Universiteit Leiden
The Netherlands

Opleiding Informatica

Reducing the required number of qubits
for a quantum algorithm for topological data analysis

Daan Planken

Supervisors:
Casper Gyurik & Vedran Dunjko

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

10/02/2021

Abstract

Quantum computers have been researched for years, with quantum algorithms promising large speedups over classical algorithms. However, since quantum computers with many gates or many qubits are hard to build, there is a need to look at methods to reduce the number of required gates or qubits by optimizing existing algorithms.
In this thesis we look at an algorithm for topological data analysis that estimates the so-called Betti numbers of a dataset. The k-th Betti number of a dataset is equal to the number of k-dimensional "holes" the dataset has. These Betti numbers can provide insight about the dataset, and have a wide variety of uses. In this bachelor thesis we show a way to reduce the number of ancilla qubits required for quantum phase estimation, an integral part of the algorithm. We also determine the time complexity of the changed algorithm, and compare this to the time complexity of the original algorithm.

Contents

1 Introduction

Quantum computers have for a long time been hailed as the next step in computing, promising exponential speed ups compared to classical (non-quantum) computers, with many algorithms being developed for them.
However, at the moment, the biggest quantum computer has only 53 qubits [AAB+19]. On top of that, this quantum computer, called the Sycamore processor, cannot be used as a general purpose quantum comptuter, since its gates are very limited, because they are noisy, and can only implement a limit set of quantum gates. However, fortunately, quantum computers are projected to have an increasing number of qubits in the coming years, while also increase their gate reliability and usability, and decreasing the noise, and it is expected that we enter the so-called NISQ era [Pre18]. NISQ stands for Noisy Intermediate Scale Quantum. Here noisy refers to the fact that qubits, the "bits" of a quantum computer, and gates are very error prone, and the intermediate scale refers to the fact that these quantum computers will not have many qubits, only between fifty and around a thousand. Despite these limitations, recently the Sycamore processor has achieved what is known as quantum supremacy, meaning that it can perform calculations that take (practically) infinitely on a classical super computer [AAB+19]. However, the task this quantum computer performed was purely synthetic, and did not calculate anything useful. We are therefore looking for a quantum algorithm that can be executed on a NISQ quantum computer, and that actually computes something with a real world application.
Since the NISQ quantum computers will be very limited in both the number of qubits and the number of gates, it is important that in order to find one such algorithm, we look at ways to both reduce the number of qubits and the number of gates required for existing algorithms. This way the already developed algorithms can be executed sooner, without having to wait for quantum computers to be as advanced.
One candidate algorithm is an algorithm for topological data analysis. This algorithm, called the LGZ algorithm, is used to esimate the so-called Betti numbers, where the k-th Betti number is essentially the number of k-dimensional "holes" a dataset has. In this bachelor thesis, we will look at the LGZ algorithm, and reduce the number of qubits required for a part of this algorithm called quantum phase estimation. In addition, we look at the complexity of the resulting algorithm, and compare it to the existing algorithm.

2 Quantum computers

In order to talk about what is going on inside a quantum computer, we need a mathematical model to describe its states and transitions, similar to how boolean algebra describes the states and transitions of classical computers. The language of this model is complex linear algebra.

2.1 State space

On a classical computer, the state space is given by the values of the bits in the computer. These values can be 0 or 1. Quantum computers do not use normal bits, instead, they use a special kind of bit called a qubit. Where as a classical bit can only take the values 0 and 1, a qubit can be in a

lot of different states: the state of a qubit can be written as

$$\alpha_0|0\rangle + \alpha_1|1\rangle$$

where $\alpha_0, \alpha_1 \in \mathbb{C}$ and $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Here $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ and $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$ are vectors in $\mathbb{C}^2$, forming an orthonormal basis, written as so-called ket vectors using what is known as the bra-ket, or Dirac, notation. This notation, stemming from physics, is used in quantum computing to denote vectors. So we could write $|\phi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$, with $|\phi\rangle \in \mathbb{C}^2$.
In order to describe multi qubit systems, we make use of the tensor product. We write $|00\rangle$ or $|0\rangle|0\rangle$ for $|0\rangle \otimes |0\rangle$. Then we can write state vector in an n qubit quantum computer as $\sum_{x\in\{0,1\}^n} \alpha_x|x\rangle$, where x sums over all strings of zeroes and ones of length n, satisfying $\sum_{x\in\{0,1\}^n} |\alpha_x|^2 = 1$. Here the $|x\rangle$ are the basis vectors, and in traditional linear algebra notation we have $|00...00\rangle = [1, 0, ..., 0]^\top$, $|00...01\rangle = [0, 1, 0, ..., 0]^\top$, etc.
Note that for n qubits, this results in 2^n basis states. This is the reason that classically simulating a quantum computer is hard. The α_x are called the quantum amplitudes.
An example of this for $n = 2$ is given by

$$|\phi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$

with $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$. So $|\phi\rangle$ is a state vector for 2 qubits.

2.2 Unitary evolution

Just like normal bits, qubits are useless without a way to change their values. If we only consider one-bit gates, classically there is only one non-trivial option: we invert the value of the bit, a **NOT** gate. But since qubits can have more than one value, the one *qu*bit gates are also a lot more interesting.
Since every qubit is of the form $\alpha_0|0\rangle + \alpha_1|1\rangle$, it's tempting to think that we can simply describe what happens with $|0\rangle$ and $|1\rangle$, the basis vectors, in order to describe what happens when we apply the gate to any qubit. While this is true, we do need to be careful, since the resulting qubit $\alpha_0'|0\rangle + \alpha_1'|1\rangle$ also needs to satisfy $|\alpha_0'|^2 + |\alpha_1'|^2 = 1$. It turns out that all linear operations that satisfy this condition can be implemented as a single qubit gate, and can be described by the complex matrices

$$U = \begin{bmatrix}u_{00} & u_{01}\\ u_{10} & u_{11}\end{bmatrix}$$

that satisfy $UU^\dagger = \mathbb{I}$, the so-called unitary matrices.
Some unitary matrices are especially important, and are among the standard gates used in quantum computing, such as the Hadamard gate

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}.$$

With this gate we get that $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, which is also written as $|+\rangle$. We also have that $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, which is, perhaps not surprisingly, written as $|-\rangle$. Together, $|+\rangle$ and $|-\rangle$ also

form an orthonormal basis of $\mathbb{C}^2$, sometimes called the Hadamard basis.
Other standard gates include the Pauli gates, given by

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

For each of these we find that when applied twice, we obtain the identity, $\mathbb{I} = H^2 = X^2 = Y^2 = Z^2$.
Another important gate is the parameterized Z-rotation:

$$R_z(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}.$$

This gate acts as the identity on $|0\rangle$, but sends $|1\rangle$ to $e^{i\theta}|1\rangle$.

