) also supplies most of the results in sections 6.1 through 6.2.

A Voronoi diagram in general is a decomposition of a metric space relative to a discrete set of *sites*, with many different applications (Aurenhammer 1991). Honiden et al. (2010) defines a corresponding graph decomposition, called a graph Voronoi diagram. A regular Voronoi diagram divides a space into "neighbourhoods" of the given sites – an example is seen in fig. 6.1. Each region is defined by being closer to a certain site than to any of the other sites. The graph Voronoi diagram does much the same thing with a graph – see fig. 6.2. The formal definition is given in the following section.

Graph Voronoi methods can be used to compute approximate shortest-path queries quickly (ibid.). In section 6.3, we define an approximate kernel based on this method. From here onwards, we will refer to graph Voronoi methods as simply Voronoi methods, since the ones pertaining metric spaces are irrelevant in this context.

6.1 Preliminary Voronoi method definitions

The basic idea of the Voronoi diagram is to choose the sites—a set $C \subseteq V$ of *Voronoi nodes*—and assign all nodes in V to their closest Voronoi nodes. The graph will then be partitioned in $k = |C|$ groups, each containing at least its own Voronoi node, and additionally all nodes closer to it than to any other Voronoi node. We will refer to these groups as Voronoi regions, and denote them $V_1, ...V_k$.

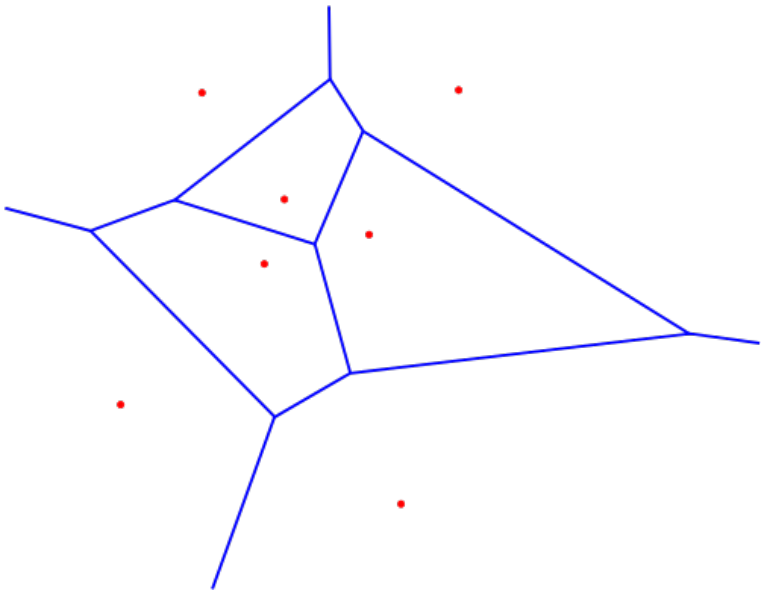

Figure 6.1: Voronoi diagram of seven Voronoi sites in the plane.

How to choose the set C is not a given. In this work, like Honiden et al. (2010), we elect to choose them randomly; considering each node for inclusion with a probability p_v.

The formal definition of the Voronoi diagram is as follows:

Definition 6.1. *(ibid.)*
In a graph $G = (V, E, \omega)$, the Voronoi diagram for a set of nodes $C = \{v_1, ..., v_k\} \subseteq V$ is a disjoint partition $\texttt{Vor}(G, K) := \{V_1, ..., V_k\}$ of V such that for each node $u \in V_i$, $d(u, v_i) \leq d(u, v_j)$ for all $j \in \{1, ..., k\}$.

In addition, let $\texttt{vor}(u)$ denote the index of the voronoi region V_i such that $u \in V_i$. Moreover, we can construct a *Voronoi dual*, which is a graph $G^\star$ with only the Voronoi nodes. In other words, we have $G^* = V$. The edges correspond to paths between Voronoi nodes (in the original G) which go from one Voronoi region to another without passing a third region. The edges' weights are the length of these paths in G. The formal definition is:

Definition 6.2. *(ibid.)*
Let $G = (V, E, \omega)$ be an edge-weighted graph and $Vor(G, K)$ its Voronoi diagram. The Voronoi dual is the graph $G^\star = (K, E^\star, \omega^\star)$ with edge set $E^\star := \{(v_i, v_j) : v_i, v_j \in K \text{ and } \exists u \in V_i \wedge \exists w \in V_j : (u, w) \in E\}$, and edge weights $\omega^\star(v_i, v_j) := \min_{\substack{u \in V_i, w \in V_j \\ (u,w) \in E}} \{d(v_i, u) + \omega(u, w) + d(w, v_j)\}$

From this, an approximate distance function is not far away. For starters, the distance between two nodes in the Voronoi dual $G^\star$ is an approximation of the distance between the corresponding nodes in G.

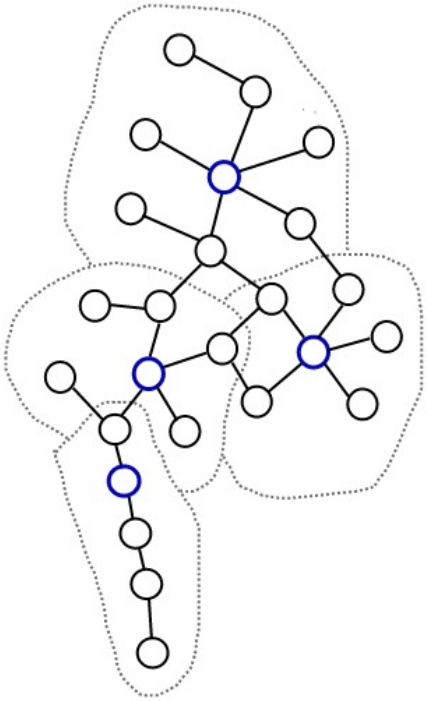

Figure 6.2: Graph Voronoi diagram, where four Voronoi nodes (shown in blue) determine a partition of the (unlabeled) graph.

6.2 Approximate shortest-path queries

In order to answer approximate shortest-path queries in G, we first create the Voronoi dual; see algorithm 4. Line 2 references another algorithm, `ComputeVoronoiDual`, which we will not present in full here. It is given in Honiden et al. (2010), along with a proof that its asymptotic running time is the $O(m+n \log n)$ of Dijkstra's algorithm. It is possible that, like the others uses of Dijkstra's algorithm discussed in section 4.2 and section 2.3, `ComputeVoronoiDual` can also be performed faster in some settings.

Algorithm 4 Preprocessing for $k_{\text{sp-Voronoi}}$ (Algorithm 2 in Honiden et al. (2010))

Input: graph $G = (V, E, \omega)$, sampling rate $p \in [0, 1]$.
Output: Voronoi dual G^* with Voronoi nodes selected independently at random with probability p.

1: Random sampling: Generate the set of Voronoi nodes by selecting each node of V independently at random: $\forall v \in V : \Pr[v \in K] = p$.
2: Compute a Voronoi dual $G^* = (K, E^*, \omega^*)$ using the `ComputeVoronoiDual` algorithm from Honiden et al. (2010).
3: Return G^*.

Before we can present the next stage of the algorithm, we need two definitions: the *Voronoi Path* and the *Voronoi sleeve*:

Definition 6.3. *(Honiden et al. 2010)*
Given a path $P = (u_0, u_1, ..., u_h)$, the Voronoi path of P is the sequence of vertices $P^* = (v_{\texttt{vor}(u_0)}, v_{\texttt{vor}(u_1)}, ..., v_{\texttt{vor}(u_h)})$.

Definition 6.4. *(ibid.)*
For a path P^* in the Voronoi dual G^* of a graph G, the Voronoi sleeve is the subgraph of G induced by the nodes in the union of all Voronoi regions V_i for which its Voronoi node v_i lies on P^*, i.e.

$$\texttt{Sleeve}_{(G,G^*)}(P^*) := G\left[\bigcup_{v_i \in P^*} V_i\right].$$

Now we can state algorithm 5, which performs approximate shortest-path queries. We denote the output (the approximate shortest path) $\hat{P}$ in place of the P used by Honiden et al. (ibid.).

Algorithm 5 SP-query for $k_{\text{sp-Voronoi}}$ (Algorithm 3 in Honiden et al. (2010))

Input: graph G, Voronoi dual G^*, source s, target t.
Output: an approximate shortest path $\hat{P}$ from s to t.

1: Find Voronoi source $v_{\texttt{vor}(s)}$ from s and Voronoi target $v_{\texttt{vor}(t)}$ from t. If thereby a shortest path $SP_G(s,t)$ has been found, return it.
2: Compute a shortest path from $v_{\texttt{vor}(s)}$ to $v_{\texttt{vor}(t)}$ in the Voronoi dual G^* : $SP_{G^*}(v_{\texttt{vor}(s)}, v_{\texttt{vor}(t)})$.
3: Compute the Voronoi sleeve $\mathcal{S} := \texttt{Sleeve}(SP_{G^*}(v_{\texttt{vor}(s)}, v_{\texttt{vor}(t)}))$.
4: Compute a shortest path from s to t in the Voronoi sleeve, $SP_{\mathcal{S}}(s,t)$.
5: Return $\hat{P} = SP_{\mathcal{S}}(s,t)$.

6.3 The approximate kernel $k_{\text{sp-Voronoi}}$

Now we can define an approximate kernel $k_{\text{sp-Voronoi}}$. Again, as per section 4.3:

$$k_{\text{sp-Voronoi}} = \hat{\varphi}\hat{\varphi}'^{\top} \tag{6.1}$$

The difference from last chapter's $k_{\text{sp-sampled}}$ is that this time, the M sampled distances whose tally makes up $\hat{F}$ are not exact. In other words, a node pair with a true shortest-path distance of d might be sampled and have an approximate distance of $\hat{d} \geq d$ computed and tallied. In addition to the error introduced by sampling, this would skew $\hat{F}$ towards its tail end, compared to the true F.

We use algorithm 6, SMP-VOR, to compute the kernel value. It is a slightly modified version of SMP-FIRST.

The note on positive definiteness in section 5.6 also applies to $k_{\text{sp-Voronoi}}$. Here, the same care must also be taken when randomly selecting the Voronoi nodes.

Algorithm 6 SMP-VOR computation of $k_{\text{sp-Voronoi}}(G, G')$

1: Compute the Voronoi duals G^* and G'^* using algorithm 4.
2: Sample M node pairs (v_{i1}, v_{i2}) from each graph.
3: Compute approximate shortest paths to find distances $\hat{d}(v_{i1}, v_{i2})$ and $\hat{d}(v'_{i1}, v'_{i2})$, using algorithm 5.
4: Construct $\hat{\varphi}$ and $\hat{\varphi}'$ by tallying the distances of each integer length and dividing by M.
5: Compute $k_{\text{sp-Voronoi}} = \hat{\varphi}\hat{\varphi}'^{\top}$.

6.4 Time complexity

Honiden et al. (2010) show that the preprocessing, i.e. algorithm 4, takes time proportional to that of Dijkstra's algorithm. The reason is that `ComputeVoronoiDual` is in fact Dijkstra's algorithm with modifications by Erwig (2000). Again, this means that the time complexity depends on the implementation. Honiden et al. (2010) review an assortment of variants of Dijkstra's algorithm along with their runtimes. In our code, we used a binary heap-based version, with a time complexity of $O(m + n \log \frac{m}{n} \log n)$.

Honiden et al. (ibid.) also show that an approximate shortest-path query (algorithm 5) can be made in constant time after a precomputation step of expected linear time. However, there are two issues with this result. Firstly, the analysis does not include the "refinement step" that makes up lines 3 and 4; only noting that the time spent on this step depends on the size of the Voronoi sleeve. Secondly, it relies on the choice of Voronoi node selection probability $p_v \in O(n^{-2/3})$.

With a general p_v, we can conclude that each approximate query without refinement takes at most $O((p_v n) \log(p_v n) + (p_v n)^2)$ which simplifies to $O(p_v^2 n^2)$. This brings the total time complexity of SMP-VOR to $O(m + n \log n + M p_v^2 n^2)$. Like the computation of $k_{\text{sp-}\delta}$ in section 4.2, the fact that part of the complexity comes from per-graph pre-processing means that computing $k_{\text{sp-Voronoi}}$ over a larger set of graphs is not as slow as this result first suggests. Computing $k_{\text{sp-Voronoi}}(G_i, G_j)$ for all $i, j \leq N$ takes $O(N(m + n \log n + N M p_v^2 n^2))$. In comparison, doing the same thing for $k_{\text{sp-sampled}}$ takes $O(N^2 M n \log n)$.

However, in the practical evaluation detailed in chapter 7 and onwards, we do conform to Honiden et al.'s (2010) choice of $p_v \in O(n^{-2/3})$. In our experiments, the algorithm is supplied a value of p_v on the form $p_v = cn^{-2/3}$. Thus, in practice SMP-VOR takes the parameter c.

Chapter 7

Experimental setup

Each approximation method we use poses two chief questions. Firstly, how good is it at comparing graphs? Secondly, how fast can it be computed? We can provide some theoretical answers to both these questions, and have done so in sections 5.3, 5.5 and 6.4; however, these answers are limited. They speak of worst-case scenarios, or expected values, and only give the asymptotic behaviour. To observe the approximations' behaviour on concrete instances and real-world problems, we must test them empirically.

7.1 Graph datasets

In order to give relevant and solid results, we believe that a graph dataset ought to meet several criteria. First, it should contain more than a few graphs. We cannot justify speaking about a kernel's behaviour on a certain type of graphs, if we only measure the behaviour on a very small sample of them. Secondly, it should be split into two or more labelled categories. This split constitutes the "right answer" to the classification task. Finally, we considered the size n of each graph. Since we are interested in the behaviour of each kernel as the graph size increases, we ideally want several sets of n-sized graphs, with similar properties across the sets, each with a different n.

