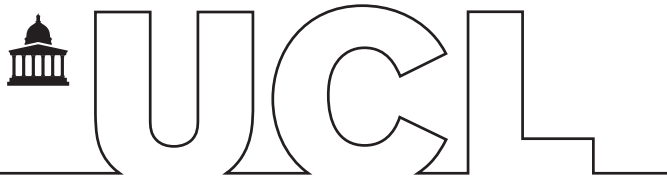

University College London

PHASGQ99 MRes Research Project

QUANTUM-INSPIRED ALGORITHMS FOR GRAPH MATCHING

Author:
Danial Dervovic

Supervisor:
Dr. Simone Severini

Submitted in partial fulfilment for the degree of **MRes Quantum Technologies**

27 August 2016

I, Danial Dervovic, confirm that the work presented in this report is my own. Where information has been derived from other sources, I confirm that this has been indicated in the report.

Signature

27-08-16

Date

Abstract

Graph isomorphism and graph similarity are important problems in computer science, with their efficient solution remaining an active area of research. Recently, the machine learning community has used kernel methods to efficiently solve a version of the graph similarity problem with some success, using graph kernels. Kernel methods are a tool in learning theory with which one can map non-linear patterns among arbitrary objects into linear patterns on vectorial data. It is in this domain that many pattern-matching procedures can then be successfully applied. One can define an object with respect to a given graph, called the average mixing matrix, which corresponds roughly to the limiting distribution of a continuous-time quantum walk on the graph. Here, we use the average mixing matrix to define a graph kernel, the Quantum Shannon kernel, which is competitive with state of the art classical methods in classification accuracy on the MUTAG bioinformatics database and runs in time $O(Nn^3b + N^2b)$, where N is the number of graphs, n is the maximum number of vertices and b is the resolution. We also present an algorithm using the average mixing matrix to refine the one-dimensional Weisfeiler-Lehman (WL) test for graph isomorphism, with runtime complexity $O(n^3hm)$, where m is the number of edges of the graphs being matched and h the number of iterations of the WL test. The algorithms are defined on unlabelled, undirected graphs. In addition, we comprehensively review quantum walks and kernel methods in pattern analysis as applied to graphs.

Acknowledgements

Thank you to my supervisor Simone Severini, for his guidance and fruitful discussions during this project. I would also like to thank my colleagues at the CDT in Delivering Quantum Technologies, and EPSRC for funding support.

Contents

Chapter 1

Introduction

Machine Learning algorithms already play a major role in our everyday lives, seeing applications as varied as spam mail filters, bioinformatics, financial risk evaluation and many more. These algorithms learn a relation between input and output data from examples, which are then applied to characterising new input data. With the ever-increasing proliferation of available data globally (currently of order several hundred exabytes [1] and growing roughly 20% year-on-year), machine learning algorithms need to remain computationally tractable in this so-called era of *big data*. Both in academia and industry, high-performance computing (HPC) resources are now required for the most cutting-edge learning applications, such as DeepMind's recent result [2] in training a machine to play Go at the international level. One area of machine learning research which has received considerable attention in recent years is adapting machine learning methods to tasks on graph or network data. It is learning on this type of data which this report is concerned with.

Given the importance of learning problems and how resource-hungry they are, a natural question to ask is whether an advantage can be gained by using quantum computation. Quantum computation gains its unique power over classical computation via an expanded state space ($\mathbb{C}^{2^n}$ as opposed to $\mathbb{Z}_2^n$, where n is the number of bits) and a wider variety of possible operations on this space (Unitary as opposed to Boolean). The most famous quantum algorithms are Shor's factoring algorithm [3] and Grover's search algorithm [4], giving an exponential and polynomial speedup respectively over the best known classical algorithms.

There has been considerable research interest in this area, coming from a multitude of different angles. The main directions are i) using gate-based or adiabatic quantum computing architectures for carrying out machine learning tasks, ii) using mathematical techniques from quantum information to improve classical learning algorithms and iii) using classical machine learning techniques for optimising quantum information protocols. Here, we are concerned with direction ii) where we use the *average mixing matrix*, a quantity derived from quantum information theory, in order to perform learning tasks on graph data.

We are interested chiefly in the problems of *graph isomorphism* and *graph similarity*. These problems are NP-intermediate; that is, they are known to be in the complexity class NP but not yet confirmed to be outside P, with $\mathsf{P} \subseteq \mathsf{NP}$. Recall that a computational problem is in P if it requires time and space resources polynomial in the size of the input to be solved. A problem

is in NP if *checking* a given solution is valid takes resources polynomial in the input. Finding the solution is not restricted to using polynomial resources and for an arbitrary NP problem requires time exponential in the size of the input. The current state of the art for solving graph isomorphism (exactly) is the recent quasi-polynomial[1] time algorithm by Babai [5]. The state of play for the graph similarity problem is more ambiguous. Since we are interested in pattern recognition applications, we are interested in *graph kernels*, to be discussed at length later. We note that the problems of *subgraph isomorphism* and *subgraph similarity*, where a graph is compared to an arbitrary subgraph of another graph, are NP-complete [6, 7]. By NP-complete we mean that they are the hardest problems in NP, thus any NP problem can be efficiently reduced to them, with 'efficiently' meaning in polynomial time.

This report begins in earnest with Chap. 2, comprising a review of graph theory and quantum walks. This includes some basic definitions, a summary of results on classical random walks and an exposition of the three main paradigms for quantum walks. These are the discrete-time, continuous time and scattering walk models. Two algorithms from the literature utilising quantum walks are also analysed.

In Chap. 3, we examine kernel methods, as applied to pattern analysis. We begin in a general sense, then include an example on vectorial data, the hard margin Support Vector Machine. We then discuss kernel methods on graphs, detailing two prominent graph kernels from the literature, the Weisfeiler-Lehman subtree kernel and the Random Walk kernel. We also examine the Quantum Jensen-Shannon kernel, which uses ideas from quantum information theory to construct a classical kernel, and the general difficulty of computing graph kernels.

In Chapter 4 we define some algorithms based on the average mixing matrix. These comprise algorithms for testing isomorphism between graphs, in addition to graph kernels. The distinguishing power of the isomorphism test is checked by seeing how many equivalence classes it partitions the full set of graphs up to eight vertices into. The graph kernels are tested by running an SVM classification task using the kernels. The results of these tests are discussed and complexity results for the algorithms proved.

Chap. 5 concludes the report with some future questions to be answered and some closing remarks.

[1] Quasi-polynomial means using resources which scale as $\exp(O(\text{poly}\log(\text{inputsize})))$.

Chapter 2

Graph Theory and Quantum Walks

In this chapter we survey classical random walks and three paradigms for quantum walks on graphs, along with important algorithmic applications.

2.1 Classical Random Walks on Graphs

A *graph* $G = (V, E)$ consists of a set V of n vertices and a set of edges $E \subseteq V \times V - \{\{v, v\} : v \in V\}$. Here we implicitly assume graphs are undirected. The *degree* of v, $d(v)$, is its number of neighbours. The *neighbourhood* of a vertex $\mathcal{N}(v)$ is the set of its adjacent vertices, i.e. $\mathcal{N}(v) = \{u : \{v, u\} \in E\}$. The *chromatic number* of a graph $\chi(G)$ is the minimum number of colours for which every pair of adjacent vertices has a different colour. The *adjacency matrix* of a graph, $A(G)$, is defined as

$$[A(G)]_{ij} = \begin{cases} 1, & \text{if } \{v_i, v_j\} \in E; \\ 0, & \text{otherwise.} \end{cases} \tag{2.1.1}$$

The adjacency matrix also has an un-normalised cousin, where each non-zero element is $d(i)$. We quote some definitions and known results about classical random walks [8]. A *classical random walk* on G is characterised by repeated application of a stochastic matrix M to an initial probability distribution over the vertices $\mathbf{p}_0$, where $M_{ij} = \frac{1}{d(v_i)}$ if $\{v_i, v_j\} \in E$ and $M_{ij} = 0$ otherwise. The distribution of the random walk after t timesteps is $\mathbf{p} = M^t \mathbf{p}_0$. If G is connected[1] and $\chi(G) \neq 2$, $\mathbf{p}^t$ converges to the (unique) *stationary distribution*, $\boldsymbol{\pi}$, given by $[\boldsymbol{\pi}]_i = d(v_i)/2|E|$.

Two frequently used quantities are the *mixing time* and *hitting time*. The mixing time, M_ϵ, is defined as

$$M_\epsilon = \min \left\{ t \,|\, \forall t' \geq t, \mathbf{p}_0 : \left\| M^t \mathbf{p}_0 - \boldsymbol{\pi} \right\| \leq \epsilon \right\}, \tag{2.1.2}$$

where $\|\cdot\|$ is some distance measure between distributions $\mathbf{p}$ and $\mathbf{q}$. One measure typically used

[1] i.e. there is a path moving only between adjacent vertices through G joining every $v \in V$ with every $u \in V$.

is the *total variation distance*:

$$\|\mathbf{p}-\mathbf{q}\| = \sum_i |\mathbf{p}_i - \mathbf{q}_i|. \tag{2.1.3}$$

The mixing time is related to the eigenvalues of M, namely *gap* between the largest ($\lambda_1 = 1$) and second largest (λ_2), in the following way:

$$\frac{\lambda_2}{(1-\lambda_2)\log 2\epsilon} \leq M_\epsilon \leq \frac{1}{(1-\lambda_2)}(\max_i \log[\boldsymbol{\pi}]_i^{-1} + \log \epsilon^{-1}). \tag{2.1.4}$$

The hitting time is the expected time taken for a random walk starting at v_i to arrive at v_j.

We have thus far assumed that a random walk can continue indefinitely. If we wish to incorporate stopping behaviour in a walk we can define a vector $\mathbf{q}$, such that $[\mathbf{q}]_v$ is the probability of the walk stopping at vertex v. The overall probability of stopping after t steps is then $\mathbf{q}^\mathsf{T} M^t \mathbf{p}_0$.

2.2 Continuous Quantum Walks

We begin by defining a Hilbert space, $\mathcal{H}_V$, spanned by basis states $|v\rangle : v \in V$ corresponding to the vertices of some graph, $G(V,E)$. We also have some initial state $|\Psi_0\rangle \in \mathcal{H}_V$. A *Continuous-Time Quantum Walk* [9] over time t is given by $\exp(\mathrm{i}H(G)t)\,|\Psi_0\rangle$, where $H(G)$ is given elementwise by

$$[H(G)]_{ij} = \begin{cases} -\gamma, & v_i \neq v_j \text{ and } \{v_i, v_j\} \in E; \\ 0, & v_i \neq v_j \text{ and } \{v_i, v_j\} \notin E; \\ d_i\gamma, & v_i = v_j; \end{cases} \tag{2.2.1}$$

with γ being the transition probability per unit time. A graph G admits *perfect state transfer* [10] between vertices v and u at time τ if there exists some $\alpha \in \mathbb{C}, |\alpha| = 1$ such that $\exp(\mathrm{i}H(G)\tau)\,|v\rangle = \alpha\,|u\rangle$. In [11], a graph is given in which an exponential separation in expected hitting time is observed between classical and quantum random walks. This graph, G_n, consists of two binary trees of depth n glued together at their leaves. An example, G_4, is shown in Fig. 2.1.

To be more precise, the probability of travelling from the root of the first tree to the root of the second in time polynomial in n is shown to be exponentially faster in the quantum case. To see this, we group the vertices of G_n into columns indexed by $j \in \{0, 1, \ldots, 2n\}$. Column 0 has just the left root, column 1 contains the adjacent two vertices and so on. Note that column n contains the 2^n vertices in the middle and column $2n$ has just the rightmost root. Analysing the classical case requires us only to keep track of the probabilities of being in a given column. When in the left tree ($0 \leq j \leq n$), the probability of moving to the right (column $j \to j+1$) is twice the probability of moving to the left (column $j \to j-1$). This means that the walk quickly moves into the middle of the graph. However, in the right hand tree the situation is reversed, whereby moving to the right has half the probability of moving to the left, thus making the time to destination exponential in n. More precisely, reaching column $2n$ from column 0 has probability less than 2^{-n}, meaning that traversing the graph in a time polynomial in n has a

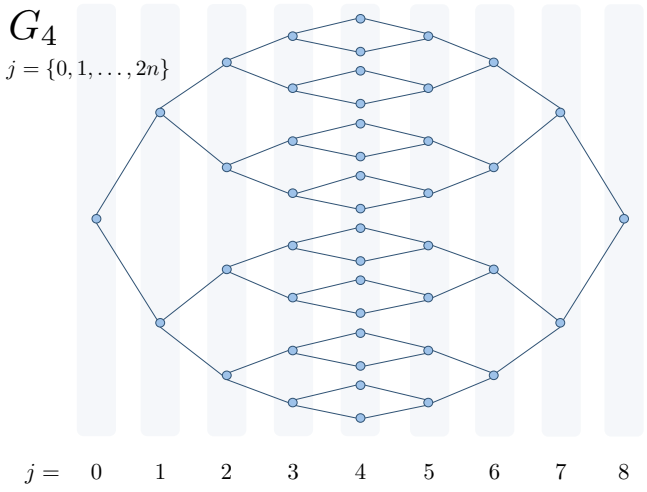

Figure 2.1: The graph G_4: Two binary trees of depth $n = 4$ glued together at their leaves, with column indices, $j = \{0, 1, \ldots, 2n\}$.

probability exponentially small in n.

In the quantum case, the symmetries of the Hamiltonian keep the state's evolution restricted to the $2n+1$-dimensional subspace spanned by the states $|\text{col } j\rangle$, the uniform superposition over all vertices in column j, that is,

$$|\text{col } j\rangle = \frac{1}{\sqrt{N_j}} \sum_{v \in \text{column } j} |v\rangle\,, \quad \text{where} \quad N_j = \begin{cases} 2^j, & 0 \le j \le n; \\ 2^{2n-j}, & n \le j \le 2n. \end{cases} \tag{2.2.2}$$

The non-zero elements of H in this basis are

$$\begin{aligned} \langle \text{col } j|\, H\, |\text{col } j \pm 1\rangle &= -\sqrt{2}\gamma; \\ \langle \text{col } j|\, H\, |\text{col } j\rangle &= \begin{cases} 2\gamma, & j \in \{0, n, 2n\}; \\ 3\gamma, & \text{otherwise.} \end{cases} \end{aligned} \tag{2.2.3}$$

This can thought of as a quantum walk on a line with $2n + 1$ vertices. In a time proportional to n, there is substantial probability of traversing this graph, in contrast to the classical case. This can be seen by first considering the infinite, translationally invariant line. The amplitude to move from vertex l to vertex m in time t is given by [12]

$$\langle m|\, \mathrm{e}^{-\mathrm{i}Ht}\, |l\rangle = \mathrm{e}^{-\mathrm{i}3\gamma t} \mathrm{i}^{m-l} J_{m-l}(2\sqrt{2}\gamma t), \tag{2.2.4}$$

where J_{m-l} is a Bessel function of order $m - l$. This corresponds to propagation speed $2\sqrt{2}\gamma$. In the limit of large n, the reduced graph of G_n becomes nearly identical to the infinite, translationally invariant line, making it plausible that the propagation speed would be the same. This has been shown numerically to be the case in [11].

We now consider the quantum analogue of the limiting distribution of a random walk, $\boldsymbol{\pi}$.

Due to unitarity, the walk will not reach a steady state. We use the definition from [13], which is dependent on the starting state $|i\rangle$:

$$\boldsymbol{\chi}_f = \lim_{t\to\infty} \frac{1}{t} \int_0^t \left| \langle f | \, \mathrm{e}^{-\mathrm{i}Ht'} \, |i\rangle \right|^2 \mathrm{d}t' \,. \tag{2.2.5}$$

This is the distribution resulting from a measurement over the vertices after a time chosen uniformly in $[0, t]$, in the limit of large t. It is found that the probability of traversing G_n is given by

$$\boldsymbol{\chi}_{\mathrm{col}\ 2n} \geq \frac{1}{2n+1}, \tag{2.2.6}$$

which is much greater than the classical result.

We can consider a slightly different definition of a continuous quantum walk, whereby the definition of $H(G)$ of Eq. 2.2.1 is modified to be merely the adjacency matrix $A(G)$. The quantum walk is thus enacted via the operator $U(t) := \exp(\mathrm{i}A(G)t)$. Godsil defines a matrix related to this walk in the following way [14]:

Definition 2.2.1 (Average mixing matrix)**.** The *average mixing matrix* is defined as

$$\widehat{M}_{G;T} = \frac{1}{T} \int_0^T U(t) \circ U(-t) \, \mathrm{d}t = \frac{1}{T} \int_0^T \exp(\mathrm{i}A(G)t) \circ \exp(-\mathrm{i}A(G)t) \, \mathrm{d}t \,,$$

where $\circ$ is the *Schur-Hadamard product* acting between matrices A and B elementwise as $(A \circ B)_{ij} = A_{ij}B_{ij}$.

We can derive an explicit form for the average mixing matrix as follows. The unitary $U(t)$ which induces the quantum walk can be written as

$$U(t) = \sum_r \exp(-\mathrm{i}\lambda_r t) E_r, \tag{2.2.7}$$

where where E_r is the rth idempotent of the spectral decomposition of $A(G)$ and λ_r the rth eigenvalue; that is, $A(G) = \sum_r \lambda_r E_r$. Then we substitute into the definition of $\widehat{M}_{G;T}$ (Def. 2.2.1), yielding

$$\widehat{M}_{G;T} = \sum_{\lambda_r \in \Lambda} \sum_{\lambda_{r'} \in \Lambda} E_r \circ E_{r'} \, \frac{1}{T} \int_0^T \exp(-\mathrm{i}(\lambda_r - \lambda_{r'})t) \, \mathrm{d}t \,, \tag{2.2.8}$$

where Λ is the set of eigenvalues of $A(G)$. Evaluating the integral we get

$$\widehat{M}_{G;T} = \sum_{\lambda_r \in \Lambda} \sum_{\lambda_{r'} \in \Lambda} E_r \circ E_{r'} \, \frac{\mathrm{i}(1 - \mathrm{e}^{\mathrm{i}T(\lambda_{r'} - \lambda_r)})}{T(\lambda_{r'} - \lambda_r)}. \tag{2.2.9}$$

We then define the average mixing matrix in the infinite time limit, $\widehat{M}_{G;\infty}$. Unless otherwise specified, when we say average mixing matrix, we mean $\widehat{M}_{G;\infty} =: \widehat{M}_G$, the infinite-time case.