2.2.1 Multi-qubit

For multiple qubits, the requirement that the matrix is unitary is still sufficient to allow the gate to be implementable. A very important gate for two qubits is the CNOT gate, given by

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

When applied to $|00\rangle$ or $|01\rangle$, we see that the matrix acts as the identity. However, when applied to $|10\rangle$ or $|11\rangle$, these vectors get swapped, meaning $|10\rangle$ gets sent to $|11\rangle$ and vice versa. This means that if the first bit of the $|x_0x_1\rangle$ is 0, then the vectors are unchanged. But if the first bit is 1, the second bit gets flipped. Therefore, the gate acts as a controlled inverter, only inverting the second bit if the first bit is 0.
We can generalise the idea of the CNOT to any unitary U acting on n qubits via the matrix

$$C - U = \left[\begin{array}{c|c} \mathbb{I}_{2^n} & 0 \\ \hline 0 & U \end{array}\right]$$

This means that if $|\phi\rangle$ is a state vector of an n qubit quantum circuit, then $C - U|0\rangle|\phi\rangle = |0\rangle|\phi\rangle$, but $C - U|1\rangle|\phi\rangle = |1\rangle U|\phi\rangle$.

2.3 Measurements

Being able to manipulate the qubit state is only useful if we are able to determine the state of the qubits after applying the operations. Unfortunately, this is not possible. Or rather, not directly. What we can do instead is perform a measurement on the qubits.

Before we can talk about measurements, we need to talk about bra vectors. Let $|\phi\rangle \in \mathbb{C}^2$ be a ket vector. Then we define $\langle\phi| \in \mathbb{C}^{*2}$ as the dual vector of $|\phi\rangle$ (with respect to the standard basis). The other way around, for $\langle\psi| \in \mathbb{C}^{*2}$ we define $|\psi\rangle \in \mathbb{C}^2$ as the corresponding dual vector (again with respect to the standard basis). If $|\phi\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}$, then we associate $\langle\phi|$ with $(\bar{\alpha}, \bar{\beta})$ where the bar indicates the complex conjugate.
Now we can define $\langle\psi||\phi\rangle = \langle\psi|\phi\rangle$ as the inner product $(|\psi\rangle, |\phi\rangle)$, which is a scalar in $\mathbb{C}$.
Let $|\phi\rangle \in \mathbb{C}^2$ be a state vector. The probabilities of measuring in the standard basis, and the changes to $|\phi\rangle$ are given by Born's rule:

1. For $i \in \{0, 1\}$, the probability that i is observed is equal to
$$p_i := \||i\rangle\langle i|\phi\rangle\|^2 = \langle\phi|i\rangle\langle i|\phi\rangle$$
2. For $i \in \{0, 1\}$, for $p_i \neq 0$ the post-measurement state is equal to
$$|\phi^i\rangle := \frac{1}{\sqrt{p_i}}|i\rangle\langle i|\phi\rangle$$

Here $|\phi^i\rangle$ is the state that $|\phi\rangle$ collapses to when we observe outcome i.
As an example, if we would measure $|\phi\rangle = |1\rangle$, we would measure 0 with probability $p_0 = \langle 1|0\rangle\langle 0|1\rangle$. Since $|0\rangle$ and $|1\rangle$ are orthogonal, this is 0. Since the probabilities have to add up to 1, we already know that $p_1 = 1$, but for the sake of completeness: $p_1 = \langle 1|1\rangle\langle 1|1\rangle = 1$. The post-measurement state equals $|1\rangle\langle 1|1\rangle = |1\rangle$.
A more interesting example would be $|\phi\rangle = |+\rangle$. Here we have $p_0 = \langle +|0\rangle\langle 0|+\rangle = \frac{1}{\sqrt{2}}(\langle 0| + \langle 1|)|0\rangle\langle 0|+\rangle = \frac{1}{\sqrt{2}}\langle 0|+\rangle = \frac{1}{2}(\langle 0|0\rangle + \langle 1|0\rangle) = \frac{1}{2}$. Then again, we have $p_1 = 1 - \frac{1}{2} = \frac{1}{2}$. This means that $|+\rangle$ is exactly "inbetween" $|0\rangle$ and $|1\rangle$.
We can also perform a measurement on a qubit when we have a system of multiple qubits. Let $|\phi\rangle = \sum_{x\in\{0,1\}^{n-1}} \alpha_{0x}|0\rangle|x\rangle + \alpha_{1x}|1\rangle|x\rangle$ be a state vector. The probability of measuring i, with $i \in \{0, 1\}$, in the first qubit is given by
$$\langle\phi|(|i\rangle\langle i| \otimes \mathbb{I}_{2^{n-1}})|\phi\rangle = \sum_{x\in\{0,1\}^{n-1}} |\alpha_{ix}|^2\langle i|i\rangle \otimes \langle x|x\rangle = \sum_{x\in\{0,1\}^{n-1}} |a_{ix}|^2.$$

2.4 Density matrix formalism

So far we have only seen state vectors. These can describe the state of a quantum computer, but sometimes we are not exactly sure what the state is, but we only know that the system is in a certain state with a certain probability, so it describes a classical probability distribution of quantum states. Note that this is different from a superposition, which is a single quantum state, with different possible measurement outcomes. To describe these states, we make use of a different notation called the density matrix formalism.
For a state $|\phi\rangle$, the density matrix notation is given by $|\phi\rangle\langle\phi|$.
If we have states $|\phi_1\rangle, ..., |\phi_n\rangle$, and we know that the circuit is in state $|\phi_i\rangle$ with probability ϵ_i, such that $\sum_{i=1}^n \epsilon_i = 1$, we can write the so-called mixed state
$$\rho = \sum_{i=1}^n \epsilon_i|\phi_i\rangle\langle\phi_i|$$

Since applying a unitary U to $|\phi\rangle$ gives the state $U|\phi\rangle$, the corresponding density matrix is given by $U|\phi\rangle\langle\phi|U^\dagger$. Applying a unitary U to a state ρ then gives $U\rho U^\dagger = \sum_{i=1}^{n} U|\phi_i\rangle\langle\phi_i|U^\dagger$.

2.4.1 Measuring a density matrix

We measure a density state $\rho = \sum_{j,k\in\{0,1\}} a_{jk}|j\rangle\langle k|$, the probability of measuring i, with $i \in \{0,1\}$ with probability

$$p_i := \mathrm{tr}\left(|i\rangle\langle i|\rho\right) = \langle i| \left(\sum_{j,k\in\{0,1\}} a_{jk}|j\rangle\langle k| \right) |i\rangle = a_{ii}.$$

2.5 Quantum circuit

Now that we know what qubits are, how we change their state, and how we measure them, we can define a quantum circuit. A quantum circuit consists first of all of a register of qubits. The starting state of these qubits can be all $|0\rangle$, or depend on the input of the algorithm. The second part of a quantum circuit consists of a set of gates to be applied to those qubits, represented by unitary matrices, as described earlier. The last part is a measurement on one or more of the qubits, to obtain a classical outcome of the calculations.