Four of our datasets are structured exactly like this. Of those four datasets, two consist of graphs that are, to some extent, artificial. ER and GAUSS are both completely fabricated. For ER, we generated graphs of a given size using the Erdős-Rényi model (Erdős and Rényi 1959), and the two classes in the classification task correspond to two different values of the model's single parameter. For GAUSS, the model was based on points in the plane drawn from a multivariate Gaussian distribution. Both of these models are described further in section 7.3.

The other two provide more realistic settings. ROADS stems from two very large graphs ($\sim 10^6$ nodes), from which we sample subgraphs of the desired size using Metropolis sampling (Hubler et al. 2008). The original graphs model road networks in Texas and California, USA. PROTO is based on the same method, but the graphs being sampled from are social networks within a simulated population. In both ROADS and PROTO, the classification task consists of deducing which of the two original graph a given graph is a subgraph of – for instance, whether a ROADS graph is sampled from the Texan or the Californian road network.

The fifth and final dataset is the Dobson & Doig set (Shervashidze and Borgwardt 2012b), henceforth D&D, which has a more varied graph size. This is not suitable for assessing how the kernels' behaviours vary with graph size. However, it is included since it has been used for benchmarking several other kernels previously (Shervashidze et al. 2009; Borgwardt and Kriegel 2005; Borgwardt and Shervashidze 2009) and thus proves an interesting point of comparison. The graphs in D&D represent molecules, specifically proteins, and the classification task consists of determining which of them are enzymes and which are not.

We have modified the D&D dataset slightly: some of the graphs were not connected, and behaviour of $k_{\text{sp-sampled}}$ is not well-defined when sampling a node pair with no possible path. Thus, the version of D&D used here includes 1157 connected graphs, compared to the 1178 of the original dataset.

7.2 Comparative experiments

Chapter 8 presents five larger runs, each comparing all our kernel variants (each using a varied but not necessarily exhaustive array of parameters) on a single dataset.

These runs follow the same structure. Each dataset consists of several size groups, where "size" refers to the number of nodes n in the graphs. Each size group contains the same number of graphs N of that size; $N/2$ each from two different categories. These categories are the subject of the graph classification task the kernels are evaluated in. D&D is an exception, in that it is not split into size groups. Instead, it consists of a single batch of $N = 1178$ graphs whose sizes n_i varying between X and Y nodes.

In each size group, we compute the kernel value matrix for each of the $k_{\text{sp-}\delta}$ and $k_{\text{sp-Voronoi}}$ kernels, and twice for the $k_{\text{sp-sampled}}$ kernel: once using SMP-LAST and once using SMP-FIRST. The kernel value matrix contains one element for each pair of graphs, denoting their similarity. Since the three approximate ones are non-deterministic, we compute this matrix $t = 20$ times each. All values reported for the approximate kernels—kernel value error, classification accuracy etc.—are average values over these t trials. Within each trial, we made sure to avoid re-randomising selections as per the note in section 5.6.

Preprocessing and parameters

The algorithms SP-KERNEL, SMP-LAST and SMP-VOR include some preprocessing – the Floyd-Warshall algorithm and the Voronoi dual construction, respectively. We do this for each size group, recording the runtime.

Furthermore, the kernels take various parameters which affect their behaviour, sometimes greatly. These parameters include the number of samples M (see section 5.1) or the Voronoi node selection probability coefficient c (section 6.4). When comparing the results of one kernel across different parameter values, we compute all t kernel value matrices for each parameter setting.

Finally, the sampling kernels' behaviour depends heavily on the number of samples M. For some datasets, we ran the experiments for a number of different values of M; for some, only the largest value was used. In order for the experiments to finish in a practical timeframe, this maximum number of samples was $M = 200$. Considering the bound presented in eq. (5.6), this ensures (depending on the graph diameter a) a kernel value error smaller than 0.5 with a probability of 0.9. This is not a particularly strong bound, but we shall see that in practice, the kernel value error is often smaller than this.

SVM procedure

The classification accuracy of a kernel is determined from feeding its kernel values for all graph pairs into an SVM, which performed a classification with 10-fold cross-validation. This means that the SVM is "trained" on 80% of the data, and then assigned the task of classifying 10% of the data (chosen from the remaining 20%); this is repeated for 10 different training data–classification data partitions. The accuracy value is the mean of these 10 results.

7.3 Graph generation and subgraph sampling

This section will attempt to outline the very basics of the three graph-generation methods used: the Erdős-Rényi model, the Gaussian distribution model, and Metropolis-Hastings subgraph sampling.

Erdős-Rényi The model we use to generate the ER graphs is named after a paper by Erdős and Rényi (1959), even though the version of it we use here was presented around the same time by Gilbert (1959). The idea is to, given a number of vertices n, randomly and independently select each possible edge with the probability p. The nodes and the selected edges make up the generated graph.

Since we have not defined our kernel for disconnected graphs, we modify the algorithm slightly: when generating ER graphs, we simply reject any that are not connected and generate new ones in their place, until we have the desired number of graphs N.

Gaussian distributions The model behind our new GAUSS dataset is based on a multivariate Gaussian distribution of points in the plane. To generate a graph of n nodes, randomly generate n points from said distribution, with the mean vector $\mu = \mathbf{0}$ and covariance matrix $\mathbf{\Sigma} = p * \mathbf{I}$. Then consider each pair of nodes (u, v). If the distance d between them is less than a threshold d_t, add an edge from u to v with cost $\lfloor d/d_i \rfloor$, where $0 < d_i < d_t$.

Like the ER model, this does not reliably produce connected graphs. One modification that helps is to add a final step where each node u not connected to any other node is considered. It is connected to the node v such that the distance d between them is the smallest, by an edge with cost $\lfloor d/d_i \rfloor$, even if $d > d_t$. Even with this modification, connected graphs are not guaranteed, but the strategy of simply rejecting disconnected graphs works here too.

Metropolis We generate the datasets ROADS and PROTO by performing *representative subgraph sampling* on very large graphs. In representative subgraph sampling, the goal is to find a subgraph $G_s = (E_s, V_s)$ such that $E_s \subseteq E, V_s \subseteq V$ where $(E, V) = G$ is the larger graph, and so that G_s is "similar" to G in whatever metrics are most relevant. The Metropolis subgraph sampling algorithm used here is presented by Hubler et al. (2008), who also show that it generates high-quality (i.e. representative) subgraphs in general.

7.4 Reference kernels

In addition to SP-KERNEL, SMP-FIRST, SMP-LAST and SMP-VOR, we also compute the kernel matrices and performed the classification tasks using two other kernels. The first one is the Weisfeiler-Lehman subtree kernel, a member of the Weisfeiler-Lehman kernel family mentioned in section 3.3. The second one is the graphlet kernel of order 3, also sketched in section 3.3. We will not provide further detail on how those kernels work, but instead note that they have both been proposed as viable graph kernels (Shervashidze et al. 2009; Shervashidze et al. 2011). The algorithms computing these kernels are used courtesy of Shervashidze and Borgwardt (2012a), and we denote them W-L and GRAPHLET, respectively.

Chapter 8

Results

As described in section 7.1, the kernels were computed for each pair of graphs in each size group of each dataset. Each kernel was computed multiple times to explore different values of various parameters. The non-deterministic kernels were computed $t = 20$ times each (for a given parameter setting), and endpoint values presented here are averages over those t trials.

This section presents results from one dataset at a time, detailing the most interesting ones. The full set of results is included in section 9.3, in the form of two tables of accuracy and runtime figures, respectively. Even so, we have not put 100% of all data collected from these tests into print. Specifically, the full set of Dijkstra operation counts is omitted. Further, the tables only include one set of figures for SMP-VOR on each dataset, namely the one achieving the best overall classification accuracy on that data. Readers who would like access to the full results are invited to contact the author.

8.1 ER graphs

The ER dataset consists of 100 graphs each of sizes 100, 200, 500, 1000, and 2000 nodes. Within each size group, 50 of the graphs are generated by the ER model (see section 7.3) with parameter $p = 0.1$, and for the remaining 50 the value was $p = 0.109$.

Runtimes

Looking at only the query runtimes (shown in fig. 8.1), SP-KERNEL, SMP-LAST and SMP-VOR are notably faster than SMP-FIRST. At this stage, the first two of those only perform lookups in a matrix of the results from the Floyd-Warshall step, counting the elements of each length, and taking the dot product between the resulting distributions. That SMP-VOR—which performs both one Dijkstra search in the Voronoi dual graph and one in the Voronoi sleeve—outperforms both, is surprising. However, the difference is fairly small and parts of the Voronoi computations are performed in C, which muddies the issue.

The slower SMP-FIRST catches up when considering the total runtimes (shown in fig. 8.2) including preprocessing. Here, the time spent in the Floyd-Warshall step is

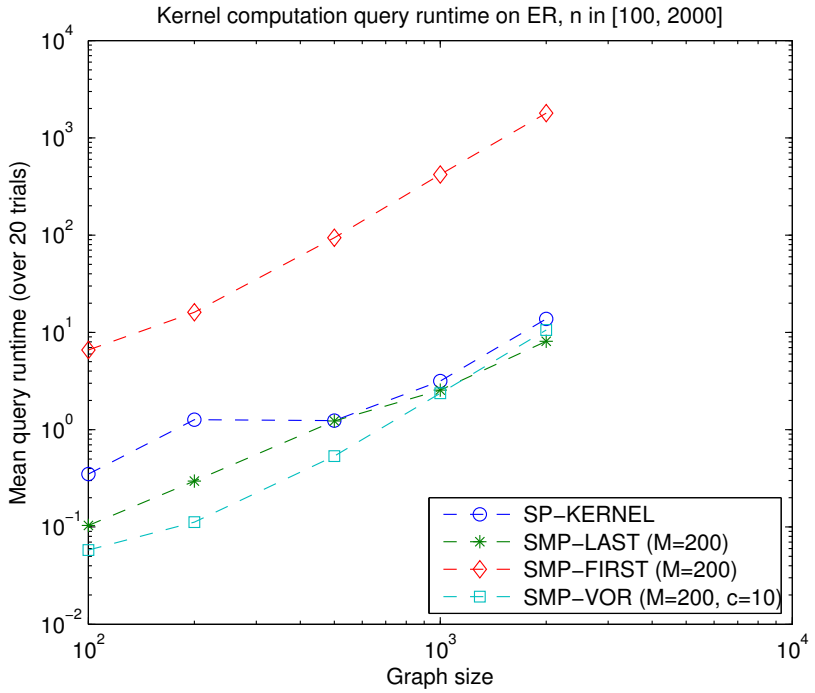

Figure 8.1: Query-time runtimes for all kernel variants on ER.

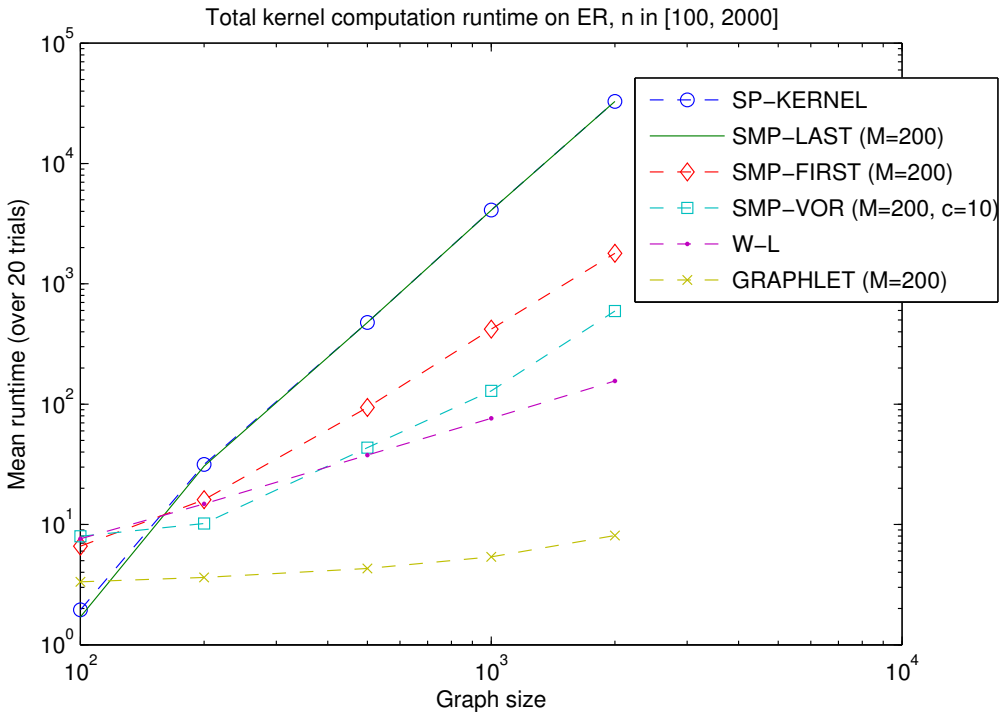

Figure 8.2: Total runtimes for all kernel variants on ER.

added to both SP-KERNEL and SMP-LAST, and the time for Voronoi dual construction is added to SMP-VOR. The Floyd-Warshall runtime dwarfs all other contributions, making the total runtimes of SP-KERNEL and SMP-LAST all but indistinguishable. SMP-VOR has far better performance, with small differences between different choices of the parameter c.

The reference kernels represented by W-L and GRAPHLET both perform generally better than the shortest-path kernel and its approximations.

Finally, fig. 8.3 shows how SMP-LAST, SMP-FIRST and SMP-VOR are affected by the number of samples M. The figure shows the results from the $n = 1000$ size group, but the results are very similar across graph sizes. SMP-FIRST and SMP-VOR both scale very predictably with the sample size (note the log-log-scale of fig. 8.3). For SMP-LAST, a larger M means the algorithm must tally more numbers, but they are all looked up in a precomputed matrix. The runtime is accordingly almost constant across different values for M.