Definition 2.2.2. The infinite-time average mixing matrix is given by

$$\widehat{M}_G := \widehat{M}_{G;\infty} = \lim_{T\to\infty} \widehat{M}_{G;T}.$$

The (i, j)-th entry of $\widehat{M}_G$ represents (in a rough sense) the average probability of a walk starting at vertex i ending at vertex j. The average mixing matrix is doubly stochastic and positive semi-definite. For G connected, all the entries of $\widehat{M}_G$ are positive and its eigenvalues lie in the interval $[0, 1]$. Furthermore, all entries of $\widehat{M}_G$ are rational. From Eq. 2.2.9, we have that

$$\widehat{M}_G = \sum_r E_r^{\circ 2}, \tag{2.2.10}$$

For certain graphs, $\widehat{M}_G$ takes a particularly simple form. In the case of a walk on a line of n vertices, the average mixing matrix takes the form

$$\frac{1}{2n+2}(2J + I + T), \tag{2.2.11}$$

where I is the identity, J is the all-ones matrix and T is the permutation matrix that swaps j and $n + 1 - j$ for each j.

Note. The average mixing matrix doesn't precisely encode the limiting distribution of a quantum walk as this ergodic behaviour is prevented by the unitarity of quantum dynamics. We treat this idea as intuition but are wary of taking it too literally.

2.3 Discrete (Coined) Quantum Walks

We now discuss the paradigm of *discrete quantum walks*, in which time is treated as a discrete variable, whereby the walk is defined by repeated application of a unitary operator on some initial state. The space on which the walk is defined is spanned by the basis states $|v, e\rangle$, such that $v \in V$, $v \in e$ and $e \in E$. Notice that the state space for the walk has grown relative to the continuous time case, with an additional subspace at each vertex for its associated edges. We also require a *coin* operator for each vertex, $C^{(v)}$, that is,

$$C^{(v)} = \sum_{e:v\in e} \sum_{e':v\in e'} c_{e,e'}^{(v)} |v, e\rangle\langle v, e'| . \tag{2.3.1}$$

A *coined quantum walk* [15] on $G(V, E)$ for t timesteps is U^t, where

$$U = S \cdot \sum_{v\in V} \sum_{e:v\in e} C^{(v)} |v, e\rangle\langle v, e| , \tag{2.3.2}$$

where we define the *shift* operator

$$S |v, e\rangle = \begin{cases} |v', e\rangle , & \text{if } \{v, v'\} = e; \\ 0, & \text{otherwise.} \end{cases} \tag{2.3.3}$$

Coined quantum walks on a line have been extensively studied [16, 17]. It has been found that contrary to the classical case, the properties of the walk are highly sensitive to both the coin and initial state.

For instance, a frequently used coin in the case of a walk on the infinite line is the *Hadamard*

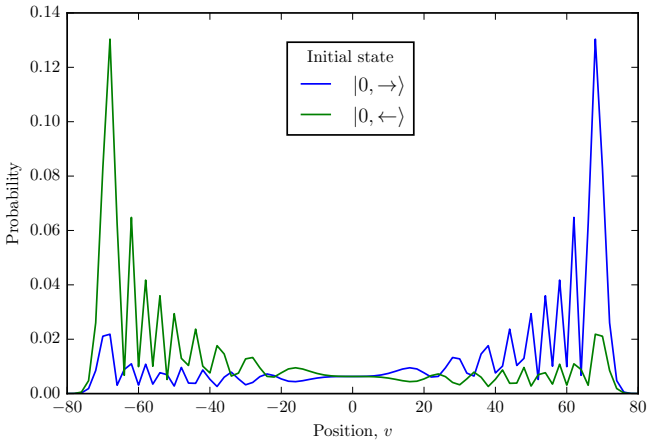

Figure 2.2: Probability distribution of Hadamard walk on a line for two starting states, $|0,\rightarrow\rangle$ and $|0,\leftarrow\rangle$ after $t = 100$ timesteps. Odd vertex values are omitted due to their probability being zero.

coin, $\frac{1}{\sqrt{2}}\left[\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right]$, providing an equal probability of moving left or right from a given vertex. For the equivalent classical random walk of t timesteps, the limiting distribution approaches a Gaussian of mean zero and variance $\sigma^2 = t$. Denoting by $\{\leftarrow,\rightarrow\}$ the edges moving left and right respectively from a given vertex v, we note from the symmetry of the graph that the basis states for the walk can be described by the states $|v\rangle \otimes |\leftrightarrow\rangle$, where $v \in \mathbb{Z}$. Our unitary which enacts the walk, U, using Eqs. 2.3.2 and 2.3.3, is therefore given by

$$U = \left(\sum_{v\in\mathbb{Z}} |v-1\rangle\langle v| \otimes |\leftarrow\rangle\langle\leftarrow| + \sum_{v\in\mathbb{Z}} |v+1\rangle\langle v| \otimes |\rightarrow\rangle\langle\rightarrow|\right) \cdot (I_V \otimes H_c), \tag{2.3.4}$$

where $I_V = \sum_{v\in\mathbb{Z}} |v\rangle\langle v|$ and H_c is the Hadamard coin. The distribution over the vertices of Hadamard walks starting in the states $|0,\leftarrow\rangle$ and $|0,\rightarrow\rangle$ differ both from a Gaussian as in the classical walk and from each other. This is shown in Fig. 2.2, the bimodality of the distribution being in stark contrast to the classical case.

2.4 Scattering Quantum Walks

Another related, but more physically motivated type of quantum walk is the *scattering walk*. Scattering quantum walks have been shown to be unitarily equivalent to the coined model [18]. We start by defining a *quantum graph* [19]:

$$\{\Gamma(V,E), \text{Hamiltonian operator } H \text{ on } E(\Gamma), \text{boundary conditions for } V(\Gamma)\},$$

where Γ is a *metric graph*, i.e. each vertex is assigned a length. Each vertex has associated to it a δ-function potential, making it a point scatterer. The total wavefunction Ψ is given by

$$\Psi = \begin{pmatrix} \psi_{e_1}(x_{e_1}) \\ \psi_{e_2}(x_{e_2}) \\ \vdots \\ \psi_{e_3}(x_{e_3}) \end{pmatrix}, \tag{2.4.1}$$

where the e_i denote edges. Note that the *state space is on the edges*, as opposed to the vertices in the previous case of continuous walks and both vertices and edges in the discrete-time paradigm.

A scattering quantum walk now involves solving the Schrödinger equation. The contemporary way this is done is by deriving the Green's function of the quantum graph [20]. This is calculated by a sum over paths and can be done analytically. We define the Green's function for a particle energy E, $G(E)$, using

$$[E - H(x_f)]G(x_f, x_i; E) = \delta(x_f - x_i), \tag{2.4.2}$$

where $H(x) = -\frac{\hbar^2}{2\mu}\frac{\mathrm{d}^2}{\mathrm{d}x^2} + V(x)$ and $G(x_f, x_i; E)$ is subject to appropriate boundary conditions. Say we have a complete set of normalised eigenstates ψ_s (discrete spectrum) and ψ_σ (continuous spectrum) satisfying

$$H\psi_s = E_s\psi_s \quad \text{and} \quad H\psi_\sigma = \frac{\hbar^2\sigma^2}{2\mu}\psi_\sigma. \tag{2.4.3}$$

Then, the solution to Eq. 2.4.2 is

$$G(x_f, x_i; E) = \sum_s \frac{\psi_s(x_f)\psi_s^*(x_i)}{E - E_s} + \int_0^\infty \mathrm{d}\sigma\, \frac{\psi_\sigma(x_f)\psi_\sigma^*(x_i)}{E - \frac{\hbar^2\sigma^2}{2\mu}}. \tag{2.4.4}$$

For a quantum graph, the approach generally used is the Feynman path integral approach. The Green's function is given by

$$G(x_f, x_i; E) = \frac{\mu}{\mathrm{i}\hbar^2 k}\sum_{\mathrm{sp}} W_{\mathrm{sp}} \exp\left[\frac{\mathrm{i}}{\hbar} S_{\mathrm{sp}}(x_f, x_i; k),\right] \tag{2.4.5}$$

where sp denotes each *scattering path*, i.e. every route from x_i to x_f. W_{sp} is the product of the reflection and transmission coefficients of all the vertices along the scattering path. S_{sp} is the classical action, kL_{sp}, where k is the wavenumber and L_{sp} the path length. An intelligent way to evaluate the sum in Eq. 2.4.5 is to separate infinite sets of scattering paths into classes, according to the scheme set out in [19]. Consider the basic example in Fig. 2.3. For the Green's function between x_i and x_f, we have

$$G_{if}(x_f, x_i; k) = \frac{\mu}{\mathrm{i}k\hbar^2}\exp[-\mathrm{i}kx_i]t_A^{(i,1)}P_1, \tag{2.4.6}$$

where $t_A^{(i,1)}$ is the transmission coefficient at vertex A, incoming from edge i, outgoing to edge 1. We then have

$$P_1 = \exp[\mathrm{i}k\ell_1]\Big(r_B^{(1)}P_2 + t_B^{(1,f)}\exp[\mathrm{i}kx_f]\Big), \tag{2.4.7}$$

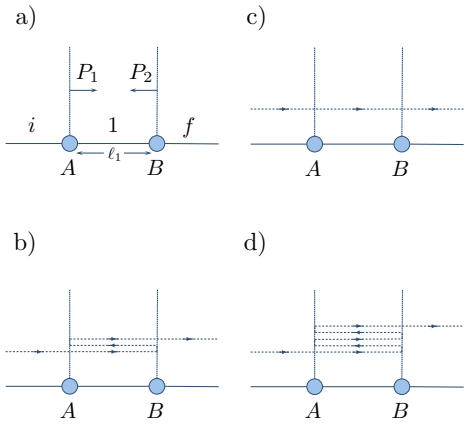

Figure 2.3: Illustration of scattering paths taken between two vertices, adapted from [19]. a) shows how we can separate paths into two classes, P_1 and P_2. We have P_1 as the set of *successful* right-moving paths starting at A (where $x = 0$) that reach x_f. P_2 is the set of left-moving paths from B reaching x_f. Here, x_i is to the left of the figure, x_f is off to the right.

where the first term represents the component of P_1 originating from P_2 and the second represents the transmitted component. Now

$$P_2 = \exp[\mathrm{i}k\ell_1]r_A^{(1)}P_1. \tag{2.4.8}$$

Solving Eqs. 2.4.7 and 2.4.8 for P_1 and substituting into Eq. 2.4.6 gives the Green's function

$$G_{if}(x_f, x_i; k) = \frac{\mu}{\mathrm{i}k\hbar^2}\frac{t_A^{(i,1)}t_B^{(1,f)}}{1 - r_A^{(1)}r_B^{(1)}\exp[2\mathrm{i}k\ell_1]}\exp[\mathrm{i}k(x_f - x_i + \ell_1)]. \tag{2.4.9}$$

Eigenstates and eigenenergies can be found using the poles of $G(x_f, x_i; k)$, k_n and Eq. 2.4.4. By grouping infinite sets of paths like so and decomposing large graphs into smaller substructures, one can calculate Green's functions of an arbitrary graph analytically. We note that scattering quantum walks have most extensively studied on *open* quantum graphs, i.e. some of the edges extend to infinity.

2.5 Algorithmic Applications

We now describe two quantum algorithms from the literature which use the discrete-time formalism.

2.5.1 Element Distinctness

First, we introduce the notation $[N] := 1, \ldots, N$. The problem of element distinctness is then formally defined as follows:

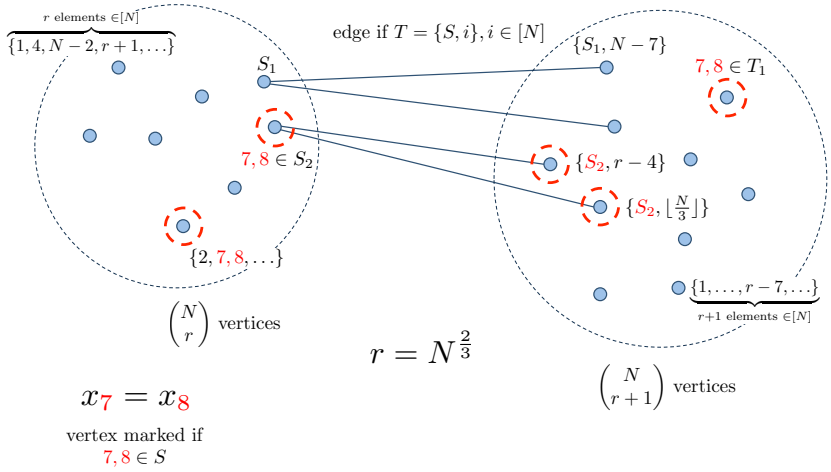

Figure 2.4: Illustration of graph G used in Ambainis' element distinctness algorithm. Example values are used for the marked elements and label sets for concreteness.

Problem 2.5.1. Element distinctness: Given $x_1, \ldots, x_N \in [M]$, does there exist $i, j \in [N]$, $i \neq j$ such that $x_i = x_j$?

This problem is a subset of k-distinctness:

Problem 2.5.2. k-distinctness: Given $x_1, \ldots, x_N \in [M]$, does there exist $i_1, \ldots, i_k \in [N]$ such that $x_{i_1} = x_{i_2} = \cdots = x_{i_k}$? We call such an event a k-collision.

Ambainis provides an algorithm in [21] solving k-distinctness in $O(N^{2/3})$ queries for $k = 2$, Prob. 2.5.1 (compared to $O(N)$ queries classically). For arbitrary k, the problem (Prob. 2.5.2) is solved using $O(N^{k/k+1})$ queries.

The procedure proceeds as follows: first we have the basis states $|i, a, z\rangle$, where i has $\lceil \log N \rceil$ *bits*, a has $\lceil \log M \rceil$ *qubits* and z contains scratch bits. We have access to an oracle, $\mathcal{O}$, acting like so:

$$\mathcal{O}\,|i, a, z\rangle \to |i, (a + x_i) \bmod M, z\rangle\,. \tag{2.5.1}$$

Define a graph G with $\binom{N}{r} + \binom{N}{r+1}$ vertices, with $r = N^{2/3}$. Each vertex v_S corresponds to a set $S \subset [N]$ of size r or $r+1$. The vertices v_S and v_T are connected by an edge if $T = S \cup \{i\}, i \in [N]$. A vertex is marked if S contains i, j such that $x_i = x_j$. An example graph is shown in Fig. 2.4. This construction results in element distinctness being reduced to finding a marked vertex in G. We can check if v_S is marked in $r + M$ steps, assuming there exists an algorithm $\mathcal{A}$ which can find a marked vertex in M steps. Now we describe $\mathcal{A}$ for the one k-collision case.

Firstly, we let $x_1 \ldots x_N \in [M]$. Consider the Hilbert spaces $\mathcal{H}$ and $\mathcal{H}'$, corresponding to the left and right subgraphs in Fig. 2.4. The basis states of $\mathcal{H}$ are $|S, x, y\rangle$ with $S \subseteq [N]$, $|S| = r$, $x \in [M]^r$, $y \in [N] \setminus S$ and $\dim(\mathcal{H}) = \binom{N}{r} M^r (N - r)$. The properties of $\mathcal{H}'$ take corresponding form (with $r \to r + 1$), apart from the third register, where $y \in S$.

The main thrust of the algorithm is alternating a quantum walk and a transformation flipping

the phase if the state of the walk contains a k-collision (i.e. has a marked element). The 'walk'

Algorithm 1 Element Distinctness Algorithm [21, Alg. 2]

1: Generate the uniform superposition $\frac{1}{\sqrt{\binom{N}{r}(N-r)}} \sum_{|S|=r, y\notin S} |S\rangle |y\rangle$.
2: Apply $\mathcal{O}$ to get the x_i, giving

$$\frac{1}{\sqrt{\binom{N}{r}(N-r)}} \sum_{|S|=r, y\notin S} \left(|S\rangle |y\rangle \bigotimes_{i\in S} |x_i\rangle \right).$$

3: **for** $t_1 = O((N/r)^{k/2})$ repeats **do**
4: **if** k-collision **then**
5: Apply phase flip $|S\rangle |y\rangle |x\rangle \to -|S\rangle |y\rangle |x\rangle$.
6: **end if**
7: **for** $t_2 = O(\sqrt{r})$ steps **do**
8: Perform one step of the quantum walk (Algorithm 2).
9: **end for**
10: **end for**
11: Measure final state.
12: **if** S contains a k-collision **then**
13: **return** YES
14: **else**
15: **return** NO
16: **end if**

Algorithm 2 Quantum Walk for Element Distinctness [21, Alg. 1]

1: Apply the 'coin-flip' transformation

$$|S\rangle |y\rangle \to |S\rangle \left(\left(-1 + \frac{2}{N-r} \right) |y\rangle + \frac{2}{N-r} \sum_{y'\notin S, y'\neq y} |y'\rangle \right).$$

2: Map the state from $\mathcal{H}$ to $\mathcal{H}'$ by adding y to S and introducing 0 in the location in x corresponding to y.
3: Apply the transformation

$$|S\rangle |y\rangle \to |S\rangle \left(\left(-1 + \frac{2}{r+1} \right) |y\rangle + \frac{2}{r+1} \sum_{y'\in S, y'\neq y} |y'\rangle \right).$$

4: Uncompute the element of x corresponding to the new y by querying $\mathcal{O}$ for x_y.
5: Map back into $\mathcal{H}$ by removing the 0 corresponding to y from x and removing y from S.

manifests as the mapping between $\mathcal{H}$ and $\mathcal{H}'$.

Proof/Analysis sketch:

1. Create superpositions of $|S, y\rangle$ which live in subspaces $\tilde{\mathcal{H}}$ and $\tilde{\mathcal{H}}'$. Then prove that the algorithm stays restricted to these subspaces.

2. The operator U is the unitary acting on $\tilde{\mathcal{H}}$ over one step of the quantum walk. Derive the form this takes and show it has $2k+1$ eigenvalues. One of them $(+1)$ corresponds to

$|\psi_{\text{start}}\rangle$. Others are $e^{\pm\theta_1 i}, \ldots, e^{\pm\theta_k i}$, with $\theta_j = (2\sqrt{j} + o(1))\frac{1}{\sqrt{r}}$. Show U is block diagonal and diagonalise the blocks. Treat steps 1-3 and 4-6 of Alg. 1 as separate unitaries, then can show that eigenvalues of U are close to eigenvalues in steps 4-6.

3. Set[2] $t_2 = \lceil \frac{\pi}{3\sqrt{k}}\sqrt{r} \rceil$ and apply U_1U_2, where $U_2 = U^{t_2}$ and U_1 is the phase flip operation. Define $|\psi_{\text{good}}\rangle$, the uniform superposition over all marked vertices. Prove general statement about operators U_1 and U_2 satisfying the following:

   (a) U_1 maps $|\psi_{\text{good}}\rangle \to -|\psi_{\text{good}}\rangle$ and leaves orthogonal states unchanged.

   (b) $U_2|\psi_{\text{start}}\rangle = |\psi_{\text{start}}\rangle$ and other eigenvalues of U_2 are $e^{i\theta}$ for $\theta \in [\epsilon, 2\pi - \epsilon]$.

   A corollary is that there exists a $t = O(1/\alpha)$ such that $|\langle\psi_{\text{good}}|(U_2U_1)^t|\psi_{\text{start}}\rangle| = \Omega(\epsilon)$. Calculate α by counting the number of vertices.