2.6 Complexity of a quantum algorithm

A quantum algorithm is an algorithm, along with a uniform family of circuits, such that for every input size there is a corresponding quantum circuit, such that this circuit, along with optional pre- and post-processing, when executed allows us to solve a problem. As with classical algorithms, we are concerned with the amount of resources that are required to execute the algorithm, as a function of the input.
When looking at the complexity of a quantum algorithm, we are concerned with three things:

1. The time complexity
2. The space complexity
3. The depth complexity

The classical and quantum time complexity is given by the total number of gates that have to be executed to run the algorithm, together with the time complexity of the classical part of the algorithm, which can consist of a computation to determine what quantum circuit should be run, and pre- and post-processing. This is especially important because it can be difficult to add many gates to a physical circuit, with the restrictions of the NISQ computers. On top of that, the gates that can currently be built are not perfect and have a high error rate.
The quantum space complexity is given by the number of qubits the algorithm needs. This is important because making a quantum computer with many qubits is hard.
The depth is given by the largest number of gates that have to be executed in order. Quantum gates that act on disjunct sets of qubits can executed in parallel, which speeds up the computation. However, since the time complexity is strongly correlated with the depth, we will look at the time

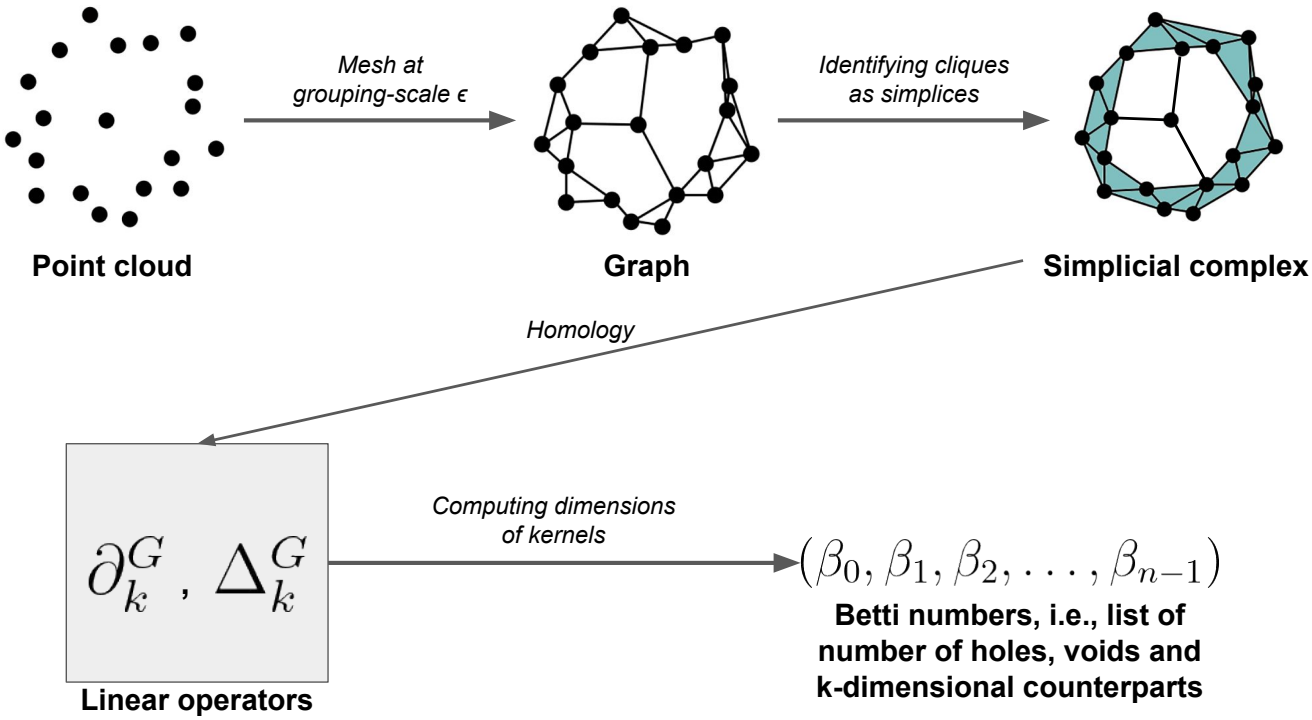

Figure 1: Topological data analysis, from [GCD20]

complexity for the rest of the thesis.
Since building quantum computers with a large number of qubits is so difficult, it is imperative to look at existing algorithms and try to reduce the number of qubits they need.

3 Topological data analysis

Often when we have measured a large data set, we want to know more about the general shape of the data. For two dimensional data, this can often be done by simply looking at the data. Sometimes, we can even learn about the features of higher dimensional data by studying planar projections of the data. However, often these higher dimensional features cannot be extracted from planar projections, because the shapes overlap and run into each other, making it difficult to understand what is going on.
Instead, one of the techniques we can use is called topological data analysis. The goal of topological data analysis is gaining insight into the higher dimension shape of data via something called persistent homology [Ghr08]. The computationally most expensive part of topological data analysis is calculating the number of k-dimensional "holes" in a data set, the so-called Betti numbers. The zeroth Betti number is the number of connected components in the dataset, the first Betti number is equal to number of "circular" holes, and the second Betti number is equal to the number of cavities, etc. Betti numbers can thus give insight into the higher dimensional shape of the data. In order to do this, as shown in figure 1, we will assume that we have a dataset as a point cloud, a finite subset of $\mathbb{R}^d$ for some d. We then transform this into a graph by connecting points that

are close together with respect to some metric. Next we define a clique complex, which allows us to calculate the Betti numbers by calculating the dimension of the kernel of something called the combinatorial Laplacian. In the next section, we will show how we can use a quantum computer to calculate this dimension.

3.1 The Vietoris-Rips complex

The data set is a point cloud $V = \{v_i\}_{i \in I} \subseteq \mathbb{R}^d$, with $I = \{1, ..., n\}$. We turn this point cloud into a graph by adding an edge between two points if their distance is smaller than a certain ϵ:

$$E_\epsilon = \{(i, j) | d(v_i, v_j) \leq \epsilon\}.$$

This gives the graph $G = (I, E_\epsilon)$.
We write $[j_0...j_k]$ for the subsets of I with vertices $j_0, ..., j_k$. In the quantum computer, these subsets are encoded as bitstrings of length n, with ones at the indices $j_0, ..., j_k$ and zeroes everywhere else. In order to calculate the Betti numbers, we need to define the simplices of the graph. We write $H_k \subseteq \{0, 1\}^n$ for the set of all bitstrings with exactly $k + 1$ ones. Then, we can define $\mathrm{Cl}_k(G) \subseteq H_k$ as the set of all $(k + 1)$-cliques of G, again encoded as bitstrings with exactly $k + 1$ ones.
Now, let $\mathrm{Cl}_k(G)$, the set of $(k + 1)$-cliques, be the k-simplices of the clique complex. Together, this data structure is called the Vietoris-Rips complex [Ghr08].

3.2 Boundary map

We can now define $\mathcal{H}_k$ as the complex vector space with basis H_k, and $\mathcal{H}_k^G \subseteq \mathcal{H}_k$ as the subspace spanned by $\mathrm{Cl}_k(G)$. Now we can define the boundary map:

Definition 1. *The boundary map $\partial_k : \mathcal{H}_k \to \mathcal{H}_{k-1}$ is given by the linear extension of*

$$\partial_k([j_0...j_k]) = \sum_{i=0}^{k} (-1)^i [j_0...\widehat{j_i}...j_k]$$

where $[j_0...\widehat{j_i}...j_k]$ equals $[j_0...j_k]$, but without the i-th vertex. We also define $\partial_k^G = \partial_k|_{\mathcal{H}_k^G}$.

If we write the subsets as bitstrings and use bra-ket notation, we get the following:

$$\partial_k |j\rangle = \sum_{i=0}^{k} (-1)^i |\widehat{j(i)}\rangle$$

where $\widehat{j(i)}$ is j, but with the entry of the i-th one set to zero.
Now that we have defined the boundary map, we can finally define the Betti numbers:

Definition 2. *The k-th Betti number, β_k is defined as the dimension of the quotient of the kernel of ∂_k^G and the image of ∂_{k+1}^G, or* $\dim(\ker \partial_k^G / \operatorname{Im} \partial_{k+1}^G)$.