The results described in the preceding paragraph hold very well for the other datasets as well. We will not look in detail at the relationship between runtimes and M for the rest of chapter 8.

Operation counts

In an attempt to show that the runtime advantage of SMP-VOR over SMP-FIRST is not entirely rooted in implementational details, we also tracked the number of relevant *operations* performed inside Dijkstra's algorithm, defined as the number of times two nodes exchanged positions in the binary heap representing the priority queue. The results are shown in fig. 8.4. For a given number of samples, SMP-VOR requires far fewer operations; this difference is about an order of magnitude at its largest.

Error

The kernel value error, presented in fig. 8.5, measures how well the approximations approximate the original function $k_{\text{sp-}\delta}$. Here, we see that the plain sampling algorithms SMP-LAST and SMP-FIRST are closer to it than SMP-VOR. Unsurprisingly, SMP-LAST and SMP-FIRST, being mathematically equivalent, are virtually identical in error.

Meanwhile, fig. 8.6 paints a similar picture. The bound from eq. (5.6), considering the case of $M = 200$, approximately works out to ensuring an error of 0.5 or less with probability 0.9 for SMP-FIRST. We have no corresponding bound on SMP-VOR, but it is clear that any such bound would necessarily be even worse than that of eq. (5.6).

It can be seen as a point against SMP-VOR that its kernel value error does not seem to decrease with the number of samples. Ought a sampling-based approximation not be more precise the more samples we take? However, SMP-VOR redeems itself when it comes to classification accuracy, as we shall see in the next section.

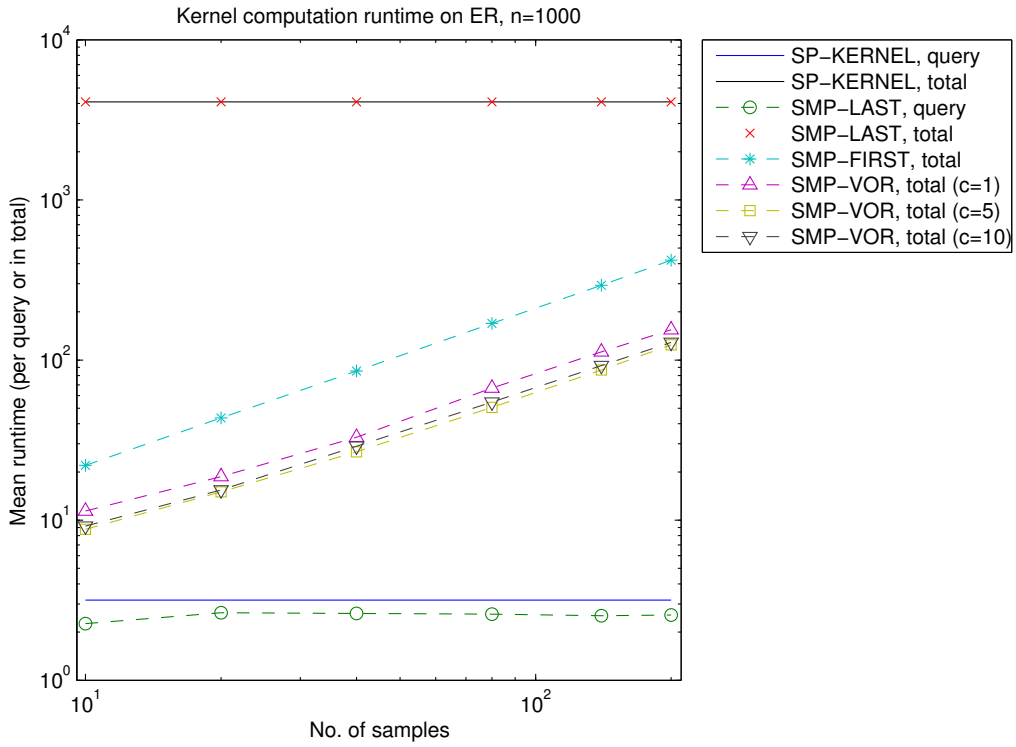

Figure 8.3: Runtimes for the sampling-based kernels on ER1000.

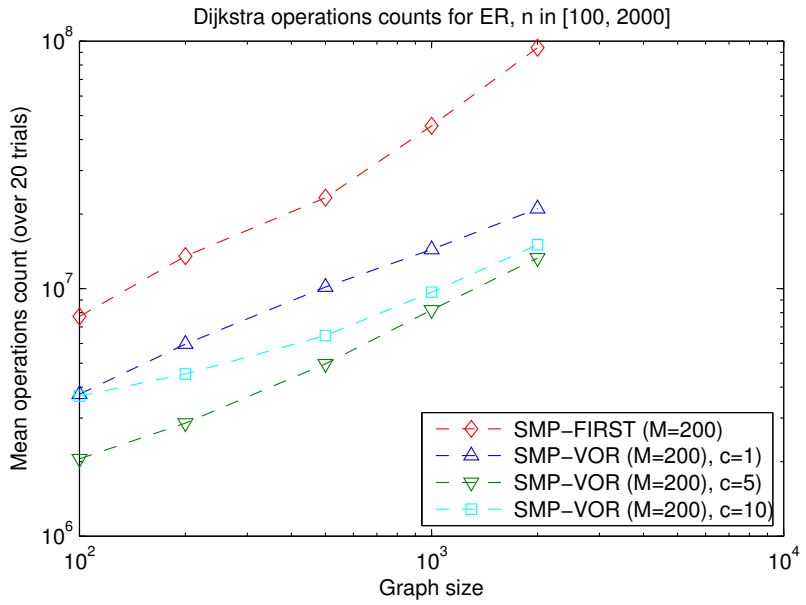

Figure 8.4: Operation counts (binary-heap exchange operations) for SMP-VOR and SMP-FIRST on ER.

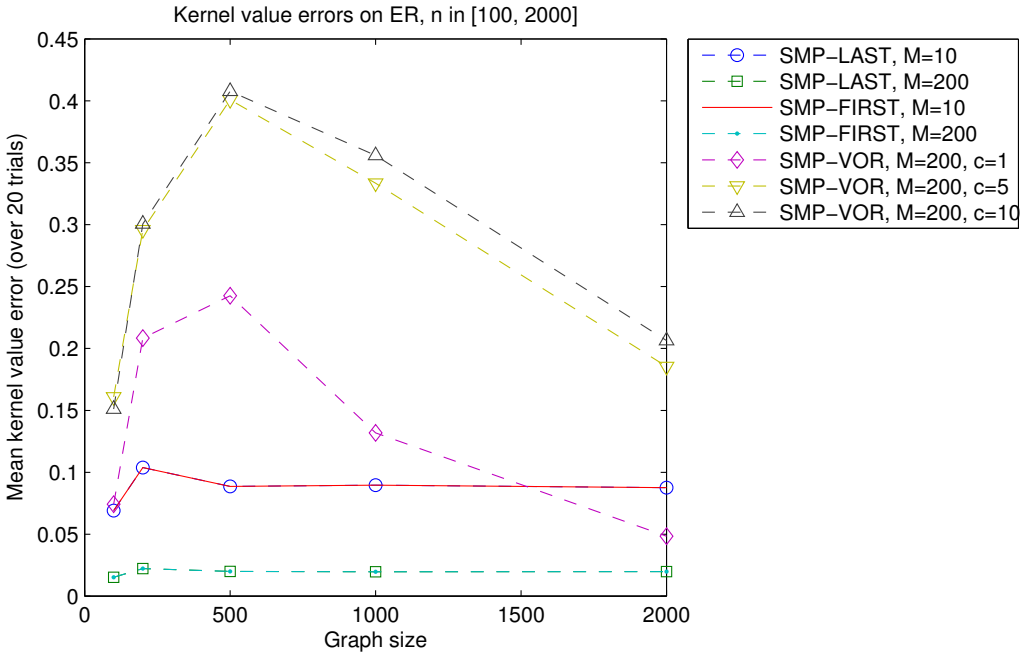

Figure 8.5: Kernel value errors, defined as $|\hat{k}_{\text{sp-}\delta} - k_{\text{sp-}\delta}|$, for the sampling-based kernels on ER.

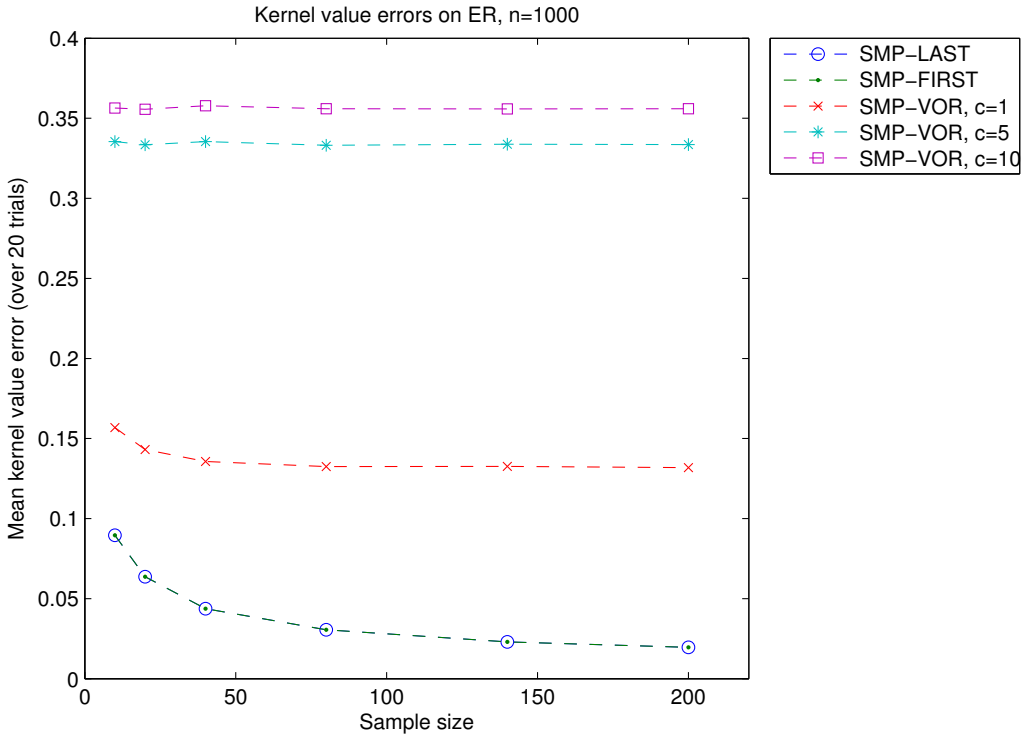

Figure 8.6: Kernel value errors for the sampling-based kernels on ER1000.

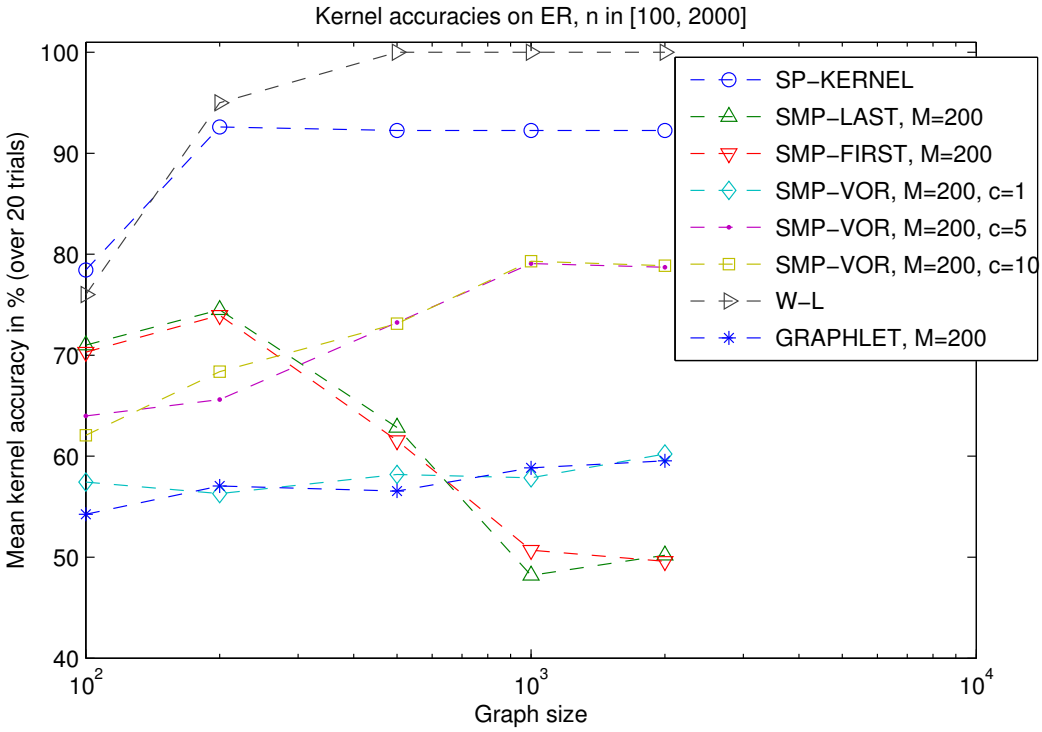

Figure 8.7: Classification accuracies for all kernel variants on ER.