4. Proceed with proof by expressing $|\psi_{\text{start}}\rangle$ and $|\psi_{\text{good}}\rangle$ in U_2's eigenbasis. Can derive eigenvectors of U_2U_1 in this basis. Manipulations and defining $|\psi_{\text{end}}\rangle$ eventually give rise to the inner product.

2.5.2 Grover Search

We now describe the discrete quantum walks formulation [22] of Grover's famous search algorithm [4], whereby a marked element in an unstructured database of size N is found in $O(\sqrt{N})$ queries to an appropriate oracle. This algorithm hinges on a discrete walk on the n-dimensional hypercube, a graph with $N = 2^n$ vertices, each of which is labelled by an n-bit binary string. Two nodes labelled by $\mathbf{x}$ and $\mathbf{y}$ are connected by an edge if the *Hamming distance* between them is unity, i.e. one bit differs. Each vertex has degree n, thus we can factor the state space of the quantum walk in a similar manner to the walk on a line treated in Sec. 2.3; namely $|v, e\rangle \in \mathcal{H} = \mathcal{H}^{2^n} \otimes \mathcal{H}^n$, with $\mathcal{H}^{2^n}$ the vertex space and $\mathcal{H}^n$ the coin space.

We describe each state in $\mathcal{H}$ by $|\mathbf{x}, d\rangle$, where $\mathbf{x}$ labels the position in the hypercube and d the direction[3]. Our shift operator is given by

$$S = \sum_{d=0}^{n-1} \sum_{\mathbf{x}\in\{0,1\}^n} |\mathbf{x} \oplus \mathbf{e}_d\rangle\langle\mathbf{x}, d| \,, \tag{2.5.2}$$

where $\mathbf{e}_d$ is the dth basis vector on the hypercube. For the coin, we first define C_0, the *diffusion operator* or *Grover coin*:

$$C_0 = G = -I + 2\left|s^C\right\rangle\!\left\langle s^C\right| , \tag{2.5.3}$$

where $\left|s^C\right\rangle = \frac{1}{\sqrt{n}}\sum_d |d\rangle$, the equal superposition over all directions. In order for the walk to perform a search algorithm we must refine the coin. Typically in a search algorithm we possess an oracle $\mathcal{O}$, acting as $\mathcal{O}|\mathbf{x}\rangle|y\rangle \to |\mathbf{x}\rangle|y \oplus f(\mathbf{x})\rangle$, where $|y\rangle$ is some marker qubit and $f(\mathbf{x}) = 1$ if the vertex $\mathbf{x}$ is 'marked' and zero otherwise. For our coin we wish to apply a 'marking coin', C_1, to the marked vertex and C_0 to unmarked vertices. We take $C_1 = -I$ to selectively phase shift

[2] This sets all eigenvalues to $e^{i\theta}$, with $\theta \in [c, 2\pi - c]$ for some constant c.

[3] Concretely, d denotes the edge connecting vertices $\mathbf{x}$ and $\mathbf{x}$ with its dth bit flipped.

the marked vertex. Without loss of generality we can assume the marked vertex is the all-zero string, $\mathbf{0}$, so our coin operator becomes

$$C' = I \otimes C_0 + |\mathbf{0}\rangle\langle\mathbf{0}| \otimes (C_1 - C_0). \tag{2.5.4}$$

The search algorithm is described by Algorithm 3. With probability $\frac{1}{2} - O(1/n)$, the outcome

Algorithm 3 Quantum Walk Search

1: Initialise in the equal superposition over all states, $|\psi_0\rangle = \left|s^V\right\rangle \otimes \left|s^C\right\rangle$, where $\left|s^V\right\rangle = \frac{1}{2^n}\sum_{\mathbf{x}}|\mathbf{x}\rangle$. This can be achieved efficiently by applying single-qubit Hadamards to the $|\mathbf{0}\rangle$ state.
2: Apply the operator U^{t_f}, where $U = S \cdot C'$, with S and C' defined according to Eqs. 2.5.2 and 2.5.4 respectively. The number of steps in the quantum walk, $t_f = \pi/2\sqrt{2^n}$.
3: Measure the state in the $|\mathbf{x}, d\rangle$ basis.

of the measurement will be the marked vertex. Repeating the algorithm a constant number of times can bring this probability arbitrarily close to unity.

Proof/Analysis sketch: The proof proceeds by evaluating U^t on the state $|\psi_0\rangle$ as follows:

1. The walk on the hypercube is first mapped to a random walk on the line, using the symmetry of the hypercube.

2. Two approximate eigenvectors of U, $|\psi_0\rangle$ (initial state) and $|\psi_1\rangle$ (orthogonal state to $|\psi_0\rangle$ close to final state) are used to show that only two eigenvectors of U need to be considered, $|\omega_0\rangle$ and $|-\omega_0\rangle$.

3. It is demonstrated that $|\psi_0\rangle$ and $|\psi_1\rangle$ can be written as linear combinations of $|\omega_0\rangle$ and $|-\omega_0\rangle$, thus meaning that each application of U acts as a rotation in the $|\omega_0\rangle$, $|-\omega_0\rangle$ plane from ψ_0 towards $|\psi_1\rangle$.

4. It is shown that this angle of rotation for each application of U is $1/\sqrt{2^{n-1}}$, so the search is complete after approximately $\frac{\pi}{2}\sqrt{2^{n-1}}$ steps, or $O(\sqrt{N})$ calls to the oracle, where $N = 2^n$ vertices.

2.6 Universality of Quantum Walks

Quantum Walks have been shown to be universal for quantum computation in the discrete [23] and continuous-time [24] regimes[4]. We briefly describe how this is done in the continuous time case, with the discrete time case being similar.

The general idea is to represent each computational basis state by virtual 'wires', idealised semi-infinite lines which can be simulated by a finite line [9]. Gates are implemented by subgraph structures known as 'widgets'. The single-qubit unitary gates implemented are the phase-gate, $\left[\begin{smallmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{smallmatrix}\right]$ and $-\frac{1}{\sqrt{2}}\left[\begin{smallmatrix} i & 1 \\ 1 & i \end{smallmatrix}\right]$, which generate the group SU(2). Along with the CNOT gate

[4]The universality of the discrete model coupled with the unitary equivalence [18] between discrete and scattering walks suggests universality of the scattering walks model, but this is yet to be explicitly shown.

which is also implemented, these form a universal set of gates for quantum computation. This construction allows any m-gate quantum circuit on n qubits to be simulated by a continuous-time quantum walk on an N-vertex graph of maximum degree 3 over time $t = \text{poly}(\log N)$, where $\log N = \text{poly}(m)$. We compare this to the result that any *sparse* graph; that is, a graph of bounded degree, with N vertices can be simulated by a quantum computer using $\text{poly}(\log N)$ gates [25]. A corollary drawn from this is that the decision problem of whether a quantum walk evolves from a given vertex to some other vertex or not, is BQP-complete, where BQP is the complexity class characterised by decision problems which are solvable in polynomial time by a quantum computer with error probability at most $\frac{1}{3}$.

Chapter 3

Learning on Graphs

3.1 Kernel Methods in Pattern Analysis

Pattern analysis [26] concerns the detection of structure and relations within data, and sees wide application in fields such as machine learning, bioinformatics and data mining. We formally define a *pattern* as a function f acting on some data X arising from some source such that

$$f(X) \approx 0 \tag{3.1.1}$$

for all possible data X, where what approximately equal to zero means is dependent on the context. A *pattern analysis algorithm* is an algorithm $\mathcal{A}$, which upon input of a finite set of *training* data points $\{X_i\}$, returns a pattern function f such that

$$\mathbb{E}[f(X)] \approx 0, \tag{3.1.2}$$

where $\mathbb{E}$ denotes the expectation over all possible source data X. The algorithm $\mathcal{A}$ returns NO PATTERN if no pattern is found. We have deliberately left what form the data X takes ambiguous so as to emphasise the general application of pattern analysis algorithms to many different data types.

The most well-understood and efficient algorithms, such as Ridge Regression [27] and Support Vector Machines [28] typically find linear relations between training vectors $\mathbf{x} \in \mathbb{R}^n$, where we call $\mathbb{R}^n$ the *feature space*. Often one is concerned with non-linear relations within data. By an appropriate mapping

$$\phi : \mathbf{x} \in \mathbb{R}^n \mapsto \boldsymbol{\phi}(\mathbf{x}) \in F \subseteq \mathbb{R}^N \tag{3.1.3}$$

into a larger feature space, non-linear relations $f(\mathbf{x})$ can be re-cast as linear relations $g(\boldsymbol{\phi}(\mathbf{x}))$, as illustrated in Fig. 3.1. Many of these algorithms have two formulations, the *primal* and the *dual* formulations. The main difference between the primal and dual formulations is that typically in the primal formulation, one has to invert an $N \times N$ matrix yielding a vector $\mathbf{w} \in \mathbb{R}^n$ encoding f, whereas in the dual formulation an $\ell \times \ell$ matrix is solved, where ℓ is the number of training vectors and $\mathbf{w}$ is returned as a linear combination of the training vectors. As inverting an $N \times N$

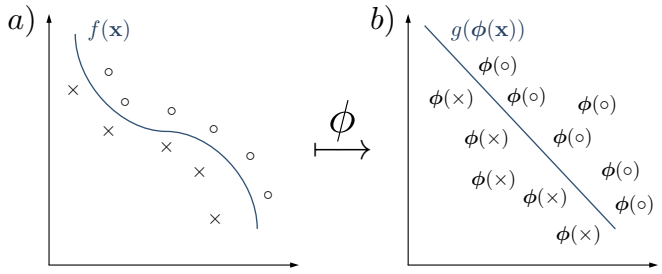

Figure 3.1: Illustration of how the mapping ϕ embeds the data into a feature space in which the pattern is now linear. a) Training data with nonlinear boundary. b) Training data mapped into feature space (RKHS) now with linear separating boundary.

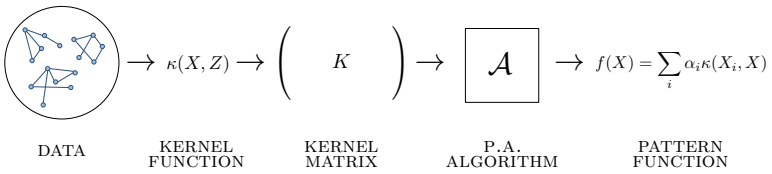

Figure 3.2: Summary of how kernel methods are applied to pattern recognition.

matrix has complexity $O(N^3)$, we note that it becomes advantageous to solve dual problems for highly non-linear pattern problems as the size of the feature space induced by ϕ grows with the non-linearity.

An additional benefit is that in the dual formulation, the solution is found using *only* inner products between training vectors in the feature space of interest. We thus introduce the notion of a *kernel*.

Definition 3.1.1. A *kernel function*, $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, is a function satisfying

$$\kappa(X_1, X_2) = \langle \phi(X_1), \phi(X_2) \rangle,$$

for some objects $X_1, X_2 \in \mathcal{X}$ and a feature mapping $\phi : \mathcal{X} \mapsto F$.

Crucially, computing the kernel *doesn't* require an explicit mapping into the larger feature space, F, meaning efficient computation of kernels in high-dimensional feature spaces is possible. Also, the space of training data, $\mathcal{X}$, doesn't have to be $\mathbb{R}^N$, so kernels between many different types of data objects can be computed, while still taking advantage of well-understood pattern analysis algorithms defined for linear patterns on data in $\mathbb{R}^N$. In this document we will be concerned with *graph kernels*.

Fig. 3.2 is a graphic summarising how kernel methods are used in pattern recognition algorithms. We have as input some training data, then the kernel function is applied between all pairwise combinations of training data. This now defines a *kernel matrix*, which is used as input to a pattern analysis algorithm which returns a pattern function $f(X)$ that can be applied to new data.

3.2 Properties of Kernels

Given a set of objects $S = \{X_1, X_2, \ldots X_\ell\}$ the $\ell \times \ell$ *kernel matrix* is defined element-wise by

$$K_{ij} = \langle \phi(X_i), \phi(X_j) \rangle = \kappa(X_i, X_j). \tag{3.2.1}$$

We note that the kernel matrix is *symmetric*, i.e. $K = K^\mathsf{T}$, from its definition. Also note that the kernel matrix is the *Gram matrix* in feature space. We now state some properties of kernel matrices without proof.

Kernel matrices are *positive semi-definite* (p.s.d.). A symmetric matrix is p.s.d. if its eigenvalues are all non-negative. Equivalently, this holds if and only if

$$\mathbf{v}^\mathsf{T} K \mathbf{v} \geq 0 \tag{3.2.2}$$

for all vectors $\mathbf{v}$. It is this property which guarantees that the kernel function κ does indeed compute the inner product between objects mapped by ϕ, which we shall see later. In fact, the new feature space induced by ϕ is a Hilbert space, $\mathcal{H}$, where by a Hilbert space we mean an inner product space which is *complete* and *separable*. Explicitly, the kernel function $\kappa(X, Z)$ can be expanded in terms of the functions ϕ_j, the basis of $\mathcal{H}$, satisfying $\langle \phi_i, \phi_j \rangle = \delta_{ij}$ as such:

$$\kappa(X, Z) = \sum_{j=1}^{\infty} \phi_j(X) \phi_j(Z). \tag{3.2.3}$$

The Hilbert space mapped into by a kernel is call a *reproducing kernel Hilbert space* (RKHS). The *Moore-Aronszajn theorem* states that for κ a symmetric, positive semi-definite kernel on a set X there exists a unique Hilbert space of functions on X for which κ is a reproducing kernel. Thus a kernel matrix being p.s.d. ensures that an inner product between vectors in feature space is taking place.

New, more complex kernels can be made out of combinations of simpler kernels, as follows. Let κ_1 and κ_2 be kernels over $X \times X$, $a \in \mathbb{R}^+$, $f : X \to \mathbb{R}$, $\phi : X \to \mathbb{R}^N$. The kernel κ_3 is over $\mathbb{R}^N \times \mathbb{R}^N$ and B is a symmetric, $n \times n$, p.s.d. matrix. Then the following are kernels:

1. $\kappa_1(X, Z) + \kappa_2(X, Z)$,
2. $a\kappa_1(X, Z)$,
3. $\kappa_1(X, Z)\kappa_2(X, Z)$,
4. $f(X)f(Z)$,
5. $\kappa_3(\phi(X), \phi(Z))$,
6. $X^\mathsf{T} B Z$ iff $X, Z \subseteq \mathbb{R}^n$.

3.3 Example - Hard Margin Support Vector Machine

We now provide an illustrative example of kernels being used in a classification setting, the Hard Margin Support Vector Machine (SVM), following the presentation in [26]. Consider Fig. 3.1b: the feature vectors $\boldsymbol{\phi}(\mathbf{x})$ are separated by a hyperplane[1]. The task of finding this hyperplane (and hence the nonlinear boundary in Fig. 3.1a, which *completely* separates the two classes of data is the main thrust of Hard margin SVM.

More precisely, we have a training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$ where the $\mathbf{x}_i$ are training vectors and $y_i \in \{-1, 1\}$ are the respective class labels. We have a function $g(\mathbf{x})$ given by

$$g(\mathbf{x}) = \langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i) \rangle + b, \tag{3.3.1}$$

where $\mathbf{w}$ is a *weight vector* in the RKHS induced by ϕ and b is a *threshold*. The quantity $y_i g(\mathbf{x}_i)$ measures the distance of a given vector $\boldsymbol{\phi}(\mathbf{x}_i)$ is from the separating hyperplane, $\{\mathbf{x} : g(\mathbf{x}) = 0\}$. The smallest distance of any of the vectors to the hyperplane is the *margin*, γ. The vectors which lie at distance γ from the hyperplane are known as *support vectors*. The goal of the SVM algorithm is to find the separating hyperplane which maximises the margin γ, which is achieved by solving the following optimisation problem:

$$\begin{aligned} \max_{\mathbf{w},b,\gamma} \quad & \gamma \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i) \rangle + b) \geq \gamma, \ i = 1, \ldots, \ell, \\ & \text{and } \|\mathbf{w}\|^2 = 1. \end{aligned} \tag{3.3.2}$$

This problem doesn't appear immediately amenable to kernel methods. In order to use a kernel we transform to the *dual* optimisation problem, by introducing Lagrange multipliers $\boldsymbol{\alpha}$, λ and solving the Lagrangian (now minimising $-\gamma$ for convenience):

$$L(\mathbf{w}, b, \gamma, \boldsymbol{\alpha}, \lambda) = -\gamma - \sum_{i=1}^{\ell} \alpha_i [y_i(\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i) \rangle + b) - \gamma] + \lambda\Big(\|\mathbf{w}\|^2 - 1\Big). \tag{3.3.3}$$

Optimising with respect to the primal variables $\mathbf{w}$, b, γ and the Lagrange multiplier λ yields (after some algebra)

$$L(\boldsymbol{\alpha}) = -\left(\sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle \right)^{1/2} = -\left(\sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \right)^{1/2}, \tag{3.3.4}$$

which we denote as the dual Lagrangian. Note that solving this optimisation now involves explicit calculation of the kernel matrix. Also, the problem is now defined on a space the size of the number of training points as opposed to the size of the feature space, which can be arbitrarily large depending on the mapping, ϕ, employed. The full hard margin SVM algorithm in dual form is given in Alg. 4. We note that the only non-zero α_i in the dual solution $\boldsymbol{\alpha}^*$ correspond to the support vectors themselves. This makes evaluation of the decision function f in Alg. 4 very

[1] In this case a hyperplane in two-dimensions, i.e. a line.

Algorithm 4 Hard Margin SVM - Dual Form

input: training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$, $\delta > 0$.

1: Find the solution $\boldsymbol{\alpha}^*$ to the optimisation problem: $W(\boldsymbol{\alpha}) = -\Big(\sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)\Big)^{1/2}$, subject to $\sum_{i=1}^{\ell} y_i \alpha_i = 0$, $\sum_{i=1}^{\ell} \alpha_i = 1$ and $\alpha_i \geq 0$, $i = 1, \ldots, \ell$.

2: Margin $\gamma^* = \sqrt{-W(\boldsymbol{\alpha}^*)}$.

3: Choose i such that $\alpha_i^* > 0$:

4: $\quad b = y_i(\gamma^*)^2 - \sum_{j=1}^{\ell} \alpha_j^* y_j \kappa(\mathbf{x}_j, \mathbf{x}_i)$.

5: $f(\,\cdot\,) = \operatorname{sgn}\Big(\sum_{j=1}^{\ell} \alpha_j^* y_j \kappa(\mathbf{x}_j, \,\cdot\,) + b\Big)$; ▷ Decision function for new data.

6: $\mathbf{w} = \sum_{j=1}^{\ell} y_j \alpha_j^* \boldsymbol{\phi}(\mathbf{x}_j)$

output: weight vector $\mathbf{w}$, dual solution $\boldsymbol{\alpha}^*$, margin γ^* and function f implementing classification rule represented by hyperplane.

efficient. Also, observe that to obtain the vector $\mathbf{w}$ knowledge of the feature map ϕ is needed, which is non-trivial to obtain from the kernel function κ. However, implementing the decision rule f on new data, the most important task of SVM, doesn't require the explicit mapping into feature space, demonstrating the power of kernel methods for learning algorithms.