Informally, the k-th Betti number is number of k-dimensional holes a dataset has. These definitions are based on [GK19] and [GCD20].

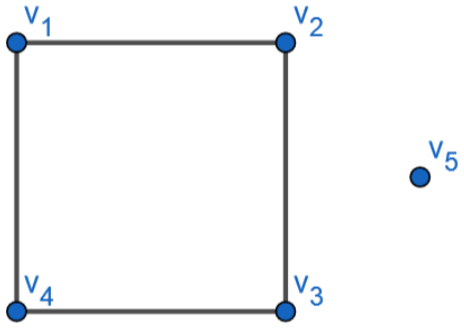

Figure 2: A representation of G

3.3 Combinatorial laplacian

Instead of calculating $\dim(\ker \partial_k^G / \operatorname{Im} \partial_{k+1}^G)$ directly, we can make use of the combinatorial Laplacian. This is the map given by $\Delta_k^G = \left(\partial_k^G\right)^\dagger \partial_k^G + \partial_{k+1}^G \left(\partial_{k+1}^G\right)^\dagger$. This combinatorial Laplacian satisfies $\ker \Delta_k^G \cong \ker \partial_k / \operatorname{Im} \partial_{k+1}$. Therefore we can calculate the k-th Betti number by calculating $\dim(\ker \Delta_k^G)$. This is easier than calculating the Betti number using the definition, because here we only need to calculate one kernel, instead of two.

3.4 Example

To give an example, we will look at the graph G, given by the representation in figure 2. By looking at the graph we can see what the Betti numbers should be as follows. There are two connected components, the square and v_5, so the zeroth Betti number should be two. There is only one circular hole, the hole in the square. Therefore the first Betti number should be one. Since there are no cliques of size 3 or higher, the higher Betti numbers have to be zero.
Now we will calculate the Betti numbers of G. We have $H_0^G = \{[v_1], [v_2], [v_3], [v_4], [v_5]\}$, equal to all the 1-cliques. Then for the the 2-cliques, we get $H_1^G = \{[v_1v_2], [v_2v_3], [v_3v_4], [v_4v_1]\}$. Since there are no 3-cliques, or cliques of a higher order, we know for $k \geq 2$ that $H_k^G = \emptyset$.
Using the definition, we can then calculate

$$\partial_1^G = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Since $H_2^G = \emptyset$, we also have that $\partial_2^G = 0$, the zero map. Since $\Delta_k^G = (\partial_k^G)^\dagger \partial_k^G + \partial_{k+1}^G (\partial_{k+1}^G)^\dagger$, we have that $\Delta_0^G = \partial_1^G (\partial_1^G)^\dagger$, and $\Delta_1^G = (\partial_1^G)^\dagger \partial_1^G$. For higher k, we obtain that $\partial_k = 0$ and $\partial_{k+1} = 0$, so $\Delta_k = 0$.

We start with $k = 0$, which gives us

$$\Delta_0^G = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & -1 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ -1 & 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The reduced row echelon form of this matrix equals

$$\begin{bmatrix} 1 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Since two rows are 0, we have that $\dim(\ker \Delta_0^G) = 2$, so that would mean that there are two connected components in the graph, which is correct.
Now, looking at $k = 1$, we get $\Delta_1^G = (\partial_1^G)^\dagger \partial_1^G$ because $\partial_2^G = 0$, which gives us

$$\Delta_1^G = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

Again getting the reduced row echelon form of the matrix, we obtain

$$\begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Since there is one 0 row and the rows are independent, we have that $\dim(\ker \Delta_1^G) = 1$, so we would have one circular hole. Again this is correct, since the square in the graph has a one dimensional hole.

4 The quantum algorithm for estimating Betti numbers

In order to estimate Betti numbers of a graph G on a quantum computer, we need to be able to determine $\beta_k := \ker(\Delta_k^G)$, for given k. We can do this by calculating the number of eigenvalues λ_j of Δ_k^G that are 0, since this is exactly the dimension of the kernel. However, since the matrix corresponding to Δ_k^G is very large, we will instead sample the eigenvalues to estimate the number of 0 eigenvalues. We do this by taking a random eigenvector, and by calculating the corresponding eigenvalue. We do this by first applying a quantum algorithm called Hamiltonian simulation to the Dirac operator B, which is the "square root" of Δ_k^G, to realize $e^{2\pi i B}$. After that we use an

algorithm called quantum phase estimation to determine the corresponding eigenvalue. We then use the fraction of eigenvalues that is 0, to estimate the total number of eigenvalues that are 0.
The algorithm for calculating the k-th Betti number of a graph $G = ([n], E_\epsilon)$ consists of three steps, that are each executed mulitple times:

1. Create the state $\rho = \frac{1}{N}\sum_{j=1}^{N} |j\rangle\langle j|$
2. Execute the phase estimation to e^iB, where we obtain e^iB using Hamiltonian simulation, with the eigenvector register in the state ρ
3. Perform a measurement of the eigenvalue register, and estimate a random eigenvalue λ_j

We repeat the above steps several M times, and then we calculate the fraction of the number of zero eigenvalues we found, divided by M, which is an approximation of $\frac{\beta_k}{N}$.
The algorithm is from [LGZ16], and an alternative explanation can be found in [GK19].

4.1 State preparation

Let $|\phi_j\rangle$ be the eigenvectors of Δ_k^G with eigenvalues λ_j, so $\Delta_k^G|\phi_j\rangle = \lambda_j|\phi_j\rangle$. In order to estimate the dimension of the kernel of Δ_k^G, we want to input a random eigenvector $|\phi_j\rangle$ of Δ_k^G, in other words we want to input $\frac{1}{N}\sum_{j=1}^{N} |\phi_j\rangle\langle\phi_j|$, where we take $N = |\mathrm{Cl}_k(G)|$. However, we currently do not know a way to construct a random eigenvector $|\phi_j\rangle$. Fortunately, since $\frac{1}{N}\sum_{j=1}^{N} |\phi_j\rangle\langle\phi_j| = \mathbb{I}_N = \frac{1}{N}\sum_{j\in \mathrm{Cl}_k(G)} |j\rangle\langle j|$, we can also input a random bitstring $|j\rangle$, as long as j encodes a $k+1$-clique.
We do this by classically taking a random bitstring $\phi \in \{0,1\}^n$ with Hamming weight $k+1$ (exactly $k+1$ ones), and then testing whether it is a $k+1$-clique. If it is not a $k+1$-clique, we try again with a different bitstring. If it is a $k+1$-clique, we continue to calculate the eigenvalue corresponding to $\Delta_k|\phi\rangle$.
Instead of using classical state preparation, we can make use of Grover's algorithm [GCD20]. The advantage of using this algorithm is that it is faster, however, since it is not classical, it requires more qubits. Since we are trying to reduce the required number of qubits, we will assume we use the classical method for the rest of this thesis.