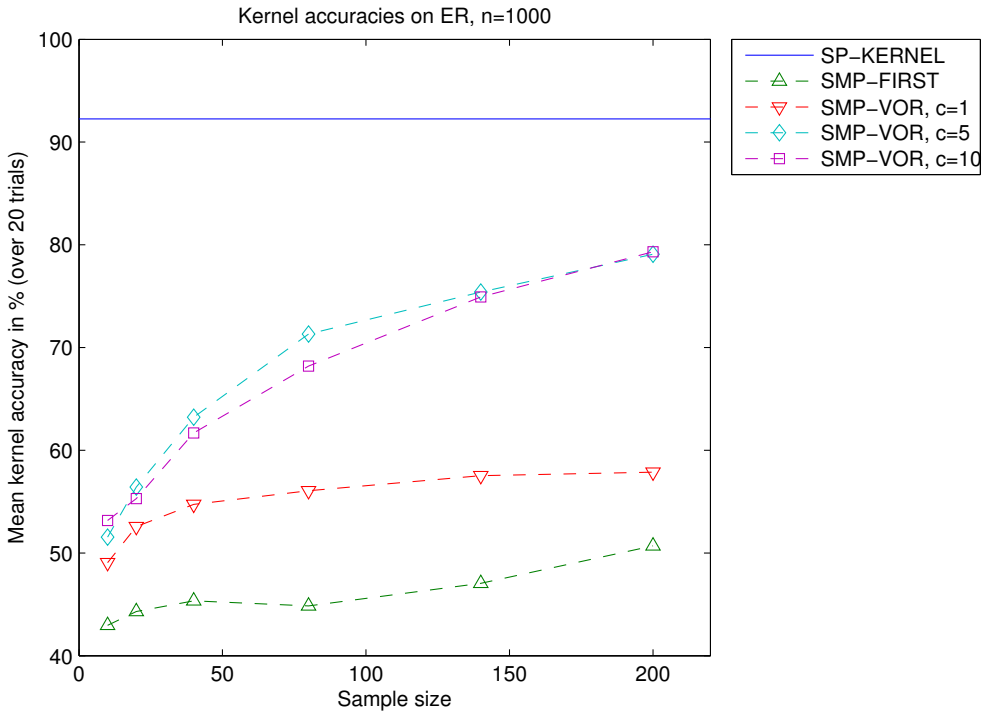

Figure 8.8: Classification accuracies for the sampling-based kernels on ER1000.

Accuracy

Figure 8.7 shows the classification accuracy of the different kernel variants. Again, SMP-LAST and SMP-FIRST are very similar. Unless otherwise noted this pattern continues for the other datasets as well: in kernel value error and classification accuracy, SMP-LAST and SMP-FIRST seem to be equivalent.

On this data, none of the approximate variants perform very well. Excepting the smallest graph set, the original SP-KERNEL maintains a stable accuracy of about 92%, while the best result among the approximations is that of SMP-VOR for $c = 5$ and $c = 10$. The choice of $c = 1$—which achieves a notably lower kernel value error—surprisingly yields much worse results; less than 60% compared to the about 80% (for the larger graph sizes). The plain sampling of SMP-FIRST does better for the two smallest size groups $n = 100$ and $n = 200$, but drops for the larger graphs, even falling below 50% at worst!

Finally, fig. 8.8 shows how the accuracy varies with the number of samples M. This data agrees with the surprising difference between c-values for SMP-VOR. As M increases, the $c = 1$ instance levels off below 60% accuracy. Meanwhile, $c = 5$ and $c = 10$ whose kernel value error was consistently larger (fig. 8.6), exhibit the desired trend of accuracy increasing as M does. The corresponding figures for other values of n are very similar, with the difference that SMP-FIRST performs better (as indicated by fig. 8.7).

8.2 ROADS graphs

The ROADS dataset consists of 100 graphs each of sizes 100, 200, 500, 1000, and 2000 nodes. Compared to the other datasets, these graphs have comparatively low degree.

Runtimes and operation counts

Figure 8.9 shows the query-time runtimes, which look very similar to those of the ER data. On this data, SMP-VOR has a greater advantage over SP-KERNEL and SMP-FIRST. Looking at the total runtimes shown in fig. 8.10, we see that SMP-VOR improves greatly over SP-KERNEL, but SMP-FIRST is yet another leap ahead.

GRAPHLET maintains its dominance runtime-wise, but on this data W-L is not better than all the shortest path variants: SMP-FIRST shows similar behaviour to W-L as n grows, and runs faster in our implementation.

However, looking again at the operation counts inside calls to Dijkstra's algorithm (fig. 8.11), we see that SMP-VOR has the advantage there. On this data, we also see a slight trend suggesting that $c = 10$ is a better choice than 5 or 1 when it comes to how the runtime scales with larger n. The parameter c determines the number of Voronoi nodes, and a plausible explanation is that more Voronoi nodes leads to smaller Voronoi sleeves, which means fewer nodes to be searched in the refinement step of SMP-VOR (see section 6.3).

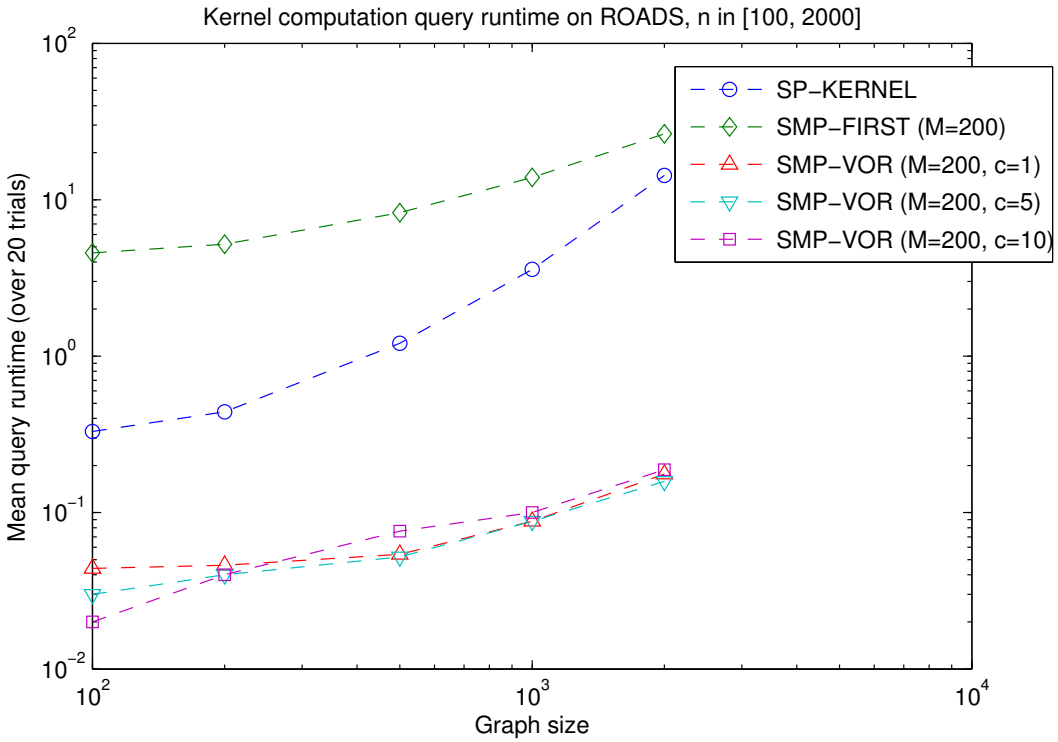

Figure 8.9: Query-time runtimes for all kernel variants on ROADS.

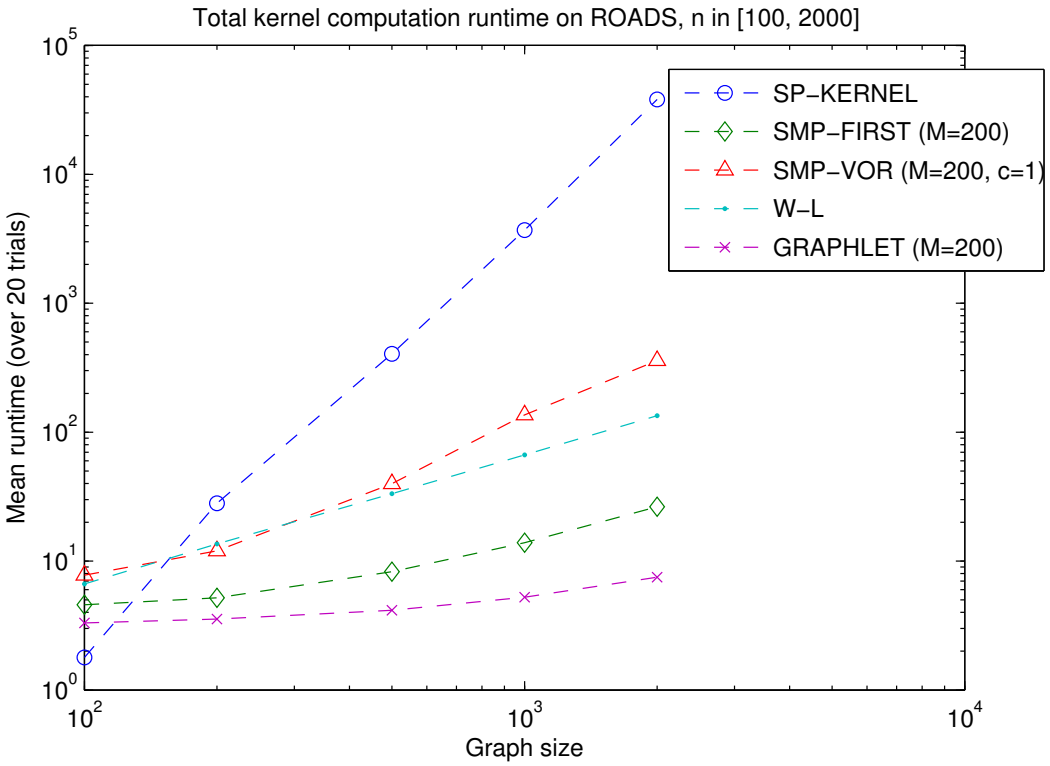

Figure 8.10: Total runtimes for all kernel variants on ROADS.

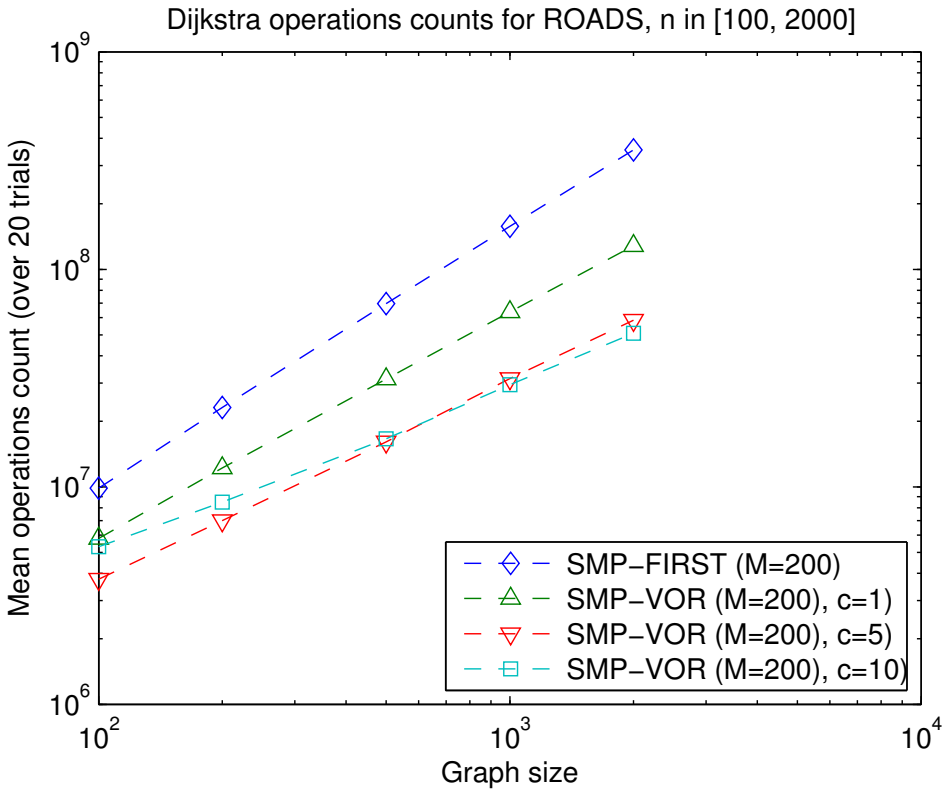

Figure 8.11: Operation counts (binary-heap exchange operations) for SMP-VOR and SMP-FIRST on ROADS.

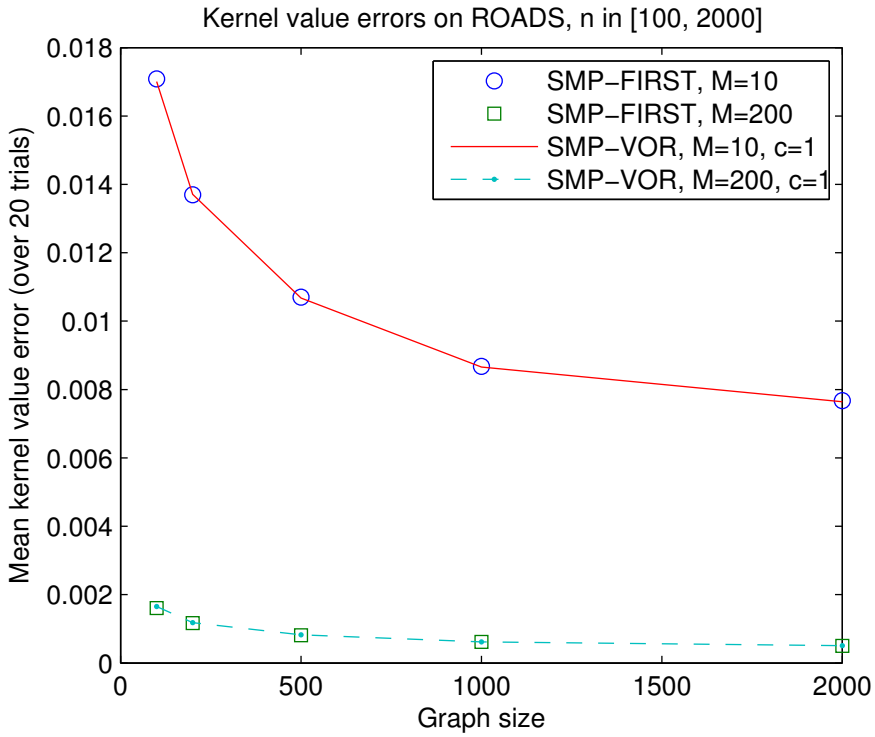

Figure 8.12: Kernel value errors, defined as $|\hat{k}_{\text{sp-}\delta} - k_{\text{sp-}\delta}|$, for the sampling-based kernels on ROADS.