3.4 Graph Kernels Overview

We now introduce kernels between graphs, otherwise known as graph kernels. We describe two important paradigms, random walk graph kernels and Weisfeiler-Lehman graph kernels.

3.4.1 Random Walks Graph Kernels

We now discuss a kernel construction between graphs with *labelled edges*, based on classical random walks. We let $\mathcal{X}$ be a set of labels including the special label ξ. An edge-labelled graph G has associated with it a label matrix $X \in \mathcal{X}^{|V|\times|V|}$ where X_{ij} is the label of the edge $\{v_i, v_j\}$ and $X_{ij} = 0$ if $\{v_i, v_j\} \notin E$. We let $\mathcal{H}$ be the RKHS induced by the kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, with ϕ denoting the corresponding feature map. We assume ξ maps to the zero element of $\mathcal{H}$ under ϕ. Furthermore, we introduce the *feature matrix* $\Phi(X)$, where $[\Phi(X)]_{ij} = \phi(X_{ij})$. Note that the elements $[\Phi(X)]_{ij}$ are vectors in $\mathcal{H}$.

We define a *tensor product graph*, $G_\otimes$, between two Graphs G and G' such that

$$V_\otimes = \{\{v_i, v_r'\} : v_i \in V, v_r' \in V'\} \quad \text{and} \tag{3.4.1}$$

$$E_\otimes = \{(\{v_i, v_r'\}, \{v_j, v_s'\}) : \{v_i, v_j\} \in E, \{v_r', v_s'\} \in E'\}. \tag{3.4.2}$$

We note that the adjacency matrix $A(G_\otimes) = A(G) \otimes A(G')$. An example of a tensor-product graph is given in Fig. 3.3. Performing a classical random walk on the graph $G_\otimes$ is equivalent to a random walk on G and G' simultaneously [29]. We let $\mathbf{p}$ and $\mathbf{p}'$ be the initial probability distributions over the vertices of G and G' respectively. Then the initial probability distribution for the walk on $G_\otimes$ is $\mathbf{p}_\otimes = \mathbf{p} \otimes \mathbf{p}'$. Likewise, for stopping probabilities $\mathbf{q}$ and $\mathbf{q}'$ we have $\mathbf{q}_\otimes = \mathbf{q} \otimes \mathbf{q}'$.

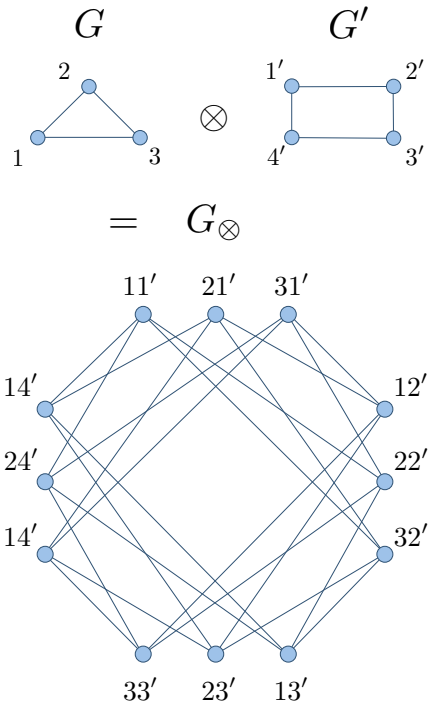

Figure 3.3: Two graphs (top) and their tensor product (bottom). There is an edge between two nodes if an edge exists between *both* sets of nodes corresponding to G and G'.

For edge-labelled G, G', we can associate a weight matrix $W_\otimes \in \mathbb{R}^{|V||V'|\otimes|V||V'|}$ with $G_\otimes$:

$$W_\otimes = \Phi(X) \otimes_{\langle\cdot\rangle_\mathcal{H}} \Phi(X'), \tag{3.4.3}$$

where by $\otimes_{\langle\cdot\rangle_\mathcal{H}}$ we mean taking the usual Kronecker product but then taking the inner product between the vector elements of $\Phi(X)$, $\Phi(X')$. Explicitly,

$$[\Phi(X) \otimes_{\langle\cdot\rangle_\mathcal{H}} \Phi(X')]_{(i-1)|V'|+k,(j-1)|V'|+l} = \langle\phi(X_{ij}), \phi(X'_{kl})\rangle_\mathcal{H}. \tag{3.4.4}$$

Letting the edge label set $\mathcal{X} = \{1, 2, \ldots, d\}$ and our feature space $\mathcal{H} = \mathbb{R}^d$ with the dot product, we define our feature map ϕ to act as follows:

$$[\Phi(X)]_{ij} = \begin{cases} \mathbf{e}_\ell / d_i, & \text{if edge } \{v_i, v_j\} \text{ labelled with } \ell; \\ \mathbf{0}, & \text{otherwise.} \end{cases} \tag{3.4.5}$$

So our weight matrix W has a non-zero entry iff a corresponding edge exists in $G_\otimes$ and the associated edges and G and G' share the same label. We can thus express $W_\otimes$ in the following manner:

$$W_\otimes = \sum_{\ell=1}^{d} {}^\ell A(G) \otimes {}^\ell A(G'), \tag{3.4.6}$$

where ${}^\ell A(G)$ is the normalised adjacency matrix of G filtered by the label ℓ, i.e. $[{}^\ell A(G)]_{ij} =$

$[A(G)]_{ij}$ if $X_{ij} = \ell$.

Given initial starting and stopping distributions $\mathbf{p}_\otimes$ and $\mathbf{q}_\otimes$, the expected similarity of length k random walks on G and G' is $\mathbf{q}_\otimes^\mathsf{T} W_\otimes^k \mathbf{p}_\otimes$. A natural way to define a kernel is to sum up these terms for all k. The problem with this is that the sum may not converge, so we introduce non-negative coefficients $\mu(k)$, giving the *random walk kernel* [30]

$$\kappa(G, G') = \sum_{k=0}^{\infty} \mu(k)\, \mathbf{q}_\otimes^\mathsf{T} W_\otimes^k \mathbf{p}_\otimes. \tag{3.4.7}$$

The choice of $\mathbf{p}_\otimes$, $\mathbf{q}_\otimes$ and $\mu(k)$ give the kernel great flexibility. Typically, $\mathbf{p}_\otimes$, $\mathbf{q}_\otimes$ are chosen to be uniform, so we consider only different choices of $\mu(k)$.

Two choices of $\mu(k)$ studied in this framework are $\mu(k) := \lambda^k$ and $\mu(k) := \lambda^k/k!$ for some fixed positive λ [31]. These choices reduce Eq. 3.4.7 down to

$$\kappa(G, G') = \mathbf{q}_\otimes^\mathsf{T}(I - \lambda W_\otimes^{-1})\mathbf{p}_\otimes \quad \text{and} \quad \kappa(G, G') = \mathbf{1}^\mathsf{T} \exp\big(\lambda \tilde{A}_\otimes\big)\mathbf{1} \tag{3.4.8}$$

respectively, where $\tilde{A}_\otimes$ is the unnormalised adjacency matrix of the direct product graph. These choices of $\mu(k)$ do away with the infinite sum in Eq. 3.4.7 and allow direct computation of the kernels.

Results for the efficiency of algorithms for computing the first kernel in Eq. 3.4.8, also known as the *geometric kernel*, have been presented in [30]. The time-complexity of computing the $m \times m$ kernel matrix is $O(m^2 d n^3)$, where d is the size of the label set and each graph has n vertices. Experimental results are also presented agreeing with the runtime complexity, which we omit for brevity.

It has been shown that many different schemes for graph kernels can in fact be described using the formalism of random walks kernels, as in Eq. 3.4.7. Such examples include rational kernels [32] and R-convolution kernels [33].

3.4.2 Weisfeiler-Lehman Graph Kernels

Weisfeiler-Lehman (WL) graph kernels [34] act on graphs with labelled vertices, and are based on the 1-dimensional WL test for graph isomorphism [35] (also known as *naïve vertex classification*), shown in Algorithm 5. The main thrust of the algorithm is to augment the node labels of each graph, compress them, then to compare label *multisets* at each iteration of the algorithm, where by a multiset we mean a set where multiple copies of the same element are allowed. The WL algorithm stops if the label multisets differ (not isomorphic); if it doesn't stop after N iterations, the two graphs are either isomorphic, or the algorithm hasn't been able to determine if the graphs are isomorphic. Unlabelled graphs have initial vertex label assignments $l_0(v)$ as their degrees $d(v)$. We can see immediately that for any unlabelled *regular graph*; that is, a graph $G : \forall v \in V,\, d(v) = d$, the WL test fails. We note that vertex labels themselves are also multisets. The WL graph isomorphism test has been shown to work for most graphs [36] and its runtime complexity is $O(hm)$, where h is the number of iterations and m is the number of edges G and G' have. Before we define the WL graph kernel, we first define a *Weisfeiler-Lehman graph* of height i of the node-labelled graph $G = (V, E, \ell) = (V, E, l_0)$ as the graph $G_i = (V, E, l_i)$, where

Algorithm 5 (1-dimensional) WL test of graph isomorphism over h iterations.

for $i = 0, 1, \ldots, h$ **do**

1. Multiset-label determination

  if $i = 0$ **then**

    Set $M_i(v) := l_0(v) = \ell(v)$.

  else

    Assign multiset-label $M_i(v)$ to each each node v in G and G' consisting of the multiset $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$.

  end if

2. Sorting the multisets

  Sort elements in M_i in ascending order and concatenate into a string $s_i(v)$.

  Add l_{i-1} as a prefix to $s_i(v)$ and call the resulting string $s_i(v)$.

3. Label Compression

  Sort all of the strings $s_i(v)$ for all v from G to G' in ascending order.

  Map each string $s_i(v)$ to a new label, using the function $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ iff $s_i(v) = s_i(w)$
  (see below).

4. Relabelling

  Set $l_i(v) := f(s_i(v))$ for all nodes in G and G'.

5. Termination

  if $\{l_i(v) | v \in V\} \neq \{l_i(v') | v' \in V'\}$ **then**

    return NOT ISOMORPHIC

    terminate

  end if

end for

function $f(s_i(v))$

  Keep a counter variable, c_f, for f which records number of *distinct* strings compressed before. Keep a database of these strings.

  if String identical to $s_i(v)$ has been compressed before **then**

    return c_f

  else

    $c_f = c_f + 1$

    return c_f

  end if

end function

the l_i are the assigned vertex labels after the ith iteration of the WL graph isomorphism test, Alg. 5. Note that the topology of each G_i remains the same, only the vertex labels change. We can now define the WL kernel, with a *base kernel* κ and h iterations as

$$\kappa_{\mathrm{WL}}^{(h)}(G, G') = \kappa(G_0, G'_0) + \kappa(G_1, G'_1) + \cdots + \kappa(G_h, G'_h). \tag{3.4.9}$$

The WL-kernel $\kappa_{\mathrm{WL}}^{(h)}$ is p.s.d. provided that the base kernel is p.s.d. The most natural instance of the WL kernel is the *WL subtree kernel* [37], defined as follows:

$$\kappa_{\text{WL-subtree}}^{(h)}(G, G') = \langle \phi_{\text{WL-subtree}}^{(h)}(G), \phi_{\text{WL-subtree}}^{(h)}(G) \rangle. \tag{3.4.10}$$

We now need to specify the mapping $\phi_{\text{WL-subtree}}^{(i)}$ into the RKHS. First, let $\Sigma_i = \{\sigma_{i1}, \ldots, \sigma_{i|\Sigma_i|}\}$ as the (ordered) set of letters that occur at least once as vertex labels at the end of the ith iteration of Alg. 5, with Σ_0 containing the original node labels. We now define the map $c_i : \{G, G'\} \times \Sigma_i \to \mathbb{N}$ such that $c_i(G, \sigma_{ij})$ is how many times the letter σ_{ij} appears in the graph G. We can now describe the mapping $\phi_{\text{WL-subtree}}^{(h)}$:

$$\phi_{\text{WL-subtree}}^{(h)}(G) = (c_0(G, \sigma_{01}), \ldots, c_0(G, \sigma_{0|\Sigma_0|}), \ldots, c_h(G, \sigma_{h1}), \ldots, c_h(G, \sigma_{h|\Sigma_h|}))^{\mathsf{T}}. \tag{3.4.11}$$

Intuitively, this means for each graph recording the number of occurrences of each unique label in what is essentially a histogram vector. Combined with Eq. 3.4.10, we now have the procedure for computing the WL subtree kernel. An illustration of one step of this calculation is shown in Fig. 3.4. Computing the kernel matrix on N graphs takes time $O(Nhm + N^2hn)$, where h is the depth of the kernel, and the maximum number of vertices and edges are n and m respectively.

3.4.3 Quantum Graph Kernels in the Literature

The contemporary notion of a quantum graph kernel is a graph kernel, computed using classical means, whose definition is inspired by the apparatus of quantum information theory. The current state of the art is *Quantum Jensen-Shannon* (QJS) graph kernels [38], inspired by similar kernels using discrete [39] and continuous [40] quantum walks. We now describe the formulation of QJS graph kernels, following [38].

QJS graph kernels utilise continuous quantum walks, as discussed in Sec. 2.2, with an important distinction. The state space is still defined over the vertices of the graph, $G(V, E)$, but the Hamiltonian used is the *graph Laplacian*, L, defined as $D - A$, where $D = \mathrm{diag}(d(v_1), \ldots, d(v_n))$ is the diagonal matrix of vertex degrees and A is the adjacency matrix of G. Thus, returning to the Dirac notation, the state $|\psi_t\rangle = \sum_{v \in V} \alpha_v(t) |v\rangle$ encoding the state of the continuous walk at time t is given by

$$|\psi_t\rangle = \mathrm{e}^{-\mathrm{i}Lt} |\psi_0\rangle , \tag{3.4.12}$$

where $|\psi_0\rangle$ encodes the starting state (i.e. complex amplitudes over the vertices) and the coefficients chosen to be $\alpha_v(0) = \sqrt{\frac{d(v)}{2|E|}}$. This kernel uses the *density matrix* formalism for describing quantum systems, where states are represented as a probabilistic mixture over projectors onto

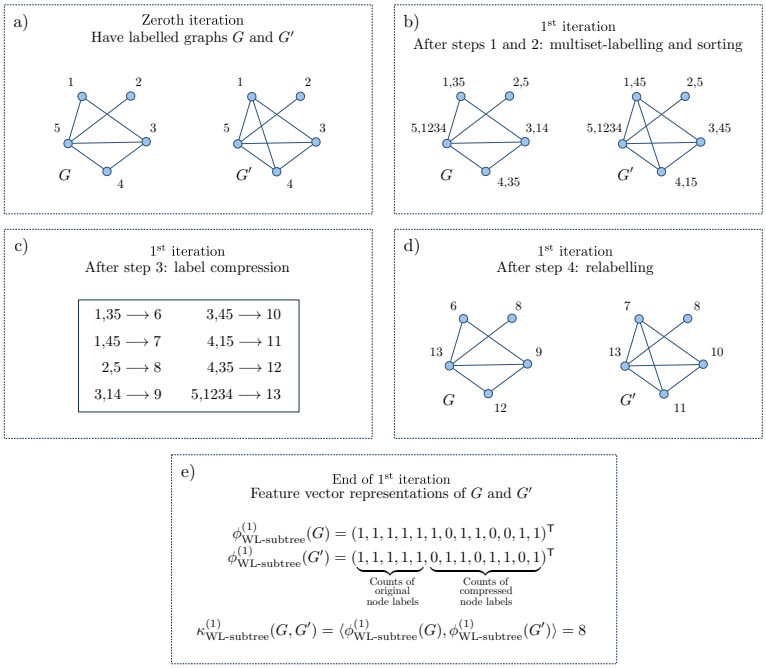

Figure 3.4: Illustration of the calculation of the WL subtree kernel between two graphs with $h = 1$. Here, the alphabet $\Sigma = \{1, 2, \ldots, 13\}$. We note that the vertex labels $\ell \in \Sigma$ are distinct from the vertex set V itself.

pure states, $|\psi_i\rangle$. A general density matrix, ρ, is described as follows:

$$\rho = \sum_i p_i \, |\psi_i\rangle\langle\psi_i| \,, \tag{3.4.13}$$

where the $\{p_i\}$ constitute a valid probability distribution. We are interested in the time-averaged density matrix of our quantum walk, given by

$$\rho_G^t = \frac{1}{t} \int_0^t |\psi_t'\rangle\langle\psi_t'| \, \mathrm{d}t' = \frac{1}{t} \int_0^t \mathrm{e}^{-\mathrm{i}Lt} \, |\psi_0\rangle\langle\psi_0| \, \mathrm{e}^{\mathrm{i}Lt} \, \mathrm{d}t' \,. \tag{3.4.14}$$

We now express the matrix L in terms of its spectral decomposition $\Phi\Lambda\Phi$, where $\Lambda = (\lambda_1, \ldots, \lambda_{|V|})$ is a diagonal matrix with the ordered eigenvalues as elements $(\lambda_1 < \lambda_2 < \cdots < \lambda_{|V|})$ and $\Phi = (\boldsymbol{\phi}_1 \; \boldsymbol{\phi}_2 \; \cdots \; \boldsymbol{\phi}_{|V|})$ is the matrix with the corresponding eigenvectors $\boldsymbol{\phi}_i$ as columns. We can now express Eq. 3.4.14 as

$$\rho_G^t = \frac{1}{t} \int_0^t \Phi^\intercal \mathrm{e}^{-\mathrm{i}\Lambda t'} \Phi \, |\psi_0\rangle\langle\psi_0| \, \Phi^\intercal \mathrm{e}^{\mathrm{i}\Lambda t'} \Phi \, \mathrm{d}t' \,. \tag{3.4.15}$$

Taking Φ_{ij} as the (i,j)-th element of Φ and $\bar{\psi}_i$ as the ith element of $\Phi^\intercal \, |\psi_0\rangle$, the elements of ρ_G^t can (after some algebra) be written as

$$[\rho_G^t]_{ij} = \sum_{k=1}^n \sum_{l=1}^n \Phi_{ik} \Phi_{jl} \bar{\psi}_k \bar{\psi}_l \frac{1}{t} \int_0^t \mathrm{e}^{\mathrm{i}(\lambda_l - \lambda_k)t'} \, \mathrm{d}t' \,. \tag{3.4.16}$$

Letting $t \to \infty$, we can simplify Eq. 3.4.16 to

$$[\rho_G^\infty]_{ij} = \sum_{\lambda \in \tilde{\Lambda}} \sum_{k \in B_\lambda} \sum_{l \in B_\lambda} \Phi_{ik} \Phi_{jl} \bar{\psi}_k \bar{\psi}_l, \tag{3.4.17}$$

where $\tilde{\Lambda}$ is the set of distinct eigenvalues of the Laplacian matrix L and B_λ is a basis of the eigenspace associated with λ.