4.1.1 Complexity of the state preparation

If we pick a random bitstring of length n with Hamming weight k, the probability that it is a k-clique equals $\frac{|\mathrm{CL}_k(G)|}{\binom{n}{k+1}} = \frac{N}{\binom{n}{k+1}}$. Therefore, on average we need to try $\frac{\binom{n}{k+1}}{N}$ bitstring, before we find one that is in $\mathrm{Cl}_k(G)$. Since checking if a bitstring is in $\mathrm{Cl}_k(G)$ takes $\mathcal{O}(k^2)$ operations, we can find a bitstring in $\mathrm{Cl}_k(G)$ in time

$$\mathcal{O}\left(k^2 \frac{\binom{n}{k+1}}{N}\right).$$

The complexity of using Grover's algorithm is

$$\mathcal{O}\left(k^2 \sqrt{\frac{\binom{n}{k+1}}{N}}\right)$$

so it provides a quadratic speedup over the classical state preparation.

4.2 The Dirac operator

Instead of looking at the combinatorial Laplacian, we look at its square root: the Dirac operator. The reason we do this, is that for Hamiltonian simulation, the next step of the algorithm, we need to access the elements of the matrix we are applying the Hamiltonian simulation to. It turns out that accessing the elements of B requires a lower number of gates than accessing the elements of B^2, resulting in a lower overall gate complexity of the algorithm.
The Dirac operator is given by the following Hermitian matrix:

$$B = \begin{bmatrix} 0 & \partial_1 & 0 & \dots & \dots & 0 \\ \partial_1^\dagger & 0 & \partial_2 & \dots & \dots & 0 \\ 0 & \partial_2^\dagger & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 & \partial_n \\ 0 & 0 & 0 & \dots & \partial_n^\dagger & 0 \end{bmatrix}$$

Since $\partial_k \partial_{k+1} = 0$, squaring B gives

$$B^2 = \begin{bmatrix} \partial_1\partial_1^\dagger & 0 & 0 & \cdots & 0 & 0 \\ 0 & \partial_1^\dagger\partial_1 + \partial_2\partial_2^\dagger & 0 & \cdots & 0 & 0 \\ 0 & 0 & \partial_2^\dagger\partial_2 + \partial_3\partial_3^\dagger & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \partial_{n-1}^\dagger\partial_{n-1} + \partial_n\partial_n^\dagger & 0 \\ 0 & 0 & 0 & \dots & 0 & \partial_n^\dagger\partial_n \end{bmatrix} = \begin{bmatrix} \Delta_0 & 0 & \cdots & 0 \\ 0 & \Delta_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & \Delta_{n-1} \end{bmatrix}$$

If $Bv = \lambda v$, then $B^2 v = \lambda^2 v$. So if we want to find eigenvalues of B^2 that are zero, we can also calculate the eigenvalues of B.
We see that because the matrix B consists of entirely of boundary maps, if we want to restrict it to our graph G, we can do this by having only eigenvectors represent cliques in G as our input. This way we do not have to change the matrix, and it is independent of G.

4.3 Hamiltonian simulation

In order to estimate the number of 0 eigenvalues of B, we want to use an algorithm called quantum phase estimation, to calculate the eigenvalues of B. However, as we will see in the next section, we can only use phase estimation on unitary matrices, and B is not unitary. In order to solve this problem, we use a quantum algorithm called Hamiltonian simulation.
Hamiltonian simulation takes a Hermitian matrix B, and produces a circuit that realizes $e^{2\pi i B t}$ for a certain $t \in \mathbb{R}$. If we have $B = PDP^{-1}$, with P invertible and D diagonal, we can write

$$e^{2\pi i B t} = e^{2\pi i PDP^{-1} t} = \sum_{k=0}^{\infty} \frac{(2\pi i t)^k}{k!} (PDP^{-1})^k = P \left(\sum_{k=0}^{\infty} \frac{(2\pi i t)^k}{k!} D^k \right) P^{-1} = P e^{2\pi i D t} P^{-1}$$

If $D = \mathrm{diag}(d_1, ..., d_l)$, we obtain $e^{2\pi i D t} = \mathrm{diag}(e^{2\pi i d_1 t}, ..., e^{2\pi i d_l t})$. Using this, we get that if $Bv = PDP^{-1}v = \lambda v$, then $e^{2\pi i B t} v = P e^{2\pi i D t} P^{-1} v = e^{2\pi i \lambda t} v$. In other words, if λ is an eigenvalue of B,

then $e^{2\pi i\lambda t}$ is an eigenvalue of $e^{2\pi iBt}$.
Since we want to know the number of eigenvalues of B that are 0, we need to make sure that for an eigenvalue λ of B, we have that $e^{2\pi i\lambda} = 1$ if and only if $\lambda = 0$. We can do this by making sure all eigenvalues are between 0 and 1. If we divide B by the largest eigenvalue, $\lambda_{\max}$, we make sure that for every eigenvalue of this new matrix, if $e^{2\pi i\lambda} = 1$, then $\lambda = 0$. This does mean that we need to know $\lambda_{\max}$, or at least an upper bound for it, in order to implement the algorithm. Fortunately, an upper bound is given by the Gershgorin circle theorem, which also shows that $\lambda_{\max} \in \mathcal{O}(n)$ [GCD20].

4.3.1 Complexity of Hamiltonian simulation

From [LC17] we know that we can do Hamiltonian simulation for time t of a d-sparse Hermitian matrix H with precision ϵ with query complexity $\mathcal{O}(td\|H\|_{\max} + \log(1/\epsilon))$. We call a matrix d-sparse if every row has at most d non-zero elements, and $\|H\|_{\max}$ is the largest element of H in absolute value. If the matrix elements are specified up to m bits of precision, the Hamiltonian simulation requires an additional $\mathcal{O}(n + m\text{polylog}(m))$ gates.
The matrix B is n-sparse, and we can implement sparse access using $\mathcal{O}(n)$ gates [GCD20]. Sparse access means that the algorithm has access to a function that takes two parameters a and b, and returns the bth non-zero element of the ath row, on top of just having access to a function that takes two parameters a and b and returns $u_{a,b}$, the element in the ath row, bth column.
For B, we have that all entries are -1, 0 or 1 before scaling with $\frac{1}{\lambda_{\max}}$. This means that $\|B\|_{\max} = \frac{1}{\lambda_{\max}} \leq 1$. The number of bits required to encode the matrix is equal to $\log_2(\lambda_{\max})$. By the Gershgorin circle theorem, since B is n-sparse, we have that $\lambda_{\max} \in \mathcal{O}(n)$, so $m \in \mathcal{O}(\log_2(n))$. This means that the additional gates required are $O(n + \log_2(n)\text{polylog}(\log_2(n)))$. We also know that $\|B\|_{\max} = \frac{1}{\lambda_{\max}} < 1$.
Therefore we can implement $e^{2\pi iBt}$ using

$$\mathcal{O}(tn^2/\lambda_{\max} + \log(1/\epsilon) + \log_2(n)\text{polylog}(\log_2(n))) = \mathcal{O}(tn^2/\lambda_{\max} + \log(1/\epsilon))$$

gates.
As for the required number of gates, from [GCD20] we know that the required number of qubits for Hamiltonian simulation is a bit, but not much, more than $2n + r + 1$, where r is the required precision of the entries of the matrix.