Error and accuracy

As shown in fig. 8.13, the shortest-path kernel was not very well suited to the ROADS data at all, and neither was the graphlet kernel. The only kernel achieving accuracy above 50% was the Weisfeiler-Lehman kernel as computed by W-L. Even W-L did not perform particularly well, however, as its accuracy drops sharply in the larger end of the graph size spectrum.

Because of this, the accuracy results on ROADS are not very relevant. However, as will be elaborated in section 9.3, an underlying assumption of this research is that the shortest-path kernel, while not perfect for every type of graph data, is useful in some contexts. The fact that ROADS appears to be of the former category does not in itself challenge this assumption.

8.3 PROTO graphs

The PROTO dataset consists of 200 graphs each of sizes 100, 150, 250, 400, 650, and 1000 nodes.

Runtimes

The query-time runtimes show SMP-FIRST to be slower than SP-KERNEL, as with the other data sets. Like with ROADS, SMP-VOR provides a consistently faster query-time than even SP-KERNEL. The total runtimes for PROTO do not show anything new; they are very similar to those of ROADS. They are not shown here, but are still included in table 2, in section 9.3.

Error and accuracy

Figure 8.15 shows that for the PROTO data, the error of SMP-VOR increases with graph size. The same trend is visible for SMP-FIRST as well, but on a much smaller scale. This contrasts to the previous datasets, where errors either decreased monotonously (fig. 8.12) or grew to a maximum and then decreased (fig. 8.5). One possibility is that PROTO shares the same trait that caused the error to grow to a maximum for the ER data, but this maximum falls after $n = 1000$ which is the largest graph size used in the PROTO tests.

We can also see another effect: that of the number of samples M. It is clearer in fig. 8.16, showing several values of M on the size group of $n = 1000$. The behaviour is the expected: taking fewer samples results in a larger error.

The classification accuracies of the different kernel variants are shown in fig. 8.17. On this dataset, the standard kernel SP-KERNEL has an accuracy of 85% on graphs of $n = 100$, and better for larger graphs. The best of our approximations turns out to be $k_{\text{sp-sampled}}$ (represented by SMP-FIRST). W-L varies but maintains a general high accuracy; GRAPHLET meanwhile is comparable to the other kernels only for the smallest graphs.

In contrast to the results from the ER data, SMP-VOR here achieves similar results for all three values of c. Surprisingly, $c = 1$ results in the best accuracy, even though it was the worst classifier on ER, as seen in fig. 8.7.

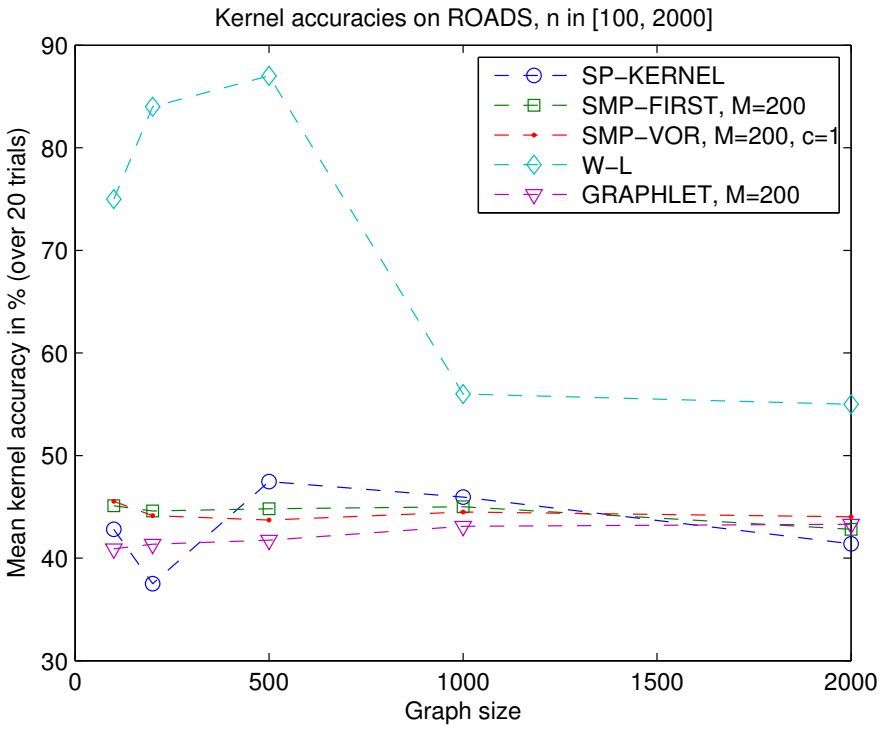

Figure 8.13: Classification accuracies for all kernel variants on ROADS.

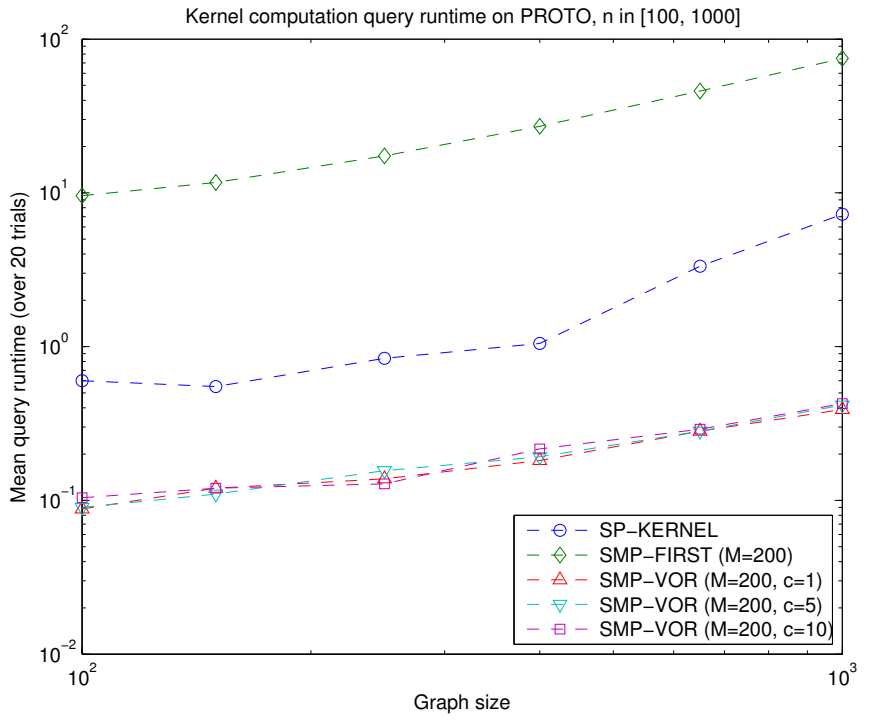

Figure 8.14: Query-time runtimes for all kernel variants on PROTO.

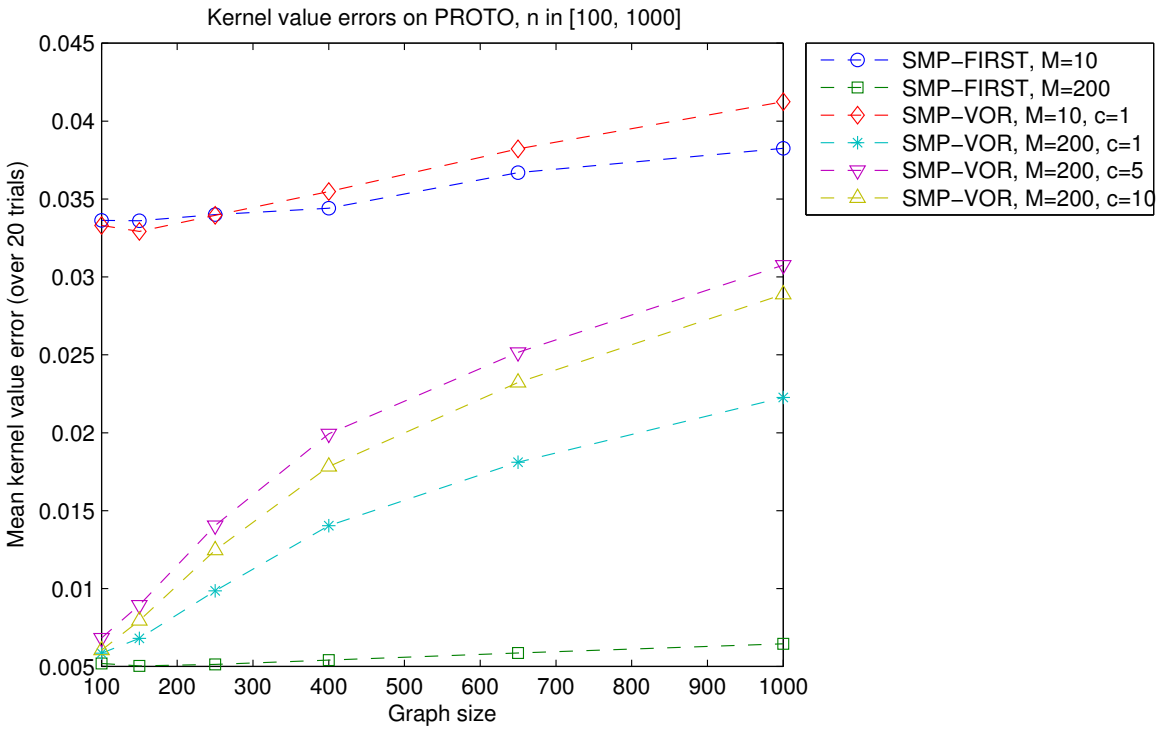

Figure 8.15: Kernel value errors, defined as $|\hat{k}_{\text{sp-}\delta} - k_{\text{sp-}\delta}|$, for the sampling-based kernels on PROTO.

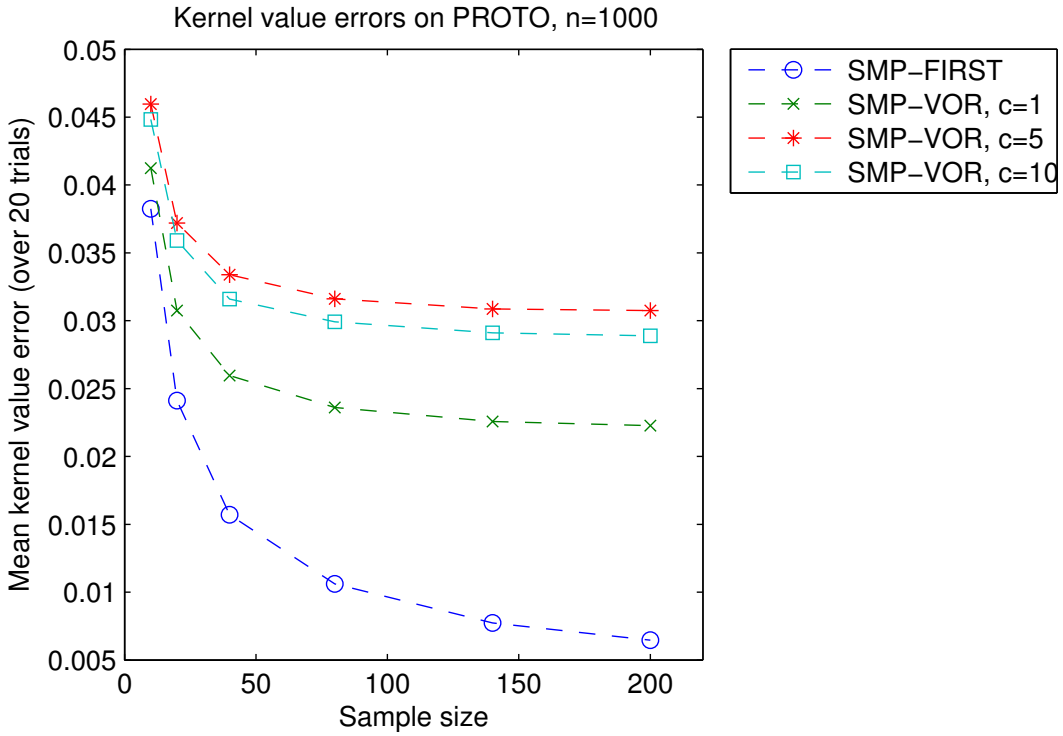

Figure 8.16: Kernel value errors for the sampling-based kernels on PROTO1000.