Another ingredient required for the QJS graph kernel is the *quantum Jensen-Shannon divergence*, derived from the classical Jensen-Shannon divergence [41]. The quantum Jensen-Shannon divergence D_{QJS} between density matrices ρ and σ is defined as follows:

$$D_{\mathrm{QJS}}(\rho, \sigma) = S\left(\frac{\rho + \sigma}{2}\right) - \frac{1}{2} S(\rho) - \frac{1}{2} S(\sigma), \tag{3.4.18}$$

where $S(\,\cdot\,)$ is the Von-Neumann entropy, defined by

$$S(\rho) = -\operatorname{Tr}(\rho \log \rho) = -\sum_i \xi_i \log \xi_i, \tag{3.4.19}$$

where ξ_i are the eigenvalues of ρ. The quantum Jensen-Shannon divergence is proven to always be well-defined, symmetric, negative-definite and bounded, i.e. $0 \leq D_{\mathrm{QJS}} \leq 1$ [42]. The measure D_{QJS} serves as a good measure of distance between quantum states as it is bounded, constitutes a statistical distance and satisfies the triangle inequality. However, these properties have been formally proved only for pure states, while strong empirical evidence is shown for mixed states.

There are two types of QJS kernels. The first, known as the *unaligned* QJS graph kernel, κ_{QJSU} between graphs G_a and G_b, is defined as

$$\kappa_{\mathrm{QJSU}}(G_a, G_b) = \exp\big(-\mu D_{\mathrm{QJS}}(\rho_{G_a}^\infty, \rho_{G_b}^\infty)\big), \tag{3.4.20}$$

where D_{QJS} is as defined in Eq. 3.4.18, ρ_G^∞ as in Eq. 3.4.17 and μ is a decay factor such that $\mu \in (0, 1]$. This kernel ignores vertex correspondence information between G_a and G_b, thus it is not permutation invariant up to vertex order. To mitigate this, the authors of [38] define the *aligned* kernel, κ_{QJSA}:

$$\kappa_{\mathrm{QJSA}} = \max_{Q \in \Sigma} \exp\big(-\mu D_{\mathrm{QJS}}\big(\rho_{G_a}^\infty, Q \rho_{G_b}^\infty Q^{\mathsf{T}}\big)\big) \tag{3.4.21}$$

$$= \exp\left(-\mu \min_{Q \in \Sigma} D_{\mathrm{QJS}}\big(\rho_{G_a}^\infty, Q \rho_{G_b}^\infty Q^{\mathsf{T}}\big)\right), \tag{3.4.22}$$

where Σ is the set of state permutations. This added alignment step optimally aligns the density matrices, i.e. minimises their distance from one another such that more fine-grained discrimination between graph topologies is allowed. Computing κ_{QJSA} is computationally hard due to the minimisation of Von-Neumann entropies involved. This process is approximated by minimising the Frobenius L_2 distance instead of the Von-Neumann entropy using the Umeyama method [43]. The computational complexity for a kernel matrix of size $N \times N$ for κ_{QJSU} and κ_{QJSA} is $O(Nn^3 + N^2 n)$ and $O(Nn^3 + N^2 n^4)$ respectively, for graphs with maximum n vertices.

The classification accuracy of both kernels is competitive with state of the art classical kernels such as the WL-subtree kernel and the random walk kernel, as is the efficiency.

3.5 Hardness of Computing Graph Kernels

Here we show a result from [31] (which we essentially quote verbatim), relating to the hardness of computing graph kernels.

Definition 3.5.1. Let $\mathcal{G}$ be the set of all graphs and $\phi : \mathcal{G} \mapsto \mathcal{H}$ be a map from this set into a Hilbert space $\mathcal{H}$. Further to this, let $k : \mathcal{G} \times \mathcal{G} \mapsto \mathbb{R}$ be such that $\langle \phi(G), \phi(G') \rangle = k(G, G')$. If ϕ is injective (one-to-one), k is called a *complete* graph kernel.

Proposition 3.5.1. *Computing any complete graph kernel is at least as hard as deciding whether two graphs are isomorphic.*

Proof. Let all functions be as in Def. 3.5.1. As ϕ is injective, $k(G, G') - 2k(G, G') + k(G, G') = \langle \phi(G) - \phi(G'), \phi(G) - \phi(G') \rangle = 0$ if and only if $G \cong G'$ □

Thus computing a complete graph kernel is NP-intermediate, as with graph isomorphism. This shows us an interesting relation between graph similarity and isomorphism. Also, observe that one can sacrifice accuracy (the injectivity of ϕ) in return for more efficient computation. How much one is willing to accept is for the designer of the graph kernel to decide.

Chapter 4

Results

In Sec. 4.1, we detail efforts to construct a quantum-inspired isomorphism test, based on the average mixing matrix of Def. 2.2.2. In Sec. 4.2 we present quantum-inspired graph kernels based on the isomorphism test.

4.1 Quantum-Inspired Isomorphism Tests

4.1.1 Quantum Weisfeiler-Lehman Isomorphism Test

We now define a 'quantum-inspired' version of the WL graph isomorphism test (Alg. 5) and provide some motivation for its definition. Pseudocode for the quantum WL (QWL) isomorphism test is shown in Alg. 6. We see that the only difference between the quantum WL isomorphism

Algorithm 6 Quantum WL Isomorphism Test

1: Calculate the average mixing matrices $\widehat{M}_G$ and $\widehat{M}_{G'}$ for the graphs being compared, G and G', as defined in Def. 2.2.2.
2: Label each vertex v with $\mathbf{u}_v$, where $\mathbf{u}_v$ is the row of $\widehat{M}_G$ corresponding to vertex v. Do likewise for G'.
3: Convert each $\mathbf{u}_v$ into a string $s_0(v)$.
4: Map each string $s_0(v)$ to a new label, using the function $f : \Sigma^* \to \Sigma$ such that $f(s_0(v)) = f(s_0(w))$ iff $s_0(v) = s_0(w)$ (see Alg. 5 for form of f).
5: Set $l_0(v) := f(s_0(v))$ for all nodes in G and G'.
6: **if** $\{l_0(v)|v \in V\} \neq \{l_0(v')|v' \in V'\}$ **then**
7: **return** NOT ISOMORPHIC
8: **terminate**
9: **else**
10: Run classical WL isomorphism test (Alg. 5) starting from $h = 1$.
11: **end if**

test and classical WL test is in the initial vertex labelling. The reasoning for applying this labelling is as follows:

In the classical WL test (in the unlabelled vertex case), the initial labels chosen are the node degrees $d(v)$. Recall from Section 2.1 that the limiting distribution for a classical random walk on a graph is given by $[\boldsymbol{\pi}]_v = d(v)/2|E|$. Thus by labelling each vertex with its degree, we are

imparting information about the dynamics of a classical walker on the graph of interest. The classical WL algorithm acts as a comparison between the limiting dynamics of a classical walker on the graphs to be compared.

The natural extension to comparison of quantum dynamics is given by labelling each vertex with its corresponding row in the average mixing matrix, $\widehat{M}_G$. This is because the elements of each row represent the limiting probability of a quantum walk starting at the corresponding vertices finishing at the particular vertex of interest, thus the rows of $\widehat{M}_G$ in some sense act as a generalisation of the vertex degrees. As such, the quantum WL test should act as a refinement of the classical WL test.

In order to confirm that this intuition is correct, we wish to compare the ability of the quantum and classical WL isomorphism tests to differentiate isomorphic graphs. It is well known that the WL test can not differentiate between d-regular graphs. This is because all the vertex labels are the degree d, so the initial labelling is the same. Then, as the vertices are re-labelled for $h > 0$, the vertex labels still stay the same for every vertex, simply being d copies of the previous iteration's vertex labels. Thus, for any h, the WL algorithm fails to distinguish between d-regular graphs.

We can think about a corresponding result for the quantum case. Since the vertex labels are the rows of the average mixing matrix, two graphs may prove to be indistinguishable if their average mixing matrices are the same, we show a case in which this is true.

Lemma 4.1.1. *The average mixing matrix of a d-regular graph G and its complement*[1] *$\bar{G}$ are equal, if and only if both G and $\bar{G}$ are connected.*

Proof. From Eq. 2.2.10, the average mixing matrix of two graphs is the same if the the adjacency matrices share the same orthogonal projections onto their respective eigenspaces E_r, i.e. they are *simultaneously diagonalisable*. Two symmetric matrices A and B have simultaneous diagonalisation iff they commute, i.e. $AB = BA$. The adjacency matrix of the complement of G, $A(\bar{G}) := \bar{A}$ is $J - I - A(G)$, where J is the all-ones matrix. We then have

$$\begin{aligned} A\bar{A} - \bar{A}A &= A(J - I - A) - (J - I - A)A, \\ &= AJ - AI - A^2 - JA + IA + A^2, \\ &= AJ - JA, \\ &= dJ - dJ = 0, \end{aligned}$$

for d-regular G. Note that since any symmetric matrix has orthogonal diagonalisation [44, Thm 4.4.7] this means that the eigenvectors will be the same for d-regular G and its complement. However, the eigenspaces won't be the same in general for G and $\bar{G}$ as the eigenvalues and their multiplicities will be different. In [14] it is shown that the E_r in general for d-regular G and its complement are only the same if both G and $\bar{G}$ are also connected, thus giving $\widehat{M}_G = \widehat{M}_{\bar{G}}$. □

We show in Fig. 4.1 the graphs with up to 8 vertices which exhibit this feature of being regular connected, with (regular) connected complement.

[1] The complement of a graph G is a graph $\bar{G}$ with the same vertices of G and edge set such that two distinct vertices of $\bar{G}$ are adjacent if and only if they are not adjacent in G.

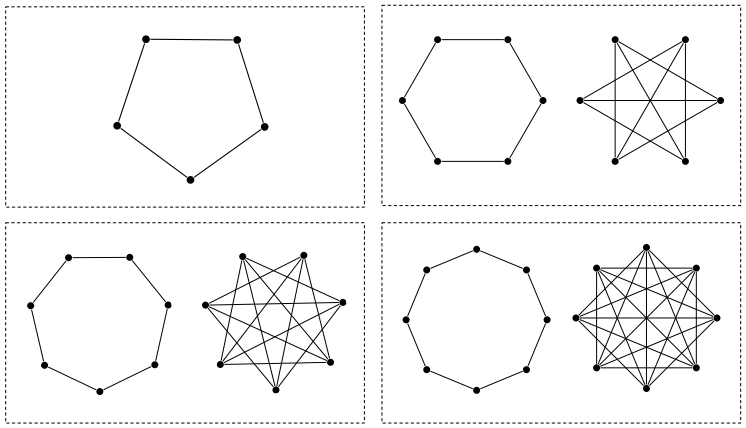

Figure 4.1: All of the graphs G up to 8 vertices for which G and $\bar{G}$ are both regular and connected, found by numerical search. Thus their average mixing matrices $\widehat{M}_G$ and $\widehat{M}_{\bar{G}}$ are identical, according to Lem. 4.1.1. Note that for 5 vertices, the graph G is self-complementary, i.e. $G = \bar{G}$.

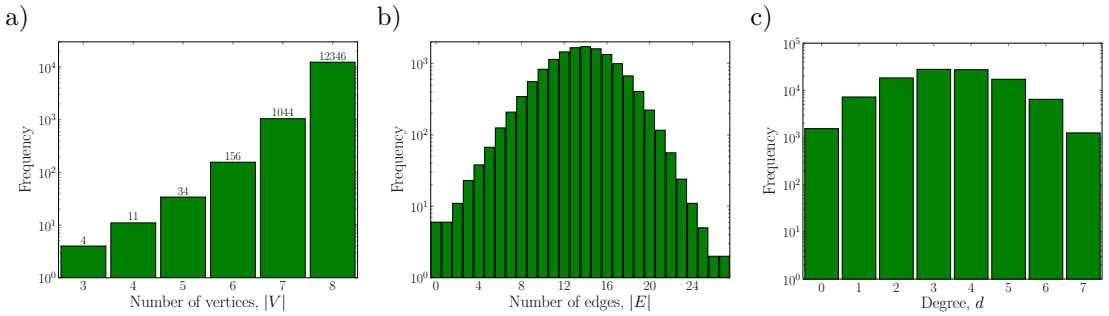

Figure 4.2: Statistics for all of the graphs on 3-8 vertices, generated by the `geng` program, for comparison of quantum and classical WL isomorphism tests. a) Distribution of size of vertex set. b) Distribution of size of edge set. c) Distribution of degrees of vertices.

A more systematic treatment is desirable if we wish to show that the quantum WL isomorphism test is more effective than its classical counterpart. We achieve this by taking a large set of non-isomorphic graphs and separating them into *equivalence classes*; that is, sets of graphs which aren't differentiated by a given algorithm. This was achieved using the `geng` program included in the `nauty` package [45]. The program `geng` generates all of the non-isomorphic graphs for a given number of vertices and of the generated graphs, 89% are connected.

We used `geng` to generate 13,595 such graphs with $n = \{3, 4, \ldots, 8\}$ vertices. Statistics on the graphs (node, edge and degree distributions) are shown in Fig. 4.2. Tab. 4.1 shows the number of equivalence classes found by WL for a varying number of iterations, h. We note that in general, as h increases, the number of equivalence classes increases. Also, as the number of vertices n increases the ratio of equivalence classes to total number of graphs decreases. As graphs get larger, the classical WL test gets worse as differentiating between isomorphic graphs.

As a contrast to this, for all $n = \{3, 4, \ldots, 8\}$ vertex graphs and for all $h = \{0, 1, \ldots, 5\}$, the

	$h=0$	$h=1$	$h=2$	$h=3$	$h=4$	$h=5$	Total # of graphs
$n=3$	4	4	4	4	4	4	4
$n=4$	11	11	11	11	11	11	11
$n=5$	31	34	34	34	34	34	34
$n=6$	102	152	152	152	152	152	156
$n=7$	342	952	1,018	1,022	1,022	1,022	1044
$n=8$	1,213	9,863	11,943	12,087	12,095	12,095	12,346

Table 4.1: Counts for number of equivalence classes found in the generated graph dataset by the classical WL isomorphism test over a range of number of iterations h and number of vertices n. Only up to $n=5$ vertices does the WL isomorphism test differentiate between all graphs in the dataset.

n, $(h=0)$	WL # classes	QWL # classes	Total # graphs	# non-isomorphic graphs
3	4	6	8	4
4	11	22	33	11
5	31	93	136	34
6	102	561	780	156
7	342	5105	6264	1044
8	1,213	75,783	86422	12346

Table 4.2: Number of equivalence classes produced by the WL and QWL isomorphism tests over the dataset now containing isomorphic graphs, with $h=0$ iterations in both cases.

quantum WL test distinguished between *every* graph in the dataset, making it more effective than the classical algorithm. This shows naïvely that the quantum WL test is more effective at differentiating between non-isomorphic graphs.

With the precision of the quantum WL appearing to be so high, it naturally begs the question of whether the quantum WL is perhaps too refined; that is, will it place in separate equivalence classes graphs which *are* isomorphic? To test this, we took the graph dataset generated by `geng` and for each graph on n vertices, generated $n-1$ graphs isomorphic to the original. Recall that graphs G and H are isomorphic (denoted by $G \cong H$) if their adjacency matrices are permutations of one another, i.e.

$$G \cong H \quad \text{if} \quad PA(G)P^{\mathsf{T}} = A(H), \tag{4.1.1}$$

where P is some permutation matrix. The isomorphic graphs were generated by instantiating $n-1$ random permutation matrices for graphs G and acting upon the adjacency matrices $A(G)$ as in Eq. 4.1.1 with these permutations. Recall that there are $n!$ total permutations for n labelled objects. The number of permutations generated, $n-1$, was chosen as a demonstrative example not requiring too much computational time.

Results from this analysis are shown in Tab. 4.2. We see that for classical WL, the results are consistent with those seen in Tab. 4.1, i.e. it cannot differentiate between all of the non-isomorphic graphs, but isomorphic graphs are assigned to the same equivalence class. We see that in the case of QWL, the equivalence classes are over-determined. All non-isomorphic graphs

are in different classes, but most isomorphic graphs are also in different classes. To explain why this is, we first prove Lem. 4.1.2 and Thm. 4.1.3.

Lemma 4.1.2. *For isomorphic graphs G and H, their spectrum is the same, i.e. their adjacency matrices have the same eigenvalues.*

Proof. Let $A(G)$ be the adjacency matrix of G and $A(H)$ that of H. The characteristic equation for $A(H)$ is as follows:

$$\begin{aligned}\det(A(H)-\lambda I) &= \det\big(PA(G)P^{\mathsf{T}}-\lambda\big),\\ &= \det\big(PA(G)P^{\mathsf{T}}-P(\lambda I)P^{\mathsf{T}}\big),\\ &= \det\big(PA(G)P^{\mathsf{T}}-\lambda IP^{\mathsf{T}}\big),\\ &= \det(P)\det(A(G)-\lambda I)\det\big(P^{\mathsf{T}}\big),\\ &= \det(A(G)-\lambda I),\end{aligned}$$

where we have used the identities $PP^{\mathsf{T}}=I$ and $\det(P)=\det\big(P^{\mathsf{T}}\big)=1$ for a permutation P. Since the characteristic equations for $A(H)$ and $A(G)$ are the same, their eigenvalues are the same. □

Theorem 4.1.3. *For isomorphic graphs G and H, with P a permutation matrix such that $A(H)=PA(G)P^{\mathsf{T}}$, the average mixing matrices $\widehat{M}_G$ and $\widehat{M}_H$ are related in the following way:*

$$\widehat{M}_H = P\widehat{M}_G P^{\mathsf{T}}.$$

Proof. The spectral decomposition of $A(G)$ is $\sum_r \lambda_r E_r$, where λ_r are its eigenvalues and E_r the corresponding idempotents. We can therefore express $A(H)$ as

$$A(H) = P\sum_r \lambda_r E_r P^{\mathsf{T}} = \sum_r \lambda_r P E_r P^{\mathsf{T}}.$$

By Lem. 4.1.2, we know the eigenvalues of $A(G)$ and $A(H)$ are the same, thus the idempotents of $A(H)$ corresponding to λ_r are PE_rP^{T}. Using Eq. 2.2.10, we express the average mixing matrix of H as $\widehat{M}_H=\sum_r\left(PE_rP^{\mathsf{T}}\right)^{\circ 2}$. We can express $\widehat{M}_H$ elementwise as

$$\left[\widehat{M}_H\right]_{il} = \left[\sum_r\left(PE_rP^{\mathsf{T}}\right)^{\circ 2}\right]_{il} = \sum_r\sum_{j,k}\left(P_{ij}[E_r]_{jk}[P^{\mathsf{T}}]_{kl}\right)^2 = \sum_r\sum_{j,k}P_{ij}[E_r]^2_{jk}[P^{\mathsf{T}}]_{kl}, \quad (4.1.2)$$

since the P_{ij} take the values 0 or 1, so $P_{ij}^2=P_{ij}$. We recognise Eq. 4.1.2 as the elements of $\sum_r PE_r^{\circ 2}P^{\mathsf{T}}$, giving

$$\widehat{M}_H = \sum_r PE_r^{\circ 2}P^{\mathsf{T}} = P\left(\sum_r E_r^{\circ 2}\right)P^{\mathsf{T}} = P\widehat{M}_G P^{\mathsf{T}},$$

where we have used the representation of Eq. 2.2.10 for $\widehat{M}_G$. □

Thus we see why the equivalence classes for the QWL test are over-determined on this dataset.