4.4 Phase estimation

Quantum phase estimation is an algorithm that given a unitary matrix U, and the ability to execute controlled unitary $C - U$ on a quantum comptuer, and a quantum state $|\phi\rangle$, such that $U|\phi\rangle = e^{2\pi i\lambda}|\phi\rangle$, can calculate (an ℓ bit approximation of) λ. This can be used to calculate the eigenvalues of the matrix B, and then use that to estimate the dimension of the kernel of B.
The input of the algorithm consists of an eigenstate, and ℓ ancilla qubits. After applying the algorithm, these ℓ ancilla qubits hold an ℓ bit approximation of the eigenvalue corresponding to the eigenstate with high probability. The circuit is given in figure 3. This implementation is from [NC00].
Quantum phase estimation starts by having ℓ ancilla qubits, initialised in the first register to $|0\rangle$. In the second register, we have an eigenstate $|\phi\rangle$ of our matrix U. We start by applying a Hadamard

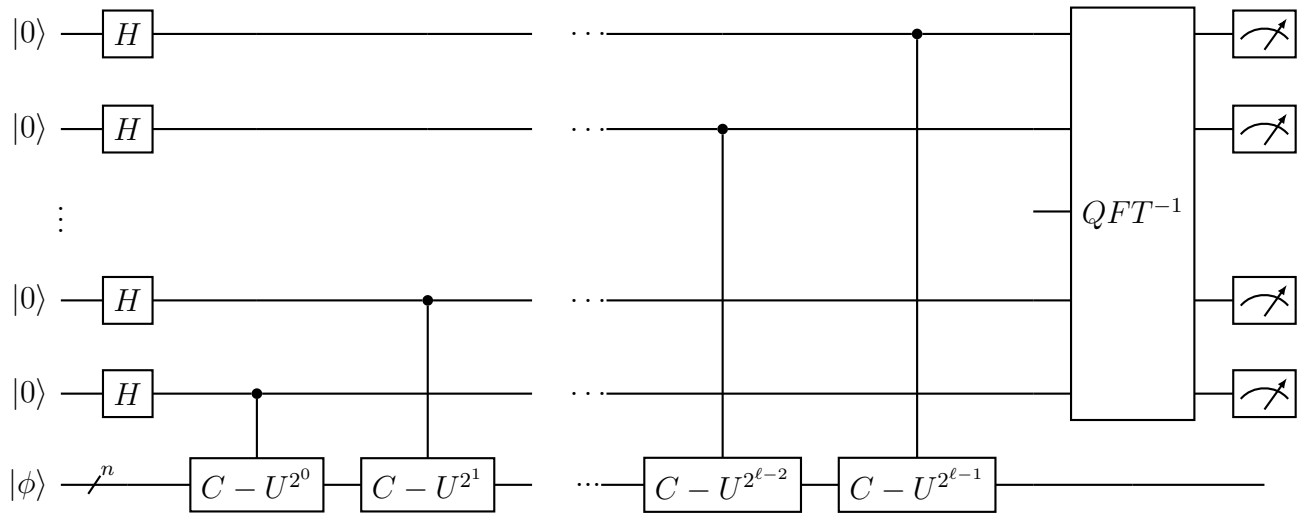

Figure 3: The phase estimation circuit

gate to each qubit in the first register: $(H^{\otimes l} \otimes I)(|0\rangle^{\otimes l} \otimes |\phi\rangle) = \frac{1}{\sqrt{2^\ell}}(|0\rangle + |1\rangle)^{\otimes \ell} \otimes |\phi\rangle$. Then we apply a controlled $U^{2^{\ell-j}}$ to each ancilla qubit and the second register, where j is the ancilla qubit number. Since $U|\phi\rangle = e^{2\pi i\lambda}|\phi\rangle$, we get that $C - U^{2^{\ell-j}} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|\phi\rangle = \frac{1}{\sqrt{2}}(|0\rangle|\phi\rangle + e^{2\pi i\lambda 2^{\ell-j}}|1\rangle|\phi\rangle)$. Therefore applying this to entire circuit gives

$$\frac{1}{\sqrt{2^\ell}}(|0\rangle + e^{2\pi i\lambda 2^{\ell-1}}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2\pi i\lambda 2^1}|1\rangle) \otimes (|0\rangle + e^{2\pi i\lambda 2^0}|1\rangle) \otimes |\phi\rangle = \frac{1}{\sqrt{2^\ell}} \sum_{k=0}^{2^\ell - 1} e^{2\pi i\lambda k}|k\rangle \otimes |\phi\rangle$$

The next step is applying the inverse Fourier transform. The quantum Fourier transform is given by

$$\text{QFT} : |j\rangle \mapsto \frac{1}{\sqrt{\ell}} \sum_{k=0}^{\ell-1} e^{2\pi ijk/\ell}|k\rangle$$

Since the quantum Fourier transform is unitary, the inverse is given by the complex conjugate, so the function

$$\text{QFT}^{-1} : |j\rangle \mapsto \frac{1}{\sqrt{\ell}} \sum_{k=0}^{\ell-1} e^{-2\pi ijk/\ell}|k\rangle$$

If we assume that $\lambda \cdot 2^\ell \in \mathbb{N}$ then $\text{QFT} : |2^\ell\lambda\rangle = \frac{1}{\sqrt{2^\ell}} \sum_{k=0}^{2^\ell-1} e^{\frac{2\pi i 2^\ell \lambda}{2^\ell}}|k\rangle = \frac{1}{\sqrt{2^\ell}} \sum_{k=0}^{2^\ell-1} e^{2\pi i\lambda}|k\rangle$, which is exactly the state we have before applying the inverse Fourier transform. Therefore if we apply the inverse Fourier transform to $\frac{1}{\sqrt{2^\ell}} \sum_{k=0}^{2^\ell-1} e^{2\pi i\lambda k}|k\rangle$, we get $|2^\ell\lambda\rangle$. If $2^\ell\lambda$ is not an integer, we measure an approximation of $2^\ell\lambda$.

4.4.1 Complexity of quantum phase estimation

From [NC00] we know that we can calculate $\tilde{\lambda}$ such that $|\lambda - \tilde{\lambda}| < \delta$ with probability $1 - \epsilon$ if we take $\ell = -\log_2(\delta) + \left\lceil \log_2\left(2 + \frac{1}{2\epsilon}\right)\right\rceil$. Since the smallest eigenvalue we need to differentiate from 0 is

$\frac{\lambda_{\min}}{\lambda_{\max}}$, we can choose $\delta = \frac{\lambda_{\max}}{\lambda_{\min}}$ to get $\ell = \left\lceil \log_2(\frac{\lambda_{\max}}{\lambda_{\min}}) \right\rceil + \left\lceil \log_2\left(2 + \frac{1}{2\epsilon}\right) \right\rceil$.
For the gate complexity, we have that since we need $U^{2^0}, ..., U^{2^{\ell-1}}$, the total time t we need for U is $\sum_{j=0}^{l-1} 2^j = 2^\ell - 1 \leq \frac{2\lambda_{\max}}{\lambda_{\min}} + \frac{\lambda_{\max}}{2\epsilon\lambda_{\min}}$. Since we know realizing U^t has gate complexity $\mathcal{O}(tn^2 \frac{1}{\lambda_{\max}} + \log(\frac{1}{\epsilon}))$, we get that the gate complexity of the execution of the U^j equals $\widetilde{\mathcal{O}}((\frac{2\lambda_{\max}}{\lambda_{\min}} + \frac{\lambda_{\max}}{2\epsilon\lambda_{\min}}) n^2 \frac{1}{\lambda_{\max}} + \log(\frac{1}{\epsilon})) = \widetilde{\mathcal{O}}(\frac{n^2}{\lambda_{\min}} + \frac{n^2}{\epsilon\lambda_{\min}}) = \widetilde{\mathcal{O}}(\frac{n^2}{\epsilon\lambda_{\min}})$, where $\widetilde{\mathcal{O}}$ is is the complexity class, but suppressing logarithmically growing factors. We examine this instead of the normal complexity since it simplifies the calculations, but still allows a fair comparison between the algorithms.
The number of qubits required is around $n + \ell = n + \left\lceil \log_2(\frac{\lambda_{\max}}{\lambda_{\min}}) \right\rceil + \left\lceil \log_2\left(2 + \frac{1}{2\epsilon}\right) \right\rceil$.