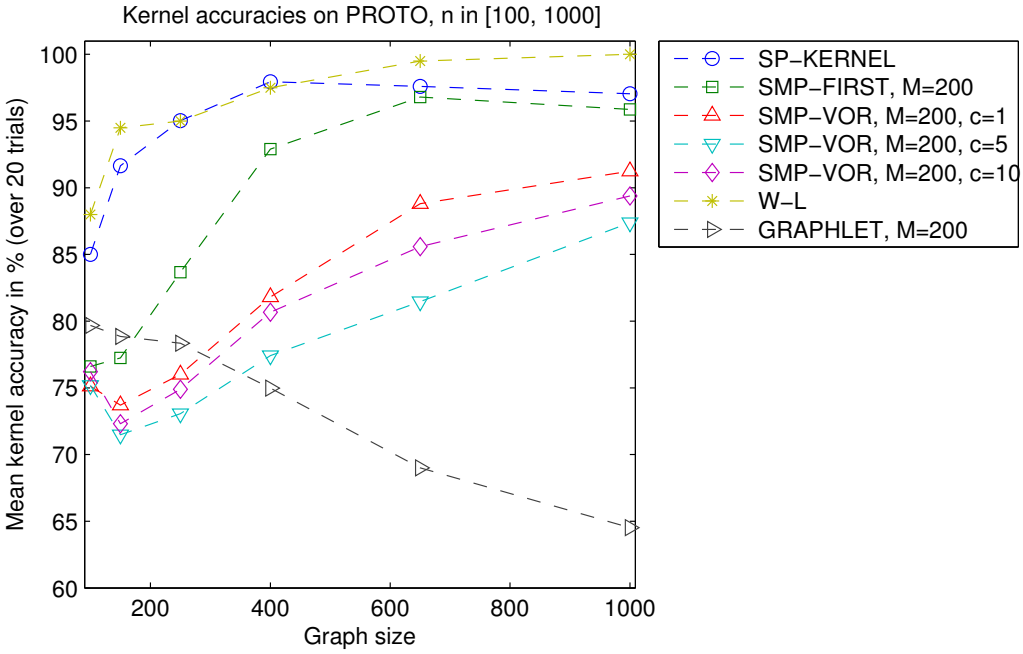

Figure 8.17: Classification accuracies for all kernel variants on PROTO.

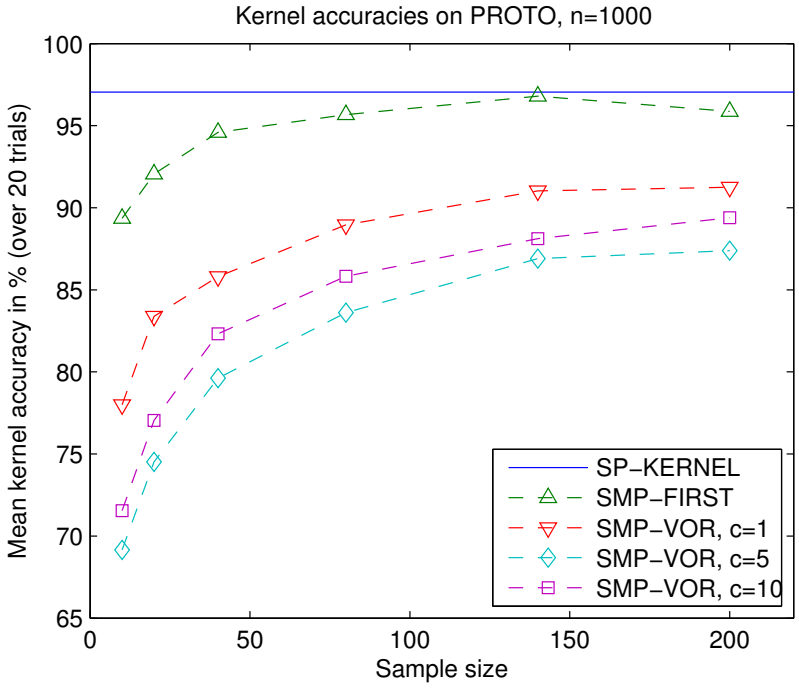

Figure 8.18: Classification accuracies for the sampling-based kernels on PROTO1000.

Finally, fig. 8.18 shows the classification accuracy over different values of M. While SMP-FIRST at its best is so close to the full kernel that one should not expect it to improve further by increasing M, SMP-VOR shows an interesting behaviour. It also seems to plateau between $M = 140$ and $M = 200$ sampled node pairs. It seems unable to reach the same accuracy as SP-KERNEL on this data, even for large values of M.

8.4 GAUSS graphs

The ER dataset consists of 100 graphs each of sizes 100, 200, 500, 1000, and 2000 nodes. Within each size group, 50 of the graphs are generated by the GAUSS model (see section 7.3) with parameter $p = 0.1$, and for the remaining 50 the value was $p = 0.16$. The threshold distance was set to $d_t = 0.5$ and the increment $d_i = 0.1$ for both classes. This means that most edges in the graphs had integer weights between 1 and 5; with a few outliers with higher-cost edges.

Runtimes

The runtime data from GAUSS does not reveal anything new. They are included in table 2, in section 9.3.

Error

Figure 8.19 confirms that we have no good model for the behaviour of the error as n increases. Equation (5.6) gives us an upper bound, but other than conforming to that bound the error values behave very differently on different datasets. On ER, SMP-LAST and SMP-FIRST were virtually constant, while SMP-VOR grew to a maximum and then decreased (fig. 8.5). On ROADS, all kernels improved drastically as the graph size increased (fig. 8.12). On PROTO, SMP-VOR showed error values greatly increasing with n while SMP-FIRST only increased marginally (fig. 8.15). GAUSS seems to be similar to PROTO in this regard, except that SMP-VOR seems to level off around $n = 1000$.

However, regardless of what trends the error values exhibit, those values are in fact very small. Looking at the accuracy results in fig. 8.20, the GAUSS data seems to be one of the best settings for our approximations. Both SMP-VOR and SMP-FIRST achieve accuracies close to or even better than SP-KERNEL. Apparently the distribution of sampled (approximate) shortest-path lengths provides as good a classification basis as the complete (exact) distribution, for this data.

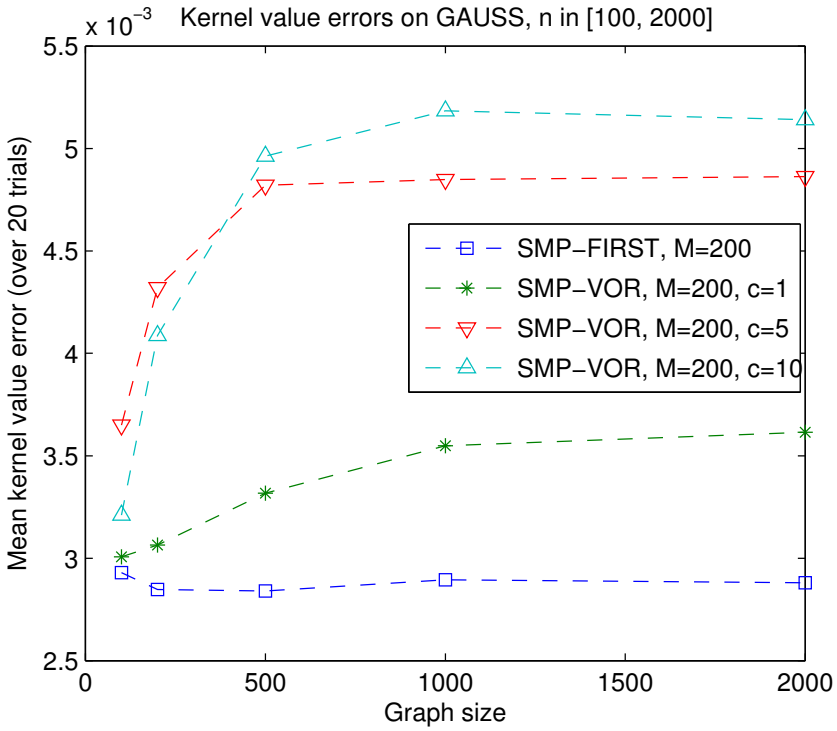

Figure 8.19: Kernel value errors, defined as $|\hat{k}_{\text{sp-}\delta} - k_{\text{sp-}\delta}|$, for the sampling-based kernels on GAUSS.

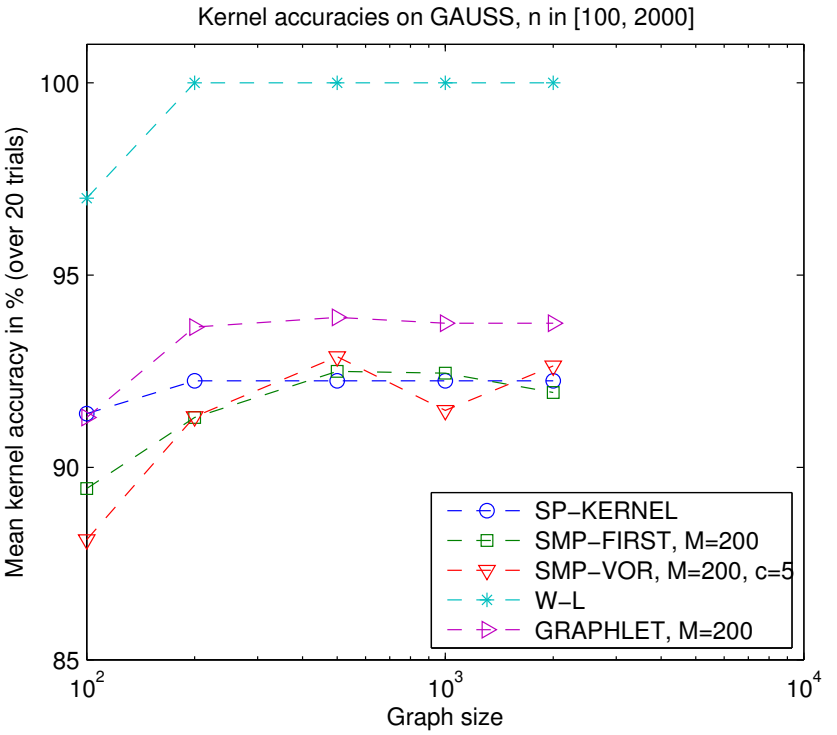

Figure 8.20: Classification accuracies for all kernel variants on GAUSS.

8.5 D&D graphs

The D&D dataset consists of one large set of 1157 graphs of varying sizes. The smallest graph has 30 nodes; the largest 4152; the median size is $\tilde{n} = 240$ nodes. 679 of these are classified as enzymes and the remaining 478 as non-enzymes. As previously mentioned, the original dataset from Dobson and Doig (2003) is larger, but we exclude 21 graphs because they are disconnected.

Runtimes

The runtime results on D&D are shown in fig. 8.21. SP-KERNEL (query and total) as well as W-L are displayed as straight lines – since they do not take the parameter M they are represented by a single value.

All kernels exhibit very similar behaviour on D&D as on the other datasets as far as runtime is concerned. SP-KERNEL is the slowest by far. SMP-FIRST is comparable to its query-time runtime at low sampling levels, its runtime increases as M does but even at $M = 200$ it is orders of magnitude from the total runtime of SP-KERNEL. Similarly to ROADS, SMP-VOR is slower than SMP-FIRST.

Error and accuracy

Figure 8.22 shows that the sampling-based kernel variants on D&D exhibit the same type of error behaviour as on PROTO and ROADS. The error values are low, and decrease drastically as M increases. Furthermore, SMP-FIRST and SMP-VOR yield values so close to each other as to be indistinguishable in the plot – the latter for all tried values of c.

Finally, fig. 8.23 shows the classification accuracy of all kernel variants on D&D. SMP-FIRST and SMP-VOR are again very close to each other; a more detailed view is found in fig. 8.24. All sampling-based kernels surprisingly stay almost constant when M varies, despite the decreasing kernel value error seen in fig. 8.22. However, they hover just below the level of SP-KERNEL, which achieves the best result on this data. The accuracy of GRAPHLET increases as M does, but it seems to level off. How much larger a value of M would be required for GRAPHLET to overtake SMP-FIRST is hard to say, though it seems a distinct possibility.

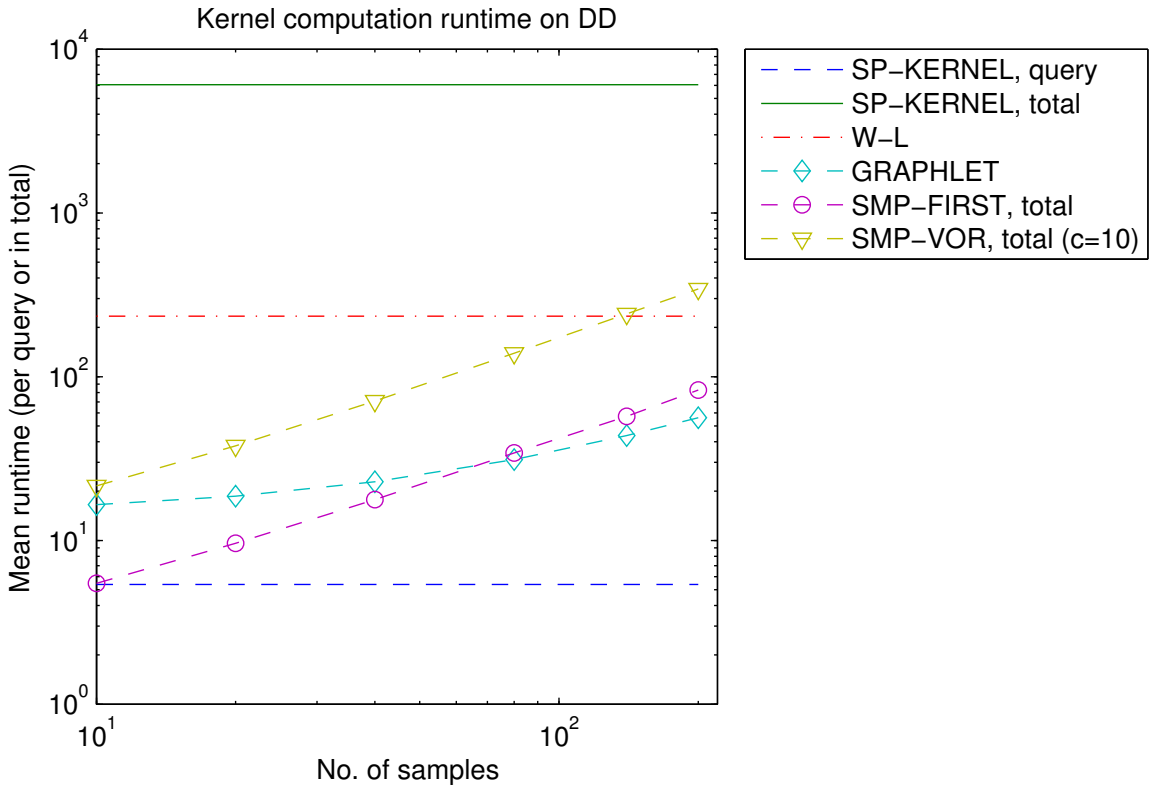

Figure 8.21: Runtimes for all kernel variants on D&D.