Since the initial vertex labels assigned are the rows of the average mixing matrix $\widehat{M}_G$, and the rows of $\widehat{M}_G$ are in general different for isomorphic graphs, the label multisets of isomorphic graphs will be different and thus the QWL test will deem the graphs to not be isomorphic. The classical WL test doesn't run into this problem as the multiset of vertex degrees is invariant with respect to permutations of the vertices.

4.1.2 Quantum Shannon WL Isomorphism Test

We require an initial labelling of graph vertices for the WL algorithm which uses the rows of $\widehat{M}_G$, retaining more density of information about a given graph than the vertex degrees but is the same for the vertices of isomorphic graphs. A property or function of a graph which has this property of invariance under vertex relabelling is known as a *graph invariant* [46, 47]. We propose to take the Shannon entropies of the rows of $\widehat{M}_G$ as vertex labels, where the Shannon entropy H of a probability distribution $\mathbf{p}$ is defined as

$$H(\mathbf{p}) = -\sum_i [\mathbf{p}]_i \log[\mathbf{p}]_i, \tag{4.1.3}$$

with $[\mathbf{p}]_i \log[\mathbf{p}]_i = 0$ for $[\mathbf{p}]_i = 0$. The Shannon entropy is always positive and acts as a useful measure of statistical uncertainty or information of a distribution. A desirable property of the Shannon entropy of a particular row of $\widehat{M}_G$ are that it doesn't depend on a random variable explicitly as with other statistical measures, so can be applied to the probability distribution itself. Another desirable property is that it maps the vertex of a graph to a label in $\mathbb{R}_{\geq 0}$ rather than $\mathbb{Z}_{\geq 0}$, thus generalising the degree as a label.

The WL isomorphism test declares two graphs to be not isomorphic if the multisets containing their vertex labels for a given iteration are unequal. We prove that the multisets of Shannon entropies of the rows of the average mixing matrix, $\{H([\widehat{M}_G]_{i*}) : v_i \in V\}$, are the same for isomorphic graphs, which means the WL algorithm will assign isomorphic graphs to the same equivalence class if we use the entropies as initial vertex labels. We call this scheme Quantum Shannon WL (QSWL) and define the corresponding isomorphism test in Alg. 7.

Theorem 4.1.4. *For two graphs G and G' such that $G \cong G'$ with vertex sets V and V', the multisets $\{H([\widehat{M}_G]_{i*}) : v_i \in V\}$ and $\{H([\widehat{M}_{G'}]_{i*} : v_i \in V')\}$ are equal, where $[\widehat{M}_G]_{i*}$ denotes the ith row of $\widehat{M}_G$.*

Proof. By Thm. 4.1.3, the elements of $\widehat{M}_{G'}$ are given by

$$\begin{aligned}
[\widehat{M}_{G'}]_{il} &= [P\widehat{M}_G P^{\mathsf{T}}]_{il}, \\
&= \sum_{j,k} \delta_{\pi(i),j} [\widehat{M}_G]_{jk} \delta_{k,\pi(l)}, \\
&= [\widehat{M}_G]_{\pi(i)\pi(l)},
\end{aligned}$$

where P is the matrix corresponding to a permutation, $\left(\begin{smallmatrix} 1 & 2 & \cdots & n \\ \pi(1) & \pi(2) & \cdots & \pi(n) \end{smallmatrix}\right)$, of the rows of I. The

	$h=0$	$h=1$	$h=2$	$h=3$	$h=4$	$h=5$	Total # of unique graphs
$n=3$	4	4	4	4	4	4	4
$n=4$	11	11	11	11	11	11	11
$n=5$	34	34	34	34	34	34	34
$n=6$	147	156	156	156	156	156	156
$n=7$	1,020	1,044	1,044	1,044	1,044	1,044	1,044
$n=8$	12,166	12,340	12,340	12,340	12,340	12,340	12,346

Table 4.3: Number of equivalence classes produced by the QSWL isomorphism test.

entropy of the ith row of $\widehat{M}_{G'}$ is given by

$$\begin{aligned} H([\widehat{M}_{G'}]_{i*}) &= H([\widehat{M}_G]_{\pi(i)\pi(*)}) \\ &= -\sum_l [\widehat{M}_G]_{\pi(i)\pi(l)} \log [\widehat{M}_G]_{\pi(i)\pi(l)} \\ &= -\sum_l [\widehat{M}_G]_{\pi(i)l} \log [\widehat{M}_G]_{\pi(i)l} \\ &= H([\widehat{M}_G]_{\pi(i)*}), \end{aligned} \tag{4.1.4}$$

where we have used Eq. 4.1.3 and the commutativity of addition. The sets $\{i\}$ and $\{\pi(i)\}$ for $i = \{1, 2, \ldots, n\}$ are equal, by the definition of a permutation, so the multisets $\{H([\widehat{M}_G]_{i*}) : v_i \in V\}$ and $\{H([\widehat{M}_G]_{\pi(i)*}) : v_i \in V\}$ are equal. From Eq. 4.1.4, we then get $\{H([\widehat{M}_G]_{i*}) : v_i \in V\} = \{H([\widehat{M}_{G'}]_{i*}) : v_i \in V'\}$. □

Algorithm 7 Quantum Shannon WL Isomorphism Test

1: Calculate the average mixing matrices $\widehat{M}_G$ and $\widehat{M}_{G'}$ for the graphs being compared, G and G', as defined in Def. 2.2.2.
2: Label each vertex v with $H(\mathbf{u}_v)$, where $\mathbf{u}_v$ is the row of $\widehat{M}_G$ corresponding to vertex v and $H(\,\cdot\,)$ is the Shannon entropy (Eq. 4.1.3). Do likewise for G'.
3: Convert each $H(\mathbf{u}_v)$ into a string $s_0(v)$.
4: Map each string $s_0(v)$ to a new label, using the function $f : \Sigma^* \to \Sigma$ such that $f(s_0(v)) = f(s_0(w))$ iff $s_0(v) = s_0(w)$ (see Algorithm 5 for form of f).
5: Set $l_0(v) := f(s_0(v))$ for all nodes in G and G'.
6: **if** $\{l_0(v) | v \in V\} \neq \{l_0(v') | v' \in V'\}$ **then**
7: **return** NOT ISOMORPHIC
8: **terminate**
9: **else**
10: Run classical WL isomorphism test (Algorithm 5) starting from $h = 1$.
11: **end if**

We see in Tab. 4.3 that using the entropies of the rows of $\widehat{M}_G$ as initial vertex labels in the WL algorithm leads to superior distinguishing power between graphs, at least up to eight vertices. Note that while the QSWL separates the set of non-isomorphic graphs into more equivalence classes than the classical WL test, the QSWL equivalence classes in the table are not all subsets of the WL equivalence classes. That is, the classical WL algorithm declaring

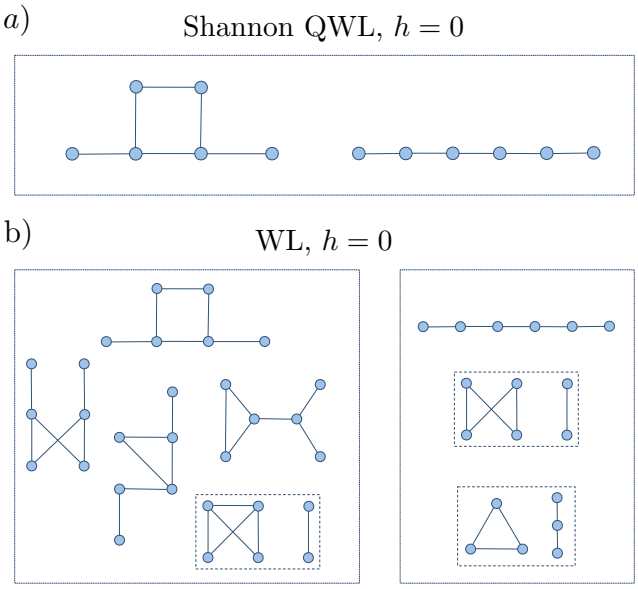

Figure 4.3: Example of a case where isomorphism under WL *doesn't* give isomorphism under QSWL, after the initial labelling at $h = 0$, due the multisets of row entropies of $\widehat{M}_G$ being the same. a) Equivalence class for QSWL. b) Equivalence classes for WL. Note that after many iterations both algorithms distinguish between these graphs.

two graphs to be isomorphic doesn't necessarily mean that the QSWL will declare them to be isomorphic also, *for a given number of iterations*. We see an example of this in Fig. 4.3, where a) shows an equivalence class containing two graphs, defined by the QSWL procedure and b) shows their respective equivalence classes under classical WL, after zero iterations (so just the initial labelling).

In the limit of many iterations, we conjecture the following.

Conjecture 4.1.5. *Any two graphs which can be distinguished by WL in the limit as* $h \to \infty$ *can also be distinguished by QSWL in the limit as* $h \to \infty$.

In Tab 4.3, we see that only in the case of graphs on eight vertices is QSWL unable to distinguish certain graphs, splitting the 12,346 graphs into 12,340 equivalence classes. While in the table this is shown up to $h = 5$, this set of equivalence classes remains as h grows large. We show the equivalence classes, all being pairs of graphs, in Fig. 4.4. It is interesting to think about what marks these graphs out as 'special' according to QSWL. We note that none of these graphs are the cycle of eight vertices or its complement, which Lem. 4.1.1 demands have the same mixing matrix. In this case, the node labels will have been the same at $h = 0$, but at $h = 1$, during relabelling, the fact that the complement of a d-regular graph is $(n-d-1)$-regular ensures that appending the neighbourhood labels for each vertex gives a different multiset of labels for the two graphs. In fact, for all of the graphs in Fig. 4.4, classical WL fails also, none are regular and all are connected. Within each pair, neither are the complement of one another and they have different average mixing matrices, although the multisets of row entropies are the same. Recall (as in Fig. 4.3) that having the same multiset of row entropies between graphs doesn't

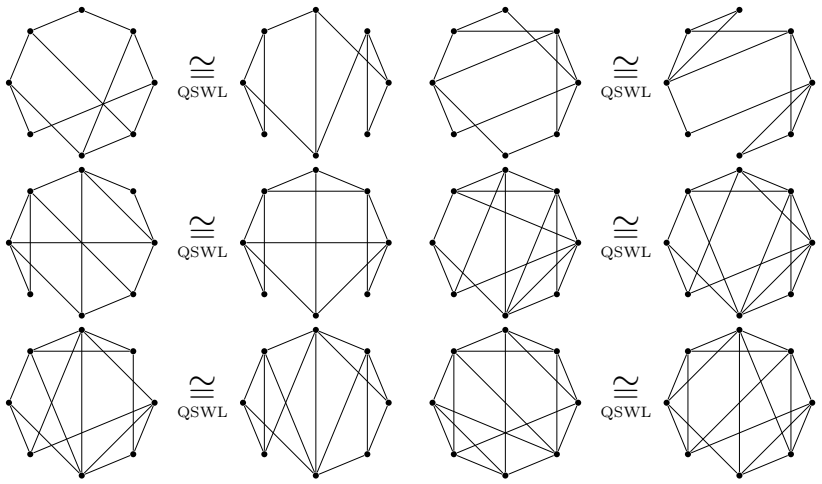

Figure 4.4: The six pairs of non-isomorphic graphs on eight vertices which the QSWL algorithm fails to distinguish.

preclude QSWL failing for arbitrary h, just $h = 0$. We have thus far not found a convincing mathematical explanation for the failure of QSWL in distinguishing pairs of graphs within these six particular equivalence classes.

In spite of this, the strength of QSWL over WL shows promise for quantum-inspired enhancement of classical algorithms for graph matching using the rows of the average mixing matrix with Shannon entropy as a general approach.

4.2 Novel Quantum Graph Kernels

4.2.1 Definitions

We now wish to use the quantum methods which have refined the Weisfeiler-Lehman isomorphism test to define new graph kernels. We define two kernels based on classical kernels with vertex labelling using the entropies of the rows of the average mixing matrix and the Quantum Shannon WL isomorphism test of Alg. 7.

Definition 4.2.1 (Quantum WL kernel)**.** We take the WL subtree kernel of Eq. 3.4.10 and instead of using the vertex degrees as the initial labelling in the unlabelled case, or the given labels otherwise; use the Shannon entropy of the rows of the average mixing matrix. More precisely, for the ith vertex of graph G with vertex set V, the initial vertex label is $H([\widehat{M}_G]_{i*})$. Otherwise the kernel is merely the WL subtree kernel.

Theorem 4.2.1. *The Quantum WL kernel is positive semi-definite.*

Proof. The Quantum WL kernel is merely a different initial labelling scheme for the (classical) WL subtree kernel, which is itself p.s.d. Thus, the quantum variant is p.s.d. □

Definition 4.2.2 (Quantum Shannon Kernel)**.** The Quantum Shannon (QS) kernel is made by explicitly evaluating feature vectors $\boldsymbol{\phi}_G$ for each graph and taking their inner product. The

feature space defined as follows: Choose a number of bins, b, then take the feature space as $\mathbb{R}^b$. Now define some minimum and maximum entropy values H_0 and H_b, such that the sequence $H_0, H_1, \ldots, H_b$ defines the edges of b equally spaced intervals on $\mathbb{R}$. A feature vector $\boldsymbol{\phi}_G$ is constructed by initialising to $\mathbf{0} \in \mathbb{R}^b$. For a graph G, calculate all of the row entropies $\{H([\widehat{M}_G]_{i*}) : v_i \in V\}$. Then for each entropy $H([\widehat{M}_G]_{i*})$ lying in the interval $[H_{j-1}, H_j]$, add $\mathbf{e}_j$ to $\boldsymbol{\phi}_G$.

Thus each feature vector is effectively a histogram and the kernel is created by taking an inner product between these 'histogram' feature vectors.

$$\kappa_{\mathrm{QS}}(G, G') = \langle \boldsymbol{\phi}_G, \boldsymbol{\phi}_{G'} \rangle.$$

Theorem 4.2.2. *The Quantum Shannon Kernel is positive semi-definite.*

Proof. Each feature vector $\boldsymbol{\phi} = \sum_j N_j \mathbf{e}_j$, where $N_j \in \mathbb{Z}_{\geq 0}$. Taking the inner product between two feature vectors, $\boldsymbol{\phi}$ and $\boldsymbol{\phi}'$ gives

$$\begin{aligned}\langle \boldsymbol{\phi}, \boldsymbol{\phi}' \rangle = \left\langle \sum_{j=1}^{m} N_j \mathbf{e}_j, \sum_{k=1}^{m} N'_k \mathbf{e}_k \right\rangle &= \sum_{j,k} N_j N'_k \langle \mathbf{e}_j, \mathbf{e}_k \rangle \\ &= \sum_{j,k} N_j N'_k \delta_{j,k} \\ &= \sum_{j} N_j N'_j,\end{aligned}$$

which is above zero as $N_j, N'_j \geq 0$ by construction. Since $\langle \boldsymbol{\phi}, \boldsymbol{\phi}' \rangle \geq 0$ for all $\boldsymbol{\phi}$, $\boldsymbol{\phi}'$, the QS kernel, $\kappa_{\mathrm{QS}}(G, G')$ is p.s.d. □

Having defined two kernels and showed that they are are p.s.d., it remained to test the kernels.

4.2.2 Experimental Validation

We wish to test the efficacy of these kernels in machine learning tasks. One way of achieving this is to use these kernels is to carry out Support Vector Machine classification on real-world graph data, which is what we do here.

Recall Alg. 4, the hard-margin SVM. We use a modified version of this algorithm, the C-Support Vector Machine (C-SVM). In the hard-margin SVM, the dual solution support vector, $\boldsymbol{\alpha}^*$ has values restricted to $\alpha_i^* \geq 0$. In C-SVM, these values are further restricted to $0 \leq \alpha_i^* \leq C$, effectively 'softening' the margin. The parameter C affects a compromise between the size of the margin between the separating hyperplane and support vectors and the misclassification error. This is illustrated in Fig. 4.5. We implemented the C-SVM algorithm using the `LIBSVM` library [48].

The graph data to which we applied our kernels was MUTAG [49], a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds labeled (i.e. $y_i \in \{-1, +1\}$) according to whether or not they have a mutagenic effect on the Gram-negative bacterium *Salmonella typhimurium*. We show some statistics on MUTAG in Fig. 4.6.

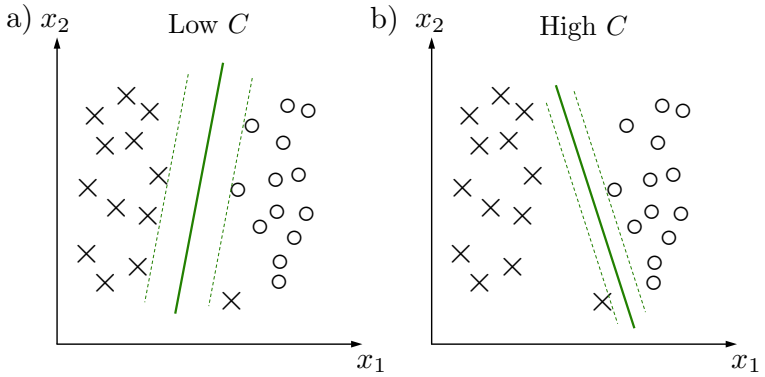

Figure 4.5: Illustration of the effect of different C values on SVM margins and misclassification. A small C as in a) favours a larger margin between the hyperplane and support vectors, sacrificing some classification accuracy on potentially anomalous training data. A large C as in b) increases classification accuracy but decreases the separation in feature space of the classes. The optimal choice of C is contingent on the characteristics of the dataset.

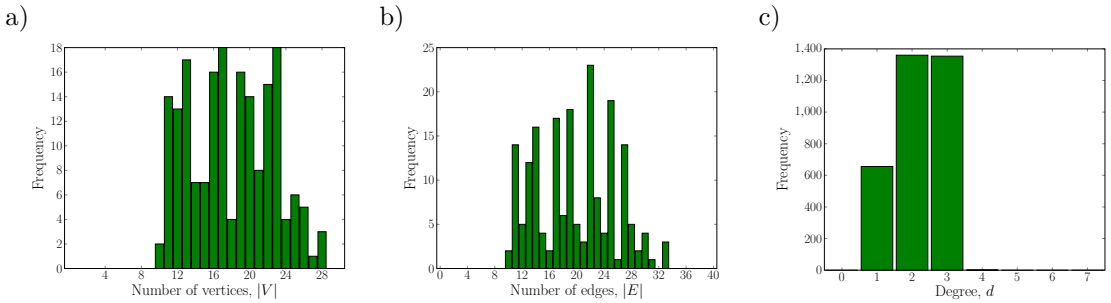

Figure 4.6: Statistics for the MUTAG graph data set. a) Number of vertices $|V|$. b) Number of edges $|E|$. c) Degrees d.