4.5 Complexity of the quantum algorithm for Betti number estimation

If we assume we do the state preparation clasically, executing Hamiltonian simulation and phase estimation requires $\widetilde{\mathcal{O}}\left(\frac{n^2}{\epsilon\lambda_{\min}}\right)$. On top of that, there is a classical time complexity of $\mathcal{O}\left(k^2 \frac{\binom{n}{k+1}}{N}\right)$ for the state preparation. Then, according to [GCD20], we estimate β_k/N up to additive precision δ using $M \in \mathcal{O}(\delta^{-2})$ repetitions of the execution of the state preparation, Hamiltonian simulation and phase estimation. If we assume no error in the quantum phase estimation, we get a total time complexity of estimating β_k/N at

$$\widetilde{\mathcal{O}}\left(\left(\frac{k^2\binom{n}{k+1}}{N} + \frac{n^2}{\lambda_{\min}}\right)/\delta^2\right).$$

If $\frac{1}{\lambda_{\min}} \in \mathcal{O}(1/\text{poly}(n))$, this algorithm is polynomial. However, in general we do not know if this is the case.
Since Hamiltonian simulation requires $2n + r + 1$ qubits, and phase estimation requires $n + \left\lceil \log_2(\frac{\lambda_{\max}}{\lambda_{\min}}) \right\rceil + \left\lceil \log_2\left(2 + \frac{1}{2\epsilon}\right) \right\rceil$. Since n qubits from the Hamiltonian simulation can be reused for phase estimation, the total number of qubits required for Hamiltonian simulation and quantum phase esimation is around $2n + r + 1 + \left\lceil \log_2(\frac{\lambda_{\max}}{\lambda_{\min}}) \right\rceil + \left\lceil \log_2\left(2 + \frac{1}{2\epsilon}\right) \right\rceil$ qubits.

5 Reducing the number of qubits

Now that we know how the algorithm works, we can look at reducing the number of qubits it requires. Specifically we look at the number of qubits required for the quantum phase estimation. Instead of using ℓ ancilla qubits, we use only one, and we replace applying the inverse quantum Fourier transform with a classical analysis of the measurement outcome.
In [OTT19], the authors show a way of changing the quantum phase estimation algorithm such that, instead of calculating all the bits of the eigenvalue at the same time, they can be calculated one after another. They then use post-processing to estimate the actual eigenvalue. We will use this approach, but our post-processing will be different, since we only care about the number of eigenvalues equal to zero, and not about their specific values. In this way we can reduce the number of ancilla qubits from ℓ to one.

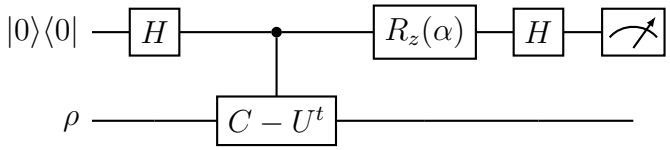

Figure 4: The quantum circuit of the new algorithm

5.1 Reducing the ancilla qubits

The new quantum circuit is given in figure 4. In order to use the circuit, we first need to calculate the probabilities of measuring 0 or 1, given the variables t and α.
In the circuit, $\rho = \frac{1}{N}\sum_{j=1}^{N} |\phi_j\rangle\langle\phi_j|$ is the input state, with $|\phi_j\rangle$ the eigenvectors of U and $N = |\mathrm{Cl}_k(G)|$, similarly to what is done in the existing algorithm. We let $e^{2\pi i\lambda_j}$ be the eigenvalue of $|\phi_j\rangle$, such that $U|\phi_j\rangle = e^{2\pi i\lambda_j}|\phi_j\rangle$. We assume that we have $0 \leq \lambda_j < 1$ for all j, since we scaled the matrix using Hamiltonian simulation. In the circuit, t and α are parameters we can change between subsequent executions of the algorithm.
In the next two subsections we show how we can use this circuit to estimate (the real part of) the function $g(t)$ given by

$$g(t) = \frac{1}{N}\sum_{j=1}^{N} e^{2\pi i t\lambda_j}$$

Then $\frac{\beta_k}{N}$ is equal to $\frac{1}{N}$ times the number of λ_j eigenvalues that are zero, which we call the constant part of $g(t)$. This means that if we estimate the constant part of $g(t)$, we can estimate $\beta_k = \dim(\ker \Delta_k^G)$. We look at ways to calculate the constant part of $g(t)$, showing one way to calculate this constant part if we know the period K of $g(t)$. However, this period does not always exist, so we also have another method that works regardless of whether the period exists or not, although at the cost of a small error.

5.2 Probabilities of the measurement of the circuit

First of all, we will calculate the probabilities of measuring 0 or 1 for U (so U has eigenvectors $|\phi_j\rangle$ with corresponding eigenvalues λ_j).
Subsections 5.2 and 5.3 are based on [OTT19].
We start in the state

$$|0\rangle\langle 0| \otimes \frac{1}{N}\sum_{j=1}^{N} |\phi_j\rangle\langle\phi_j|$$

Applying the Hadamard gate to the first qubit gives

$$\frac{1}{2}(|0\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 0| + |1\rangle\langle 1|) \otimes \frac{1}{N}\sum_{j=1}^{N} |\phi_j\rangle\langle\phi_j|$$

Applying the controlled U^t gives

$$\frac{1}{2N}\sum_{j=1}^{N}(|0\rangle\langle 0| + e^{-2\pi i\lambda_j t}|0\rangle\langle 1| + e^{2\pi i\lambda_j t}|1\rangle\langle 0| + |1\rangle\langle 1|) \otimes |\phi_j\rangle\langle\phi_j|$$

Applying the Z-rotation to the ancilla qubit then gives

$$\frac{1}{2N}\sum_{j=1}^{N}(|0\rangle\langle 0| + e^{-2\pi i\lambda_j t-\alpha}|0\rangle\langle 1| + e^{2\pi i\lambda_j t+\alpha}|1\rangle\langle 0| + |1\rangle\langle 1|)\otimes|\phi_j\rangle\langle\phi_j|$$

Lastly, we apply the Hadamard gate

$$\frac{1}{4N}\sum_{j=1}^{N}(|0\rangle\langle 0| + |1\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 1| + e^{-2\pi i\lambda_j t-\alpha}(|0\rangle\langle 0| - |1\rangle\langle 1|)+$$

$$e^{2\pi i\lambda_j t+\alpha}(|0\rangle\langle 0| - |1\rangle\langle 1|) + |0\rangle\langle 0| - |0\rangle\langle 1| - |1\rangle\langle 0| + |1\rangle\langle 1|)\otimes|\phi_j\rangle\langle\phi_j| =$$

$$\frac{1}{4N}\sum_{j=1}^{N}((2 + e^{-2\pi i\lambda_j t-\alpha} + e^{2\pi i\lambda_j t+\alpha})|0\rangle\langle 0| + (2 - e^{-2\pi i\lambda_j t-\alpha} - e^{2\pi i\lambda_j t+\alpha})|1\rangle\langle 1|)\otimes|\phi_j\rangle\langle\phi_j| =$$

$$\frac{1}{4N}\sum_{j=1}^{N}((2 + 2\cos(2\pi\lambda_j t+\alpha))|0\rangle\langle 0| + (2 - 2\cos(2\pi\lambda_j t+\alpha))|1\rangle\langle 1|)\otimes|\phi_j\rangle\langle\phi_j|$$