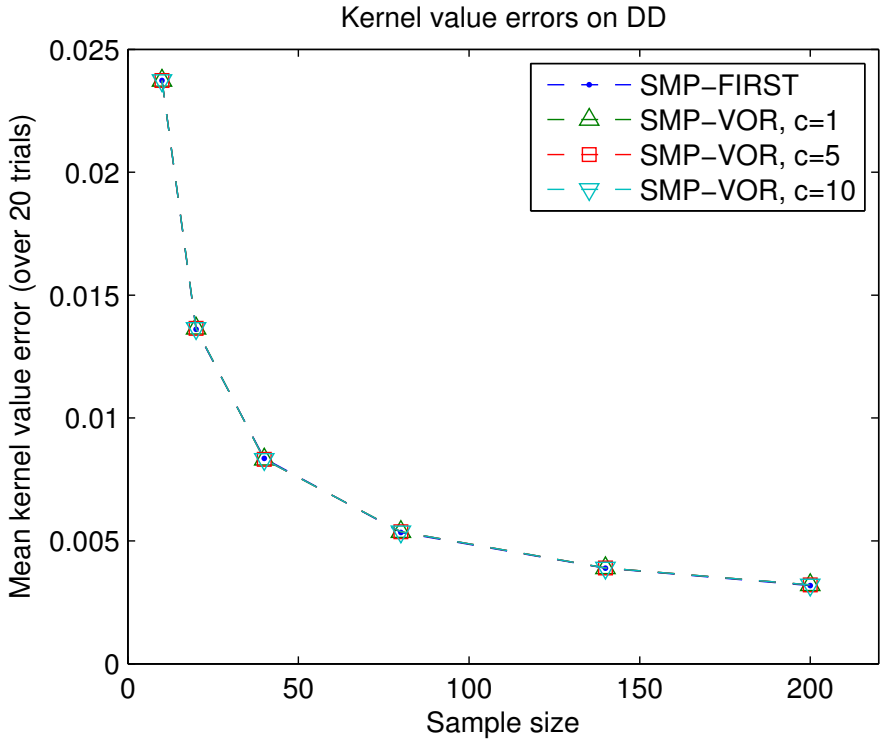

Figure 8.22: Kernel value error, defined as $|\hat{k}_{\text{sp-}\delta} - k_{\text{sp-}\delta}|$, for SMP-FIRST on D&D. All lines are, at this resolution, identical.

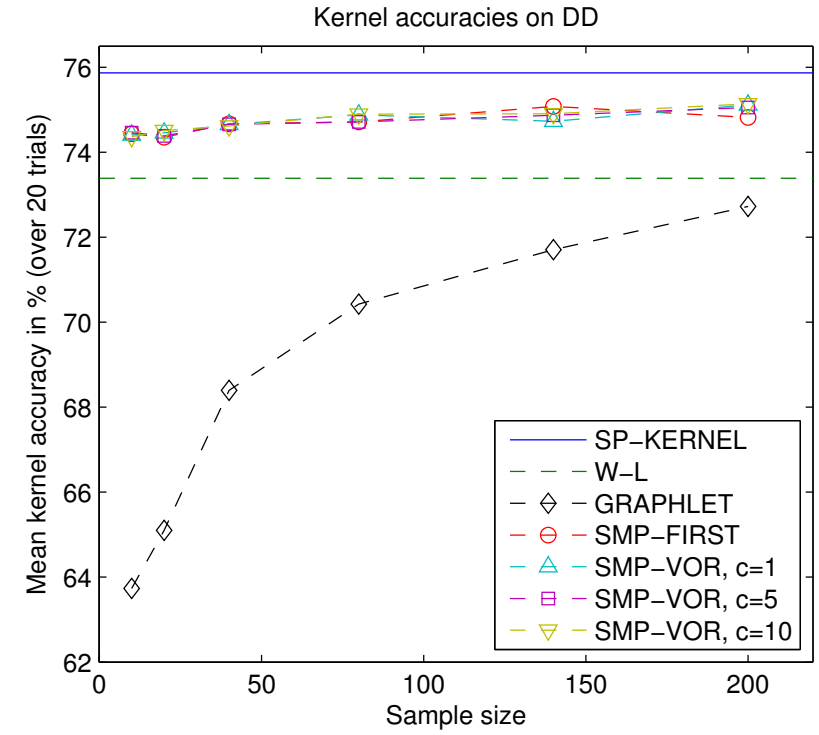

Figure 8.23: Classification accuracies for all kernel variants on D&D.

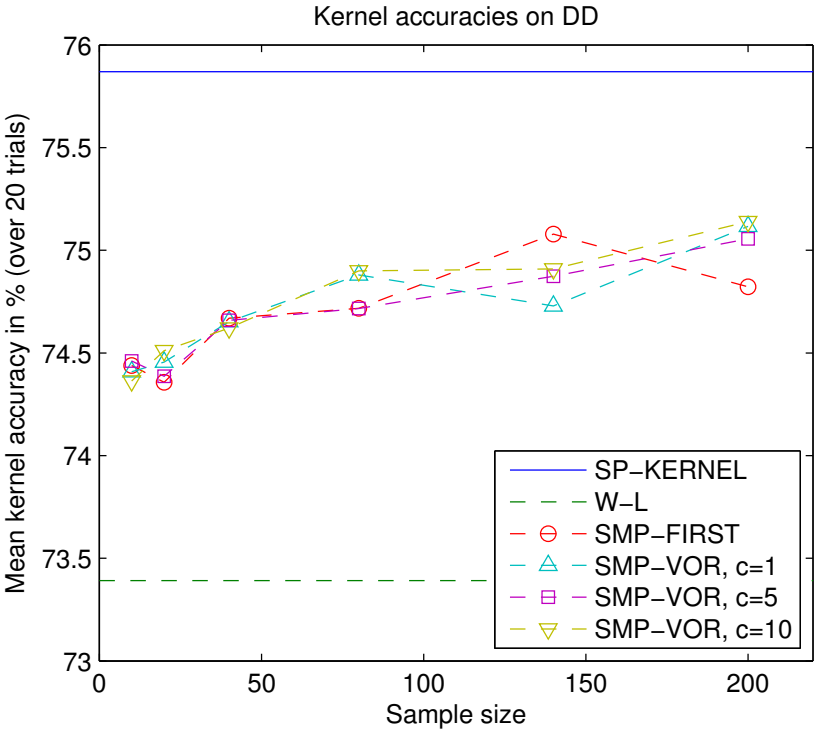

Figure 8.24: Classification accuracies for the sampling kernel variants on D&D (closeup of fig. 8.23).

Chapter 9

Conclusions

In general, we find our results supporting basic claims about the approximations, while being inconclusive about stronger ones. Those basic claims are the following:

Sampling can substantially improve runtime There is no question about the fact that computing every shortest-path distance, as SP-KERNEL does, is orders of magnitude slower than computing only a few such distances like SMP-FIRST. If this approximation performs well on a certain type of data, it is well worth using.

More samples means better accuracy On every dataset, running SMP-FIRST or SMP-VOR with a larger number of samples M improves accuracy. This behaviour is present even for SMP-VOR on those datasets where its kernel value error actually increases as M increases. Our explanation is that the distribution φ_{Vor} of Voronoi-dual-generated approximate distances also encodes information about the graph which is useful for classification. Running SMP-VOR with a large M means approximating that distribution well, even if there is a substantial discrepancy from the "pure" shortest-path-length distribution φ.

Some very large graphs are a better fit Seen most clearly on PROTO (see fig. 8.17), the kernel approximations often achieve better accuracy for larger graphs. There are some exceptions, but it is clear that there exists types of data where, the larger the graphs, the better performance our approximations will attain. Since the runtime improvement is leveraged the most on large graphs, this type of data seems like a perfect fit for the approximate shortest-path kernel. However, we cannot from our limited testing deduce exactly what type of data exhibits this effect; only that PROTO seems to be an example of it.

These three are our main conclusions. The rest of this chapter provides some further discussion on various implications of the results.

9.1 SMP-FIRST and SMP-LAST

The two sampling algorithms SMP-FIRST and SMP-LAST are mathematically identical; if implemented correctly they ought to give the same results. Our results confirm this; no significant difference was seen in kernel value error or classification accuracy. However, their runtimes differ wildly; also predicted by the theory.

SMP-LAST requires the same amount of preprocessing work as the kernel it aims to approximate (i.e. $k_{\text{sp-}\delta}$ by way of SP-KERNEL); further, this work completely dwarfs the small advantage SMP-LAST has over SP-KERNEL at query-time. Our conclusion is that SMP-LAST is of no use to anyone. If its lower accuracy is acceptable, SMP-FIRST achieves the same result faster; if the high runtime is affordable, SP-KERNEL might as well be used instead.

9.2 The Voronoi approach

In the original context of Honiden et al. (2010), the graph Voronoi dual method of computing shortest-path queries is motivated by the search for an algorithm which achieves good query-time speed at the cost of moderate preprocessing costs. Our testing confirms that this is its role, placing SMP-VOR squarely between the all-preprocessing SP-KERNEL and the query-only SMP-FIRST.

However, we believe that SMP-VOR has the potential to be unambiguously faster than SP-KERNEL when using the same number of samples m, and that this advantage would be clear if more work were put into optimising the implementations of both. Supporting this claim is the fact that the number of operations required inside calls to Dijkstra's algorithm—the backbone of both methods—shows a clear advantage for SMP-VOR. A possible explanation for the occasional longer runtimes of SMP-VOR is that where SMP-FIRST requires one large call to Dijkstra's algorithm per sampled node pair, SMP-VOR uses two smaller ones, plus an additional one per graph during preprocessing. We believe that the overhead cost of calling the Dijkstra sub-routine could be reduced, and this would benefit SMP-VOR in the comparison.

Regardless of the truth of the above claim, SMP-VOR definitely fulfills the goal of finding a trade-off between query-time and preprocessing computation time. In the simple classification problem setting our testing considers, this goal is not very important in itself. What our tests do not show is how the algorithms scale in terms of the number of graphs N. Perhaps, for datasets that are large not only in terms of n but also in terms of N, SMP-VOR could push its advantage. This is a possible subject of future research.

The previous conclusions only concern runtimes, not error or accuracy. Over the different datasets, SMP-VOR and SMP-FIRST differ, with no clear overall winner. This gives us reason to believe that both algorithms can be useful in some settings, even if neither is applicable to every problem.

9.3 Comparison to other kernels

Our testing also included two kernels from other families than the shortest-path kernel; W-L and GRAPHLET. The Weisfeiler-Lehman subtree kernel (represented by W-L) in particular achieved better accuracy than SP-KERNEL on many datasets. At the same time, its runtime was comparable to that of our approximations. Does this mean our work is wholly irrelevant?

With the large selection of graph kernels available to today's machine learning programmer, it is obvious to us that there is no "silver bullet" kernel which is the optimal choice in every situation. An underlying assumption of this thesis is that there are real-life problems or research questions where the shortest-path kernel is a good candidate for a graph-classification kernel. If this assumption were to be completely disproved, this thesis would be rendered obsolete. To our knowledge, however, no such conclusive research exists. Neither do we consider this thesis in itself to warrant that conclusion. While W-L generally performs as good as SP-KERNEL or better in our tests, those tests are not numerous or exhaustive enough to support a categorical dismissal of the shortest-path kernel.

With that said, there is an interesting comparison to be made to the other reference kernel algorithm included, i.e. GRAPHLET, computing the graphlet kernel of order 3. Like SMP-FIRST, it is based on sampling substructures of the graph to approximate their distribution – GRAPHLET samples graphlets, while SMP-FIRST samples shortest-path lengths. GRAPHLET generally achieves the best runtime of all kernels tested, but its accuracy, especially for very large graphs, is not very impressive.

One possible explanation is that large graphs, unless they are also very dense, will have a very high proportion of a particular graphlet: the one with no edges at all. After all, when sampling 3 nodes out of 2000, what are the chances of finding nodes connected to each other? For many types of graphs, those chances are very low. This effect could skew the sampled graphlet distribution towards putting all probability on that one graphlet, which would reduce the amount of information contained in the kernel value. In contrast, the lengths of randomly sampled shortest paths seem to encode more information about the nature of the graph, even for large graphs.

A: Tables of results

	PROTO					
$n =$	100	150	250	400	650	1000
SP-KERNEL	85.0	91.65	95.025	97.95	97.6	97.05
SMP-FIRST*	76.6	77.25	83.675	92.9	96.8	95.875
SMP-LAST*	76.925	77.8	84.725	92.925	96.875	96.5
SMP-VOR$^*_1$	75.12	73.71	76.01	81.81	88.81	91.25
W-L	88.0	94.5	95.0	97.5	99.5	100
GRAPHLET*	79.675	78.85	78.35	74.975	69.0	64.525

	ER				
$n =$	100	200	500	1000	2000
SP-KERNEL	78.45	92.6	92.25	92.25	92.25
SMP-FIRST*	70.3	73.95	61.55	50.7	49.6
SMP-LAST*	71.0	74.5	62.85	48.2	50.2
SMP-VOR$^*_{10}$	62.08	68.38	73.12	79.32	78.86
W-L	76	95	100	100	100
GRAPHLET*	54.25	57.05	56.55	58.85	59.55

	ROADS				
$n =$	100	200	500	1000	2000
SP-KERNEL	42.8	37.5	47.45	45.95	41.4
SMP-FIRST*	45.1	44.6	44.8	45.0	42.8
SMP-LAST*	46.8	43.75	41.75	44.25	42.1
SMP-VOR$^*_1$	45.54	44.14	43.7	44.5	44.02
W-L	75	84	87	56	55
GRAPHLET*	40.9	41.35	41.75	43.1	43.3

	GAUSS				
$n =$	100	200	500	1000	2000
SP-KERNEL	91.4	92.25	92.25	92.25	92.25
SMP-FIRST*	89.45	91.3	92.5	92.45	91.95
SMP-LAST*	90.55	91.85	93.25	93.65	93.7
SMP-VOR$^*_5$	88.12	91.32	92.88	91.48	92.64
W-L	97.0	100	100	100	100
GRAPHLET*	91.3	93.65	93.9	93.75	93.75

All numbers are % of graphs correctly classified.
* with $M = 200$
$_h$ with $c = h$

Table 1: Prediction accuracies on all graph classification tasks, on all datasets except D&D.