For each test we performed C-SVM classification on the MUTAG database for both quantum kernels and the WL subtree kernel, taking $C = \{10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}, 10^{1}, 10^{3}, 10^{5}, 10^{7}\}/188$ and choosing the C-value giving the best accuracy.

To evaluate the accuracy we used *k-fold cross validation* with 10 folds. What this means in practice is that the data is randomly divided into 10 equal partitions. One of these sets, or folds, is randomly chosen and put to one side. The remaining nine folds are then recombined and randomly partitioned into 10 equal sized folds. The optimisation over C values is carried out by training on 9 of the new folds and classifying on one, which is randomly chosen. Having chosen the best C, *all* of the new folds are used for training and the fold which was placed to one side at the beginning is used to test classification accuracy. This is to ensure no classifying is done on data used for training and to avoid overfitting.

This whole procedure was carried out 10 times, with the mean and standard error on the mean reported as the classification accuracy.

We now present results for the Quantum WL and Quantum Shannon kernel against the classical WL algorithm on MUTAG. We only test against WL as it shows the best performance on MUTAG of the classical kernels [34]. In Fig. 4.7 we see a comparison between the WL and

QWL kernels. We note that the QWL kernel is on average two orders of magnitude slower than WL to compute and apart from the $h = 0$ case, has a lower classification accuracy.

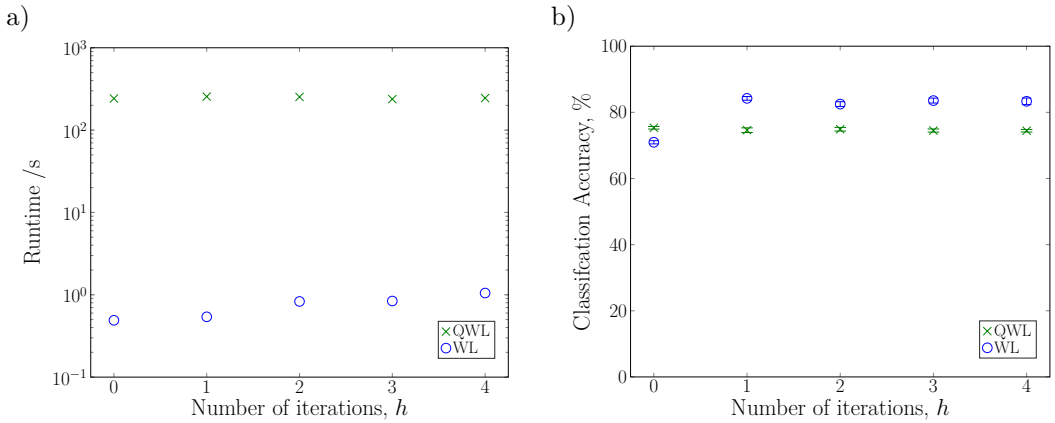

Figure 4.7: Comparison of performance of the (classical) WL subtree kernel with its quantum variant. a) Runtime comparison. b) Accuracy comparison (error bars are standard error on the mean).

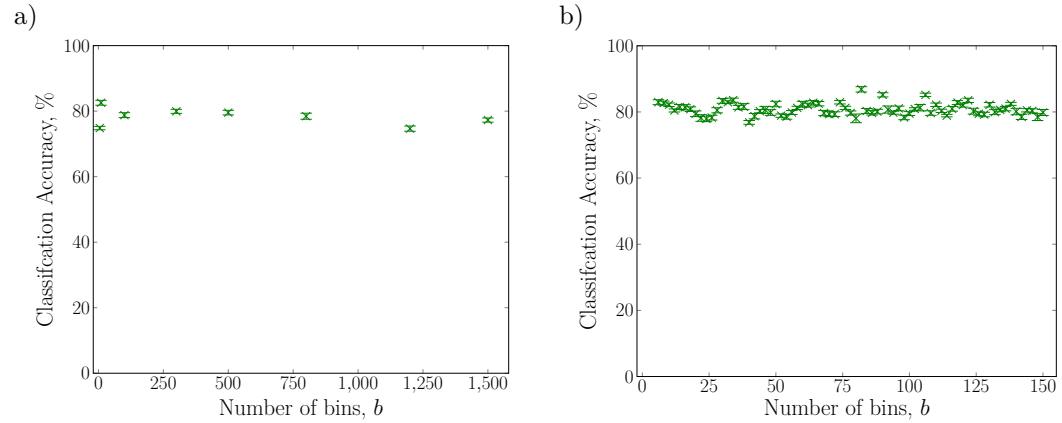

Figure 4.8: Accuracy of Shannon kernel over a varying number of bins, m (error bars are standard error on the mean). a) Coarse range. b) Fine range. The maximum value at $m = 82$ bins is 86.83±0.81%, which is larger than the maximum value for the classical WL kernel, 84.22±0.56%.

In Figs. 4.8 and 4.9 we see accuracy and runtime results for the QS kernel over a range of bin numbers m. Observe that the accuracy is in general on par with classical WL. In fact the maximum classification accuracy for the QS kernel is higher than that of the classical WL, $(86.83 \pm 0.81\%$ vs $84.22 \pm 0.56\%)$. We note that the runtime is consistently lower than that of the classical WL, by almost an order of magnitude.

4.2.3 Discussion

We see that relative to classical WL, QWL performs poorly both in terms of accuracy and runtime. Both of these factors can be explained by the following: In the course of learning on the

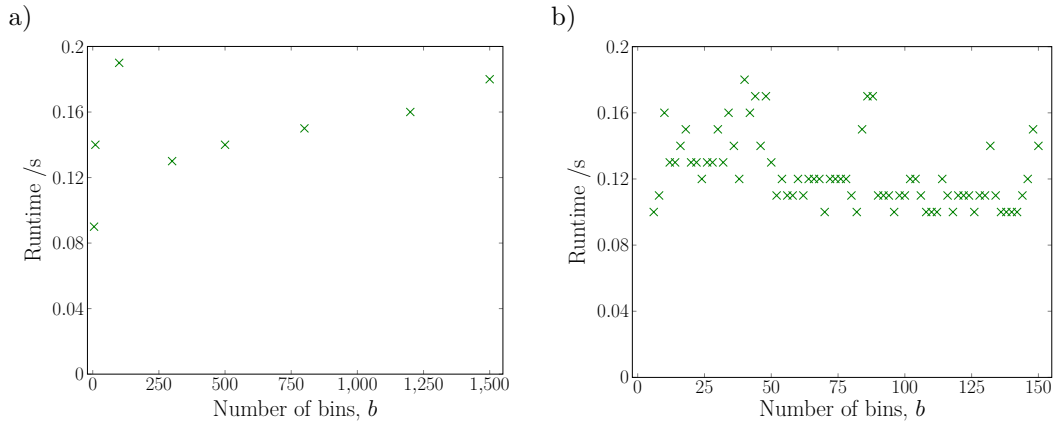

Figure 4.9: Runtime of Shannon kernel over a varying number of bins, m. a) Coarse range. b) Fine range. Note the reduction with respect to the classical and quantum WL algorithms.

MUTAG database[2], the classical WL algorithm generates 174 unique vertex labels. The QWL algorithm generates 2117 unique labels total. For each vertex in the database, its label is checked against every other label in the dataset, thus the increased runtime of QWL is primarily due to the larger number of labels. This larger number of labels (giving more detailed information) almost paradoxically is the reason for the decrease in classification accuracy also. The size of the feature space in both cases is defined as the number of unique labels, with a given graph's feature vector constructed by counting the number of each label's occurrence and setting the corresponding element of the feature vector as this count. Thus the size of the feature space (i.e. length of each vector) in the WL case is 174 and 2,117 in the QWL case. The MUTAG database has a total of 3,371 vertices overall; this means that the feature vectors in the QWL are far more sparse than WL (have a larger percentage of zero-valued elements). Since the kernel is defined as the inner product between feature vectors and the QWL feature vectors are sparse, the resulting kernel matrix is mostly zeros and thus gives little information about the structure of the dataset, reducing the classification accuracy.

What the success of the QS kernel demonstrates is that some element of coarse-graining is required in order to effectively match graph structures in this instance. As seen with the QWL kernel, having a too finely determined delineation between features in a sense 'throws away' information with regards to which class a particular graph belongs. One has to be careful when defining kernels to allow similar objects to have large overlap in feature space, as well as forcing 'different' objects to have small overlap in feature space. The runtime advantage of the QS kernel arises from the fact that there is no need to compare the vertex labels with every other label as with WL and QWL, thus avoiding the search of an increasingly large database with each vertex.

[2]We quote the total number of labels for $h = 2$ iterations as an illustrative example, but the general principle holds for any h.

4.3 Computational Considerations

In this section we consider the computational complexity of the algorithms in this report and discuss their numerical implementation. We make use of the standard 'big-O' notation of computer science, whereby $O(f(x))$ means some function upper-bounded by a constant multiple of $f(x)$. If $f(x)$ is a polynomial, $O(f(x))$ scales as the highest power of x.

4.3.1 Complexity Results

The classical WL isomorphism test (Alg. 5) between two graphs has runtime complexity $O(hm)$, where m is the maximum number of edges and h the number of iterations [34]. This is because the worst case time complexity of any of the steps of Alg. 5 is $O(m)$. There are a constant number of steps per iteration so we have $O(m)$ for each iteration, thus $O(hm)$ over h iterations.

Computing the WL subtree kernel matrix between N graphs over h iterations has complexity $O(Nhm + N^2hn)$, with n as the maximum number of vertices. Evaluating the feature vector $\phi^{(h)}_{\text{WL-subtree}}$ for all N graphs gives a complexity $O(Nhm)$, assuming $m > n$, as computing the feature vector is affected using the WL isomorphism test. The kernel matrix is computed by multiplication of all feature vectors, requiring a runtime of $O(N^2hn)$, as for each graph G there are at most hn non-zero entries in $\phi^{(h)}_{\text{WL-subtree}}(G)$.

In terms of memory requirements, we only need worry about the total number of unique labels, which is upper bounded by $O(hn)$ for the WL isomorphism test and $O(Nhn)$ for the subtree kernel.

Lemma 4.3.1. *Computing the average mixing matrix of a graph with n vertices has complexity $O(n^3)$.*

Proof. This comes straight from the expression for $\widehat{M}_G$ in Eq. 2.2.10, $\widehat{M}_G = \sum_r E_r^{\circ 2}$, where the E_r are orthogonal projections onto the rth eigenspace of $A(G)$. Calculating the E_r is essentially matrix diagonalisation, which is $O(n^3)$. The Schur product requires $O(n^2)$ multiplications. There are $O(n)$ Schur products to compute giving $O(n^3)$ for that step. Finally, there are $O(n)$ additions of matrices size n^2, so we have total runtime $O(n^3 + n^3 + n^3) = O(n^3)$. □

Theorem 4.3.2. *The calculation of the QWL kernel matrix has complexity $O(Nhmn^3 + N^2hn)$.*

Proof. This is the same as for the classical WL kernel matrix, other than the $O(n^3)$ prefactor added to the first term. This comes from the average mixing matrix being used to calculate the initial labelling, in $O(n^3)$, then the $O(n^2)$ operations to calculate the entropies of the rows. These are split into $O(n)$ operations to compute the entropy of a row, with $O(n)$ rows per graph to compute. Again we have $O(n^3 + n^2) = O(n^3)$. Since the WL and QWL kernel algorithms are the same up initial labelling, the complexity is as given. □

Corollary. *The QSWL isomorphism test has runtime complexity $O(n^3hm)$.*

Theorem 4.3.3. *Calculating the QS kernel matrix has complexity $O(Nn^3b + N^2b)$, for b entropy bins and assuming $b > n$.*

Proof. The first term comes from calculating the entropies, $O(n^3)$, for N graphs, then sorting into b bins, which is $O(n+b) = O(b)$ for $b > n$. The second term comes from N^2 inner products between the feature vectors of length b. □

The memory complexity for the quantum-inspired algorithms is similar to the classical cases, although in practice larger.

Theorem 4.3.4. *The QSWL isomorphism test uses memory resources $O(hn+n^2)$ and the QWL graph kernel uses $O(Nhn + Nn^2)$.*

Proof. Count the largest number of unique labels possible, as in the classical case. Then add the number of average mixing matrices stored, each taking $O(n^2)$ resources. □

Corollary. *The QS graph kernel requires $O(Nn^2)$ memory, purely from storing the average mixing matrices.*

4.3.2 Numerical Implementation

All computations unless otherwise specified, were carried out in MATLAB R2015b on a MacBook Pro with a 2.4GHz Intel Core i5 CPU.

Average Mixing Matrix

As stated in Lem. 4.3.1, the runtime complexity of computing the average mixing matrix is $O(n^3)$, for a graph on n vertices. Explicitly, the matrix was computed by diagonalising (using MATLAB's `eig` function), returning its eigenvalues λ_r and corresponding eigenvectors $\mathbf{v}_{\lambda_r}$. The orthogonal projections onto the eigenspace for given λ_r were constructed as follows:

$$E_r = \begin{cases} \mathbf{v}_{\lambda_r} \mathbf{v}_{\lambda_r}^{\mathsf{T}}, & \lambda_r \text{ has multiplicity 1 (non-degenerate);} \\ \displaystyle\sum_{i:\lambda_i=\lambda_r}^{k} \mathbf{v}_{\lambda_i} \mathbf{v}_{\lambda_i}^{\mathsf{T}}, & \lambda_r \text{ has multiplicity } k. \end{cases} \tag{4.3.1}$$

In fact, the first case is merely the second for $k = 1$. We make the distinction as it is computationally faster in practice to establish the multiplicity (or degeneracy) of eigenvalues and treat the degenerate values separately. Since we use the average mixing matrix to compare graphs, it is important to be certain that rotating the basis in which the E_r are represented has no effect on the computations. We thus prove Thm. 4.3.5:

Theorem 4.3.5. *The matrix representation of an orthogonal projection onto a subspace W of the vector space V (over the field $\mathbb{C}$) is independent of which orthonormal basis one chooses for vectors in W.*

Proof. Let U be a unitary matrix such that $UU^\dagger = U^\dagger U = I$, where $(\cdot)^\dagger$ is the Hermitian conjugate. If we choose an orthonormal basis for W, $\{\mathbf{w}_i\}$, then $\{U\mathbf{w}_i\}$ is also an orthonormal

(rotated) basis. We show that the orthogonal projection onto W in both bases is the same:

$$\sum_i \mathbf{w}_i \cdot \mathbf{w}_i^\dagger = I_W, \tag{4.3.2}$$

$$U\left(\sum_i \mathbf{w}_i \cdot \mathbf{w}_i^\dagger\right)U^\dagger = UU^\dagger = I_W,$$

$$\sum_i (U\mathbf{w}_i) \cdot \left(\mathbf{w}_i^\dagger U^\dagger\right) = \sum_i (U\mathbf{w}_i) \cdot (U\mathbf{w}_i)^\dagger = I_W, \tag{4.3.3}$$

where we have the used the fact that the sum of orthogonal projections onto a complete basis forms the identity. Comparing Eqs. 4.3.2 and 4.3.3 we have that $\sum_i \mathbf{w}_i \cdot \mathbf{w}_i^\dagger = \sum_i (U\mathbf{w}_i) \cdot (U\mathbf{w}_i)^\dagger$ for any U and thus the orthogonal projection onto the subspace $W \subseteq V$ is independent of the basis used. □

Using Thm. 4.3.5 we can thus say Eq. 4.3.1 is a valid way of evaluating the average mixing matrix, i.e. the matrix representation doesn't depend on the basis one uses to project onto the eigenspaces.

WL Method

The code for implementing the WL algorithm was adapted from [50]. The label sets in all WL based algorithms in this report were represented by `Map` objects acting as a lookup table with the keys as strings to look up the integer vertex labels. The complexity of the WL subtree kernel is $O(Nhm + N^2hn)$, as discussed in Sec. 4.3.1, as opposed to the QS kernel with complexity $O(Nn^3b + N^2b)$. Thus one would assume that the classical kernel would run faster than its quantum variant. We conjecture that the slower runtimes observed in Sec. 4.2.2 are due to overheads in the reading in and out of data from the label lookup table. Since the QS kernel is made up solely of matrix operations, MATLAB's proficiency in matrix computations leads to a fast calculation of the QS kernel.

When comparing entropy labels used in QSWL, one had to be careful of declaring the entropies of two rows different when the rows had the same elements (up to permutations). This difference would arise due to floating point error. Thus when comparing entropies we did so to within a tolerance of 10^{-12}. We can see why this value was chosen by considering the change in Shannon entropy brought about by a change in the probabilities p_i it takes as its argument.

$$H(p_1, p_2, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log p_i,$$

$$\frac{\partial H}{\partial p_i} = -(\log p_i + 1),$$

$$\Rightarrow \Delta H \approx \sum_{i=1}^{n} [-(\log p_i + 1)\Delta p_i]. \tag{4.3.4}$$

Proceeding with a heuristic argument, we take typical numerical precision as $\sim 10^{-15}$. Then we can use $\Delta p_i = 10^{-15}$ in Eq. 4.3.4. It is when p_i is at (or below) this value that it is effectively taken as zero in $H(\,\cdot\,)$, giving $p_i \log p_i \approx 0$, so any p_i below 10^{-15} are ignored in the calculation of

Shannon entropy. Now since $\log\left(10^{-15}\right) \approx -35$, we have from Eq. 4.3.4 that $\Delta H \approx n \cdot 35 \cdot 10^{-15}$. Thus a tolerance of 10^{-12} safely protects against numerical error in the regime where we have graphs with $n < 10$ vertices, but still allows for the correct differentiation between entropy labels due to its small magnitude.

Chapter 5

Conclusion

5.1 Future Directions

There are a number of directions in which the research presented thus far can be extended. We now consider some of the options.

The first extension would be to use the average mixing matrix to define a kernel in a more intelligent way. Labelling the vertices of a graph G by the entropies of the corresponding rows in $\widehat{M}_G$ seems to be a good choice thus far, as the multiset of vertex labels is a graph invariant and when used as an initial labelling in the WL algorithm, is effective at determining isomorphism. At the moment, with the QS kernel, in merely binning the vertex labels by their entropies a lot of the relationships between graph substructures is lost. If one can construct a kernel which is cognisant of the relation between structures *within* each graph then perhaps the classification accuracy of the kernel could be increased. Thus it is the (implicit or explicit) mapping into feature space which is preventing an increase in classification accuracy. However, from Prop. 3.5.1, doing this in a complete way is computationally non-trivial. Nevertheless, we can consider the following.