	PROTO					
$n =$	100	150	250	400	650	1000
SP-KERNEL	3.6″	10.5″	1′47″	6′25″	35′23″	2h9′57″
SMP-FIRST*	9.6″	11.7″	17.4″	27.1″	45.9″	1′15″
SMP-LAST*	3.2″	10.3″	1′47″	6′26″	35′23″	2h9′56″
SMP-VOR$^*_1$	14.8″	17.2″	33.2″	55.3″	2′43″	4′54″
W-L	14.4″	21.8″	35.4″	57″	1′33″	2′21″
GRAPHLET*	6.8″	7.2″	7.6″	8.1″	9.2″	10.8″

	ER				
$n =$	100	200	500	1000	2000
SP-KERNEL	1.95″	31.5″	7′57″	1h8′18″	9h6′25″
SMP-FIRST*	6.6″	16.1″	1′34″	7′	29′52″
SMP-LAST*	1.7″	30.5″	7′57″	1h8′18″	9h6′20″
SMP-VOR$^*_{10}$	7.99″	10.2″	43.5″	2′9″	9′54″
W-L	7.6″	14.8″	37.7″	1′16″	2′36″
GRAPHLET*	3.3″	3.6″	4.3″	5.4″	8.1″

	ROADS				
$n =$	100	200	500	1000	2000
SP-KERNEL	1.8″	28.1″	6′45″	1h1′31″	10h35′
SMP-FIRST*	4.6″	5.2″	8.3″	13.9″	26.4″
SMP-LAST*	1.6″	28.1″	6′45″	1h1′31″	10h34′57″
SMP-VOR$^*_1$	7.8″	12″	39.8″	2′16″	6′
W-L	6.7″	13.6″	33.4″	1′7″	2′14″
GRAPHLET*	3.3″	3.5″	4.2″	5.3″	7.5″

	GAUSS				
$n =$	100	200	500	1000	2000
SP-KERNEL	1.9″	34.6″	7′24″	1h1′4″	8h12′57″
SMP-FIRST*	17.6″	1′8″	7′25″	31′5″	2h23′11″
SMP-LAST*	1.7″	34.5″	7′24″	1h1′2″	8h12′52″
SMP-VOR$^*_5$	7.9″	10.7″	39.7″	2′25″	6′53″
W-L	7.6″	16.1″	47.1″	2′	5′3″
GRAPHLET*	3.3″	3.8″	4.5″	6.2″	9.5″

* with $M = 200$

$_h$ with $c = h$

Table 2: Runtimes for all datasets except D&D.

	PROTO					
$n =$	100	150	250	400	650	1000
SMP-FIRST*	0.0052	0.005	0.0051	0.0054	0.0059	0.0065
SMP-LAST*	0.0052	0.005	0.0051	0.0054	0.0059	0.0065
SMP-VOR$^*_1$	0.0059	0.0068	0.0099	0.014	0.0181	0.0223
SMP-VOR$^*_5$	0.0068	0.0089	0.0140	0.0199	0.0252	0.0308
SMP-VOR$^*_{10}$	0.0060	0.0079	0.0125	0.0178	0.0232	0.0289
	ER					
$n =$	100	200	500	1000	2000	
SMP-FIRST*	0.0151	0.0221	0.0199	0.0196	0.0197	
SMP-LAST*	0.0151	0.0221	0.0199	0.0196	0.0197	
SMP-VOR$^*_1$	0.0742	0.2084	0.2425	0.1318	0.0484	
SMP-VOR$^*_5$	0.1608	0.2959	0.4009	0.3336	0.1854	
SMP-VOR$^*_{10}$	0.1511	0.3007	0.4078	0.3560	0.2064	
	ROADS					
$n =$	100	200	500	1000	2000	
SMP-FIRST*	0.0016	0.0012	0.0008	0.0006	0.0005	
SMP-LAST*	0.0016	0.0012	0.0008	0.0006	0.0005	
SMP-VOR$^*_1$	0.0017	0.0012	0.0008	0.0006	0.0005	
SMP-VOR$^*_5$	0.0018	0.0012	0.0009	0.0006	0.0005	
SMP-VOR$^*_{10}$	0.0018	0.0013	0.0009	0.0006	0.0005	
	GAUSS					
$n =$	100	200	500	1000	2000	
SMP-FIRST*	0.0029	0.0028	0.0028	0.0029	0.0029	
SMP-LAST*	0.0029	0.0028	0.0028	0.0029	0.0029	
SMP-VOR$^*_1$	0.003	0.0031	0.0033	0.0035	0.0036	
SMP-VOR$^*_5$	0.0036	0.0043	0.0048	0.0048	0.0049	
SMP-VOR$^*_{10}$	0.0032	0.0041	0.0050	0.0052	0.0051	

* with $M = 200$

$_h$ with $c = h$

Table 3: Kernel value errors, defined as $|\hat{k}_{\text{sp-}\delta} - k_{\text{sp-}\delta}|$, for the approximate shortest-path kernels, on all datasets except D&D.

	D&D			
	Query runtime	Total runtime	Error	Accuracy (%)
SP-KERNEL	$5.4''$	$1h40'57''$	0	75.87
SMP-FIRST*	$1'23''$	$1'23''$	0.0032	74.82
SMP-LAST*	$3.9''$	$1h40'55''$	0.0032	74.55
SMP-VOR$^*_1$	$5'57''$	$6'0''$	0.0032	75.12
SMP-VOR$^*_5$	$5'54''$	$5'56''$	0.0032	75.06
SMP-VOR$^*_{10}$	$5'41''$	$5'43''$	0.0032	75.14
W-L	$3'54''$	$3'54''$	N/A	71.21
GRAPHLET*	$56.1''$	$56.1''$	N/A	72.80

* with $M = 200$

$_h$ with $c = h$

Table 4: Runtimes, kernel value errors and accuracies for all kernels on D&D.

Bibliography

Aronszajn, N. (1950). "Theory of Reproducing Kernels". In: *Transactions of the American Mathematical Society* 68.3, pp. 337–404. ISSN: 00029947.

Aurenhammer, Franz (1991). "Voronoi Diagrams—a Survey of a Fundamental Geometric Data Structure". In: *ACM Comput. Surv.* 23.3, pp. 345–405. ISSN: 0360-0300. DOI: 10.1145/116873.116880.

Borgwardt, Karsten M. and Hans-Peter Kriegel (2005). "Shortest-Path Kernels on Graphs". In: *Proceedings of the Fifth IEEE International Conference on Data Mining*. ICDM '05. Washington, DC, USA: IEEE Computer Society, pp. 74–81. ISBN: 0-7695-2278-5. DOI: 10.1109/ICDM.2005.132.

Borgwardt, Karsten M. and Nino Shervashidze (2009). "Fast subtree kernels on graphs". In: *Neural Information Processing Systems Conference*.

Borgwardt, Karsten M. et al. (2006). "Integrating structured biological data by Kernel Maximum Mean Discrepancy". In: *Bioinformatics* 22.14, e49–e57. DOI: 10.1093/bioinformatics/btl242.

Bui, Ngot and Vasant Honavar (2014). "Labeling Actors in Social Networks Using a Heterogeneous Graph Kernel". English. In: *Social Computing, Behavioral-Cultural Modeling and Prediction*. Ed. by WilliamG. Kennedy, Nitin Agarwal, and ShanchiehJay Yang. Vol. 8393. Lecture Notes in Computer Science. Springer International Publishing, pp. 27–34. ISBN: 978-3-319-05578-7. DOI: 10.1007/978-3-319-05579-4_4.

Chang, Yin-Wen et al. (2010). "Training and Testing Low-degree Polynomial Data Mappings via Linear SVM". In: *J. Mach. Learn. Res.* 11, pp. 1471–1490. ISSN: 1532-4435.

Cormen, Thomas H. et al. (2001). *Introduction to algorithms*. Cambridge, Mass: MIT Press, pp. 595–601. ISBN: 0262032937.

Dijkstra, Edsger W. (1959). "A note on two problems in connexion with graphs". English. In: *Numerische Mathematik* 1.1, pp. 269–271. ISSN: 0029-599X. DOI: 10.1007/BF01386390.

Dobson, P. D. and A. J. Doig (2003). "Distinguishing enzyme structures from non-enzymes without alignments". In: *J. Mol. Biol.* 330.4, pp. 771–783.

Erdős, Paul and Alfréd Rényi (1959). "On random graphs". In: *Publicationes Mathematicae Debrecen* 6, pp. 290–297.

Erwig, Martin (2000). "The graph Voronoi diagram with applications". In: *Networks* 36.3, pp. 156–163. ISSN: 1097-0037. DOI: 10.1002/1097-0037(200010)36:3<156::AID-NET2>3.0.CO;2-L.

Floyd, Robert W. (1962). "Algorithm 97: Shortest Path". In: *Commun. ACM* 5.6, pp. 345–. ISSN: 0001-0782. DOI: 10.1145/367766.368168.

Fredman, Michael L. and R.E. Tarjan (1984). "Fibonacci Heaps And Their Uses In Improved Network Optimization Algorithms". In: *Foundations of Computer Science, 1984. 25th Annual Symposium on*, pp. 338–346. DOI: 10.1109/SFCS.1984.715934.

Gärtner, Thomas, Peter Flach, and Stefan Wrobel (2003). "On Graph Kernels: Hardness Results and Efficient Alternatives". English. In: *Learning Theory and Kernel Machines*. Ed. by Bernhard Schölkopf and Manfred K. Warmuth. Vol. 2777. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 129–143. ISBN: 978-3-540-40720-1. DOI: 10.1007/978-3-540-45167-9_11. URL: http://dx.doi.org/10.1007/978-3-540-45167-9_11.

Gilbert, E. N. (1959). "Random Graphs". In: *The Annals of Mathematical Statistics* 30.4, pp. 1141–1144. DOI: 10.1214/aoms/1177706098.

Goddard, Wayne and Ortrud R. Oellermann (2011). "Distance in Graphs". English. In: *Structural Analysis of Complex Networks*. Ed. by Matthias Dehmer. Birkhäuser Boston, pp. 49–72. ISBN: 978-0-8176-4788-9. DOI: 10.1007/978-0-8176-4789-6_3.

Hart, P.E., N.J. Nilsson, and B. Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2, pp. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

Honiden, Shinichi et al. (2010). "Approximate Shortest Path Queries in Graphs Using Voronoi Duals". In: *Transactions on Computational Science* 9. Special Issue of the 6th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2009), pp. 28–53.

Hubler, C. et al. (2008). "Metropolis Algorithms for Representative Subgraph Sampling". In: *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, pp. 283–292. DOI: 10.1109/ICDM.2008.124.

Mehlhorn, Kurt and Peter Sanders (2008). *Algorithms and Data Structures: The Basic Toolbox*. 1st ed. Springer Publishing Company, Incorporated. ISBN: 9783540779773.

Schölkopf, Bernhard and Alexander J. Smola (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press. ISBN: 0262194759.

Shervashidze, Nino and Karsten M. Borgwardt (2012a). *Graph Kernels Code*. URL: http://mlcb.is.tuebingen.mpg.de/Mitarbeiter/Nino/Graphkernels/graphkernels.zip (visited on 01/07/2015).

— (2012b). *Graph Kernels Data*. URL: http://mlcb.is.tuebingen.mpg.de/Mitarbeiter/Nino/Graphkernels/data.zip (visited on 01/07/2015).

Shervashidze, Nino et al. (2009). "Efficient graphlet kernel for large graph comparison". In: *Artificial Intelligence and Statistics, Twelfth International Conference on*, pp. 488–495.

Shervashidze, Nino et al. (2011). "Weisfeiler-Lehman Graph Kernels". In: *J. Mach. Learn. Res.* 12, pp. 2539–2561. ISSN: 1532-4435.

Thorup, Mikkel (2003). "Integer Priority Queues with Decrease Key in Constant Time and the Single Source Shortest Paths Problem". In: *Proceedings of the Thirty-fifth Annual ACM Symposium on Theory of Computing*. STOC '03. San Diego, CA, USA: ACM, pp. 149–158. ISBN: 1-58113-674-9. DOI: 10.1145/780542.780566.

Warshall, Stephen (1962). "A Theorem on Boolean Matrices". In: *J. ACM* 9.1, pp. 11–12. ISSN: 0004-5411. DOI: 10.1145/321105.321107.

Weissman, Tsachy et al. (2003). "Inequalities for the L_1 Deviation of the Empirical Distribution". In: *Algorithms in Bioinformatics*.

Wiener, Harry (1947). "Structural Determination of Paraffin Boiling Points". In: *Journal of the American Chemical Society* 69.1, pp. 17–20. DOI: 10.1021/ja01193a005.

Zeng, W. and R. L. Church (2009). "Finding shortest paths on real road networks: the case for A^*". In: *International Journal of Geographical Information Science* 23.4, pp. 531–543. DOI: 10.1080/13658810801949850.