5.1.1 Graphlet Spectrum

A classical scheme taking substructure into account, which one could take inspiration from is presented in [51]. The authors define a *graphlet spectrum*, derived from group representation theory, which captures information about the number and position of subgraphs within a graph. The rough idea is to consider a small group of subgraphs, or graphlets, of structural interest $\{g_i\}$. For instance, when looking at chemical data the graphlets could be functional groups. Then, construct an indicator function, $f_G(v_1, \ldots, v_k)$ for each g_i which is $+1$ if g_i is contained within the vertices $\{v_1, \ldots, v_k\}$ and zero otherwise. This indicator function must also be permutation invariant. Then the Fourier transform of the indicator f_G over the symmetric group $\mathbb{S}_n$ is taken. The symmetric group $\mathbb{S}_n$ is the group of all permutations $(1, 2, \ldots, n) \to (1, 2, \ldots, n)$ between n objects, with $n!$ elements. It can be easily seen that $\mathbb{S}_n$ satisfies the group axioms of closure, associativity, identity and having an inverse for each element. The Fourier transform over $\mathbb{S}_n$ is

given by

$$\hat{f}(\lambda) = \sum_{\sigma \in \mathbb{S}_n} f(\sigma)\rho_\lambda(\sigma), \tag{5.1.1}$$

where λ is an ordered tuple of natural numbers indexing a particular irreducible representation, $\rho_\lambda(\,\cdot\,)$, of an element of $\mathbb{S}_n$, σ. Taking the Fourier transform of the indicator function, $\hat{f}_G(\lambda)$ thus gives a matrix. Taking the Hermitian conjugate of $\hat{f}_G(\lambda)$ and multiplying by $\hat{f}_G(\lambda)$, over all graphlets gives a collection of matrices $\hat{q}_{i,j} = (\hat{f}_{G,g_i}(\lambda))^\dagger \cdot \hat{f}_{G,g_j}(\lambda)$. These matrices are then used to construct features which can then be used as a kernel.

The average mixing matrix, being constructed out of the idempotents of the adjacency matrix, gives information about a given graph in Fourier space. Perhaps using a construction such as in [51] in conjunction with the average mixing matrix and Shannon entropy could prove to be useful as a graph kernel.

5.1.2 Graph kernels using a Quantum Annealer

Quantum annealing is the finite-temperature analogue of adiabatic quantum computing. The idea is to map the problem of interest into a Hamiltonian H_p such that its ground state encodes the solution of the problem. One then initialises the quantum system into the ground state of a know Hamiltonian, H_d. The adiabatic theorem [52] then ensures that by interpolating between the two Hamiltonian slowly enough, the system remains in the ground state and thus at the end of its evolution encodes the solution of the problem at hand. The Hamiltonian can thus be expressed as

$$H = (1-s)H_d + sH_p, \tag{5.1.2}$$

where $s \in [0,1]$ is the anneal parameter.

D-Wave have constructed an experimental quantum annealer [53] which can encode Hamiltonians of the form

$$H = (1-s)\sum_{v \in V} h_v \sigma_v^x + s \sum_{\{u,v\} \in E} J_{uv}\sigma_u^z \sigma_v^z, \tag{5.1.3}$$

where σ_v^x, σ_v^z are Pauli matrices acting on the *qubit* v and (V, E) are the vertex and edge sets of some graph G. In [54], the authors demonstrate that for many strongly regular graphs, a class of graphs for which isomorphism testing is hard, taking statistics of certain quantities and comparing them can determine if these graphs are isomorphic. It might prove interesting to see if using a similar approach, one could construct a positive semi-definite kernel and see how it fares on classification tasks. While this approach was not hugely successful for determining isomorphism between graphs, perhaps the spectral characteristics of Hamiltonians of graphs running on D-Wave could provide a good similarity measure.

5.1.3 Fully Quantum Graph kernels

In this report we have only considered quantum-inspired graph kernels. One can consider graph kernels defined relative to the properties of a quantum walk. In [55], the authors define a graph kernel similar to that discussed in Sec. 3.4.3, which uses the quantum Jensen-Shannon divergence as a distance measure between density matrices encoding the evolution of a quantum walk. Here,

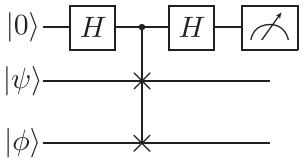

Figure 5.1: Circuit for the swap test, which measure the overlap between two states $|\psi\rangle$ and $|\phi\rangle$. The probability of measuring a zero on the ancilla qubit is $P(|0\rangle) = \frac{1}{2} + \frac{1}{2}|\langle\phi|\psi\rangle|^2$. Over many iterations, the overlap can be determined to arbitrary accuracy with only single copies of $|\psi\rangle$ and $|\phi\rangle$.

instead of just taking the limiting behaviour, they construct a new graph based on the two graphs $G_1(V_1, E_1)$, $G_2(V_2, E_2)$. This construction $\mathcal{G}(\mathcal{V}, \mathcal{E})$ has $\mathcal{V} = V_1 \cup V_2$ and $\mathcal{E} = E_1 \cup E_2 \cup E_{12}$, where $\{u, v\} \in E_{12}$ if and only if $u \in V_1$ and $v \in V_2$. They then define two quantum walks, with starting states

$$\left|\psi_0^-\right\rangle = \frac{\sum_{u\in V_1} d(u)\,|u\rangle - \sum_{v\in V_2} d(v)\,|v\rangle}{C} \quad \text{and} \quad \left|\psi_0^+\right\rangle = \frac{\sum_{u\in \mathcal{V}} d(u)\,|u\rangle}{C}, \tag{5.1.4}$$

where C is a normalising constant. At various times t the respective states are compared using the quantum Jensen-Shannon divergence (Eq. 3.4.18) with this quantity acting as the kernel between two graphs for a given pair of graphs G_1, G_2.

It was found that due to the unitary, non-ergodic character of quantum walks, the classification accuracy varied widely over a range of timesteps, and for certain times t, did very well as compared with classical graph kernels.

This work required simulating the quantum walks classically, then calculating the quantum Jensen-Shannon divergence classically also. A similar scheme could be conceived which is 'fully quantum'; that is, the quantum walks are simulated on a quantum computer and some efficient comparison between walks is made which takes advantage of the quantum hardware. There are many initial states and graph constructions one could use. In terms of a 'fully-quantum' distance measure, one could consider the swap test [56]. The quantum circuit for the swap test is shown in Fig. 5.1.

5.2 Closing Remarks

In this report we have seen how the theory of quantum walks may be applied to practical problems of graph classification, namely graph isomorphism and graph similarity. Having introduced the field in Chap. 1, we presented a detailed survey of quantum walks in Chap. 2 and machine learning on graphs in Chap. 3.

In Chap. 4, we presented algorithms using the average mixing matrix, defined with respect to a continuous quantum walk, to test isomorphism between graphs and to compute a similarity measure between graphs. In terms of the isomorphism test, QSWL, as compared with its classical counterpart, WL, proved to be better at distinguishing graphs. With this improvement in distinguishing power comes a penalty in the runtime complexity. The similarity measure or more precisely, the QS graph kernel which was defined, is competitive in classification accuracy

and superior in runtime on the MUTAG bioinformatics dataset. However, testing on more datasets is required to establish its credibility as a new method for computing graph similarity. We believe that while effective on MUTAG, this kernel needs to take more account of graph substructure to be decisively better than existing methods on arbitrary sets of data.

Overall, we have observed that quantum-inspired algorithms which are computationally tractable can hold their own in the areas of graph similarity and graph isomorphism, problems in computer science which have traditionally been hard to solve. With some improvements, we hope that these methods, extended into the fully quantum realm, can one day show unambiguous superiority over the best possible classical algorithms.

5.3 Update - Further Work

A number of changes have been made to the QS kernel, which we discuss here.

1. **Taking the low T Limit:** Instead of using the definition of the average mixing matrix with $T = \infty$, using the finite T definition (Eq. 2.2.9) yielded accuracies a percentage point or two higher for $T \sim 1$.

2. **Take Gaussian/RBF kernel between features:** With the QS kernel (Def. 4.2.2), feature vectors are constructed then the linear kernel is taken between them to build the kernel matrix. We can try different kernels $\kappa : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$, such as the Gaussian kernel

$$\kappa(\mathbf{x}, \mathbf{y}) = \exp(-\gamma|\mathbf{x} - \mathbf{y}|), \quad \text{for} \quad \gamma \in \mathbb{R}. \tag{5.3.1}$$

   Here, for $\gamma \sim 0.01$, improvement in accuracy of a couple of percentage points was observed.

3. **Optimal Assignment kernel:** Here we take inspiration from the optimal assignment kernel presented in [57]. Consider two graphs, $G = (V, E)$ and $G' = (V', E')$. We construct a complete bipartite graph $B(V, V', E, W)$ between the two vertex sets V, V' with the edges having weight determined by $W : V \times V' \mapsto \mathbb{R}$. The kernel is then constructed by finding the optimal matching on B and totalling the weight, where by optimal matching we mean a set of vertex disjoint edges with the total weight of the edges fulfilling some optimality condition.

   A number of schemes were attempted, such as finding the matching where the difference between the Shannon entropies of the rows of the average mixing matrix of G and G' were minimised. Also attempted was using the (sorted for permutation invariance) rows of the average mixing matrix, then minimising Kullback-Liebler divergence, as well as the Euclidean norm. None of these schemes improved the kernel's accuracy, perhaps due to the fact the optimal assignment kernel is not positive semi-definite in general [58].

4. **Use Vertex Labels:** The QS kernel was originally defined on unattributed graphs. We can extend the definition to include node labels by binning not only by average mixing matrix row entropy, but also by vertex label. This returned a modest accuracy improvement also.

Combining improvements 1), 2) and 4) leads to accuracies (attributed graphs) of 84.67±0.61% on MUTAG and 54.9±0.27% for ENZYMES, compared with MUTAG: 83.3±0.99% and ENZYMES: $53.7 \pm 0.39\%$ for the WL subtree kernel. These aren't huge but are slight improvements.

References

[1] M. Hilbert, P. López, C. Perez *et al.* The world's technological capacity to store, communicate, and compute information. *Science*, **332**, 60 (2011).

[2] D. Silver, A. Huang, C. J. Maddison *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature*, **529**, 484 (2016).

[3] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Sci. Stat. Comput.*, **26**, 1484 (1997).

[4] L. K. Grover. Quantum Mechanics Helps in Searching for a Needle in a Haystack. *Phys. Rev. Lett.*, **79**, 325 (1997).

[5] L. Babai. Graph Isomorphism in Quasipolynomial Time (2015). `arXiv:1512.03547`.

[6] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. third Annu. ACM Symp. Theory Comput. - STOC '71*, pages 151–158. ACM Press, New York, New York, USA (1971).

[7] L. De Nardo, F. Ranzato, and F. Tapparo. The Subgraph Similarity Problem. *IEEE Trans. Knowl. Data Eng.*, **21**, 748 (2009).

[8] W. Feller. *An introduction to probability theory and its applications*. Wiley, New York, NY, USA (1968).

[9] E. Farhi and S. Gutmann. Quantum Computation and Decision Trees. *Phys. Rev. A*, **58**, 28 (1998).

[10] S. Bose. Quantum Communication through an Unmodulated Spin Chain. *Phys. Rev. Lett.*, **91**, 207901 (2003).

[11] A. M. Childs, E. Farhi, and S. Gutmann. An example of the difference between quantum and classical random walks. *Quantum Inf. Process.*, **1**, 35 (2001).

[12] E. Farhi and S. Gutmann. The functional integral constructed directly from the Hamiltonian. *Ann. Phys. (N. Y).*, **213**, 182 (1992).

[13] D. Aharonov, A. Ambainis, J. Kempe *et al.* Quantum walks on graphs. In *Proc. 33rd Annu. ACM Symp. Theory Comput. - STOC '01*, pages 50–59. ACM Press, New York, NY, USA (2001).

[14] C. Godsil. Average mixing of continuous quantum walks (2011). `arXiv:1103.2578`.

[15] J. Kempe. Quantum random walks: An introductory overview. *Contemp. Phys.*, **44**, 307 (2003).

[16] A. Nayak and A. Vishwanath. Quantum Walk on the Line (2000). `arXiv:quant-ph/0010117`.

[17] A. Ambainis, E. Bach, A. Nayak *et al.* One-dimensional quantum walks. In *Proc. 33rd Annu. ACM Symp. Theory Comput. - STOC '01*, pages 37–49. ACM Press, New York, New York, USA (2001).

[18] F. M. Andrade and M. G. E. Da Luz. Equivalence between discrete quantum walk models in arbitrary topologies. *Phys. Rev. A*, **80**, 052301 (2009).

[19] F. M. Andrade, A. G. M. Schmidt, E. Vicentini *et al.* Green's function approach for quantum graphs: an overview (2016). `arXiv:1601.01018`.

[20] F. M. Andrade and M. G. E. Da Luz. Green-function approach for scattering quantum walks. *Phys. Rev. A*, **84**, 042343 (2011).

[21] A. Ambainis. Quantum walk algorithm for element distinctness. *45th Annu. IEEE Symp. Found. Comput. Sci.*, **354**, 22 (2014).

[22] N. Shenvi, J. Kempe, and K. B. Whaley. Quantum random-walk search algorithm. *Phys. Rev. A*, **67**, 052307 (2003).

[23] N. B. Lovett, S. Cooper, M. Everitt *et al.* Universal quantum computation using the discrete-time quantum walk. *Phys. Rev. A*, **81**, 042330 (2010).

[24] A. M. Childs. Universal computation by quantum walk. *Phys. Rev. Lett.*, **102**, 180501 (2009).

[25] A. M. Childs, R. Cleve, E. Deotto *et al.* Exponential algorithmic speedup by a quantum walk. In *Proc. 35th ACM Symp. Theory Comput. - STOC '03*, pages 59–68. ACM Press, New York, New York, USA (2003).

[26] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK (2004).

[27] A. E. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, **12**, 55 (1970).

[28] C. Cortes and V. Vapnik. Support-Vector Networks. *Mach. Learn.*, **20**, 273 (1995).

[29] W. Imrich and S. Klavžar. *Product Graphs, Structure and Recognition*. Wiley (2000).

[30] S. Vishwanathan, N. N. Schraudolph, R. Kondor *et al.* Graph Kernels. *J. Mach. Learn. Res.*, **11**, 1201 (2010).

[31] T. Gärtner, P. Flach, and S. Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. In *Learn. Theory Kernel Mach.*, pages 129–143. Springer Berlin Heidelberg (2003).

[32] C. Cortes, P. Haffner, H. A. Com *et al.* Rational Kernels: Theory and Algorithms. *J. Mach. Learn. Res.*, **5**, 1035 (2004).

[33] D. Haussler. Convolution Kernels on Discrete Structures. UCSC-CRL-99-10. *Technical report*, UC Santa Cruz (1999).

[34] N. Shervashidze, P. Schweitzer, E. Jan van Leeuwen *et al.* Weisfeiler-Lehman Graph Kernels. *J. Mach. Learn. Res.*, **12**, 2539 (2011).

[35] B. Y. Weisfeiler and A. A. Lehman. Reduction of a graph to a canonical form and an algebra which appears in the process. *Nauchno-Technicheskaya Informatsiya, Ser. 2*, **9**, 12 (1968).

[36] L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *20th Annu. Symp. Found. Comput. Sci.*, pages 39–46. IEEE (1979).

[37] N. Shervashidze and K. M. Borgwardt. Fast subtree kernels on graphs. In *Adv. Neural Inf. Process. Syst. 22 (NIPS 2009)*, pages 1660–1668 (2009).

[38] L. Bai, L. Rossi, A. Torsello *et al.* A quantum Jensen-Shannon graph kernel for unattributed graphs. *Pattern Recognit.*, **48**, 344 (2015).

[39] L. Bai, L. Rossi, P. Ren *et al.* A quantum Jensen-Shannon graph kernel using discrete-time quantum walks. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, **9069**, 252 (2015).

[40] L. Bai, E. R. Hancock, A. Torsello *et al.* A Quantum Jensen-Shannon Graph Kernel Using the Continuous-Time Quantum Walk. In *Proc. Graph-Based Represent. Pattern Recognit.*, pages 121–131 (2013).

[41] A. F. T. Martins, N. A. Smith, E. P. Xing *et al.* Nonextensive Information Theoretic Kernels on Measures. *J. Mach. Learn. Res.*, **10**, 935 (2009).

[42] P. W. Lamberti, A. P. Majtey, A. Borras *et al.* Metric character of the quantum Jensen-Shannon divergence. *Phys. Rev. A*, **77**, 052311 (2008).

[43] S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, **10**, 695 (1988).

[44] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge University Press (2013).

[45] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, **60**, 94 (2014).

[46] T. Mikkonen. The Ring of Graph Invariants - Graphic Values (2007). `arXiv:0712.0146`.

[47] D. Emms, E. R. Hancock, S. Severini *et al.* A matrix representation of graphs and its spectrum as a graph invariant. *Electron. J. Comb.*, **13**, R34 (2006).

[48] C.-C. Chang and C.-J. Lin. LIBSVM. *ACM Trans. Intell. Syst. Technol.*, **2**, 1 (2011).

[49] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath *et al.* Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *J. Med. Chem.*, **34**, 786 (1991).

[50] N. Shervashidze. Graphkernels (2012). `http://mlcb.is.tuebingen.mpg.de/Mitarbeiter/Nino/Graphkernels/`.

[51] R. Kondor, N. Shervashidze, and K. M. Borgwardt. The Graphlet Spectrum. In *Proc. 26th Int. Conf. Mach. Learn.*, pages 529–536 (2009).

[52] M. Born and V. Fock. Beweis des Adiabatensatzes. *Zeitschrift für Phys.*, **51**, 165 (1928).

[53] M. W. Johnson, M. H. S. Amin, S. Gildert *et al.* Quantum annealing with manufactured spins. *Nature*, **473**, 194 (2011).

[54] I. Hen and A. P. Young. Solving the graph-isomorphism problem with a quantum annealer. *Phys. Rev. A*, **86**, 042310 (2012).

[55] L. Rossi, A. Torsello, and E. R. Hancock. Measuring graph similarity through continuous-time quantum walks and the quantum Jensen-Shannon divergence. *Phys. Rev. E*, **91**, 022815 (2015).

[56] H. Buhrman, R. Cleve, J. Watrous *et al.* Quantum Fingerprinting. *Phys. Rev. Lett.*, **87**, 167902 (2001).

[57] H. Fröhlich, J. K. Wegner, F. Sieker *et al.* Optimal assignment kernels for attributed molecular graphs. In *Proc. 22nd Int. Conf. Mach. Learn. - ICML '05*, pages 225–232. ACM Press, New York, New York, USA (2005).

[58] J.-P. Vert. The optimal assignment kernel is not positive definite (2008). `arXiv:0801.4061`.