Then the probability of measuring $|0\rangle\langle 0|$ in the ancilla qubit equals

$$\begin{aligned}
P_{t,\alpha}(0) &= \frac{1}{4N}\sum_{j=1}^{N} 2 + 2\cos(2\pi t\lambda_j+\alpha) \\
&= \frac{1}{4N}\sum_{j=1}^{N} 2 + 2(2\cos^2(\frac{2\pi t\lambda_j+\alpha}{2}) - 1) \\
&= \frac{1}{N}\sum_{j=1}^{N} \cos^2(\frac{2\pi t\lambda_j+\alpha}{2})
\end{aligned}$$

The probability of measuring $|1\rangle\langle 1|$ equals

$$\begin{aligned}
P_{t,\alpha}(1) &= \frac{1}{4N}\sum_{j=1}^{N} 2 - 2\cos(2\pi t\lambda_j+\alpha) \\
&= \frac{1}{4N}\sum_{j=1}^{N} 2 + 2\cos(2\pi t\lambda_j+\alpha-\pi) \\
&= \frac{1}{4N}\sum_{j=1}^{N} 2 + 2(2\cos^2(\frac{2\pi t\lambda_j+\alpha-\pi}{2}) - 1) \\
&= \frac{1}{N}\sum_{j=1}^{N} \cos^2(\frac{2\pi t\lambda_j+\alpha-\pi}{2})
\end{aligned}$$

Combining these two, we get that the probability of measuring $m \in \{0, 1\}$ equals

$$P_{t,\alpha}(m) = \frac{1}{N}\sum_{j=1}^{N} \cos^2(\frac{2\pi t\lambda_j+\alpha-m\pi}{2})$$

Now that we know the measurement probabilities, we can use them to calculate $g(t)$.

5.3 Estimating $g(t)$

We will now show how to calculate $g(t)$ using the circuit provided in figure 4. In order to do this we will write $g(t) = \frac{1}{N}\sum_{j=1}^{N} e^{2\pi i t\lambda_j}$ in terms of $P_{t,\alpha}(m)$ for different values of t, α and m.
We have that

$$\begin{aligned}
P_{t,\alpha}(m) &= \frac{1}{N}\sum_{j=1}^{N}\cos^2(\frac{2\pi t\lambda_j + \alpha - m\pi}{2}) \\
&= \frac{1}{N}\sum_{j=1}^{N}\frac{1}{2}\cos(2\pi t\lambda_j + \alpha - m\pi) + \frac{1}{2} \\
&= \frac{1}{2} + \frac{1}{N}\sum_{j=1}^{N}\frac{1}{2}(\cos(\alpha + m\pi - 2\pi t\lambda_j) + \cos(\alpha + m\pi + 2\pi t\lambda_j)) \\
&- \frac{1}{2}(\cos(\alpha + m\pi - 2\pi t\lambda_j) + \cos(\alpha + m\pi + 2\pi t\lambda_j)) \\
&= \frac{1}{2} + \frac{1}{2N}\sum_{j=1}^{N}\cos(\alpha + m\pi)\cos(2\pi t\lambda_j) - \sin(\alpha + m\pi)\sin(2\pi t\lambda_j) \\
&= \frac{1}{2} + \frac{1}{2}\cos(\alpha + m\pi)\frac{1}{N}\sum_{j=1}^{N}\cos(2\pi t\lambda_j) - \frac{1}{2}\sin(\alpha + m\pi)\frac{1}{N}\sum_{j=1}^{N}\sin(2\pi t\lambda_j) \\
&= \frac{1}{2} + \frac{1}{2}\cos(\alpha + m\pi)\mathrm{Re}(g(t)) - \frac{1}{2}\sin(\alpha + m\pi)\,\mathrm{Im}(g(t))
\end{aligned}$$

We can now use the fact that

$$\begin{aligned}
g(t) &= \frac{1}{2} + \frac{1}{2}\mathrm{Re}(g(t)) - \frac{1}{2} + \frac{1}{2}\mathrm{Re}(g(t)) - i(\frac{1}{2} - \frac{1}{2}\,\mathrm{Im}(g(t))) + i(\frac{1}{2} + \frac{1}{2}\,\mathrm{Im}(g(t))) \\
&= P_{t,0}(0) - P_{t,0}(1) - iP_{t,\frac{\pi}{2}}(0) + iP_{t,\frac{\pi}{2}}(1)
\end{aligned}$$

This allows us to calculate $g(t)$ by calculating $P_{t,0}(0)$, $P_{t,0}(1)$, $P_{t,\frac{\pi}{2}}(0)$ and $P_{t,\frac{\pi}{2}}(1)$. Alternatively, we have

$$\mathrm{Re}(g(t)) = P_{t,0}(0) - P_{t,0}(1)$$

which for our purpose will be just as useful as $g(t)$, but only requires $P_{t,0}(0)$ and $P_{t,0}(1)$.

5.4 Overview

Now that we can calculate $g(t)$, we need to calculate the constant part of $g(t)$, since this is equal to $\frac{1}{N}$ times the number of zero eigenvalues of U. We found two ways to do this. The first way assumes that we know that the function $g(t)$ is periodic, and that we know the period K of $g(t)$. Using this, we show in section 5.5 how we can estimate $\frac{\beta_k}{N}$ using this period, and in section 5.6 how we can estimate it, which drastically reduces the number of calculations required.
However, this period K does not always exist. If it does, we can estimate it, for example using [QCM12], and then use this estimate to calculate the constant part. However, in section 5.7 and 5.8 we show another way to estimate $\frac{\beta_k}{N}$, derived from the first, and this time not requiring that the we know the period of $g(t)$, or that it exists. However, this way is computationally a bit more expensive than the case where we do know the period.

5.5 Calculating the constant part of $g(t)$ using the period

As a start, we show we can calculate β_k if we know a period K of $g(t)$, so a number $K > 0$ such that $g(K) = 1$, and the value of $g(t)$ for $t = 0, ..., K-1$.

Lemma 1. *Given a $K \in \mathbb{N}$ such that $g(K) = 1$, then $\frac{1}{K}\sum_{t=1}^{K} g(t) = \frac{\beta_k}{N}$ holds.*

Proof. Suppose we know the period K of $g(t)$, so $g(K) = 1$ with $K \in \mathbb{N}$. Since $\mathrm{Re}(e^{2\pi i\lambda_j t}) \leq 1$ for all j, we have that $g(K) = 1$ if and only if $e^{2\pi i\lambda_j K} = 1$ for all j. This is only true if $\lambda_j K = p$, with $p \in \mathbb{Z}_{\geq 0}$. Then we have that if $\lambda_j \neq 0$, then $e^{i\lambda_j}$ satisfies $(e^{i\lambda_j})^K - 1 = 0$. For $\lambda_j \neq 0$, we have that $e^{2\pi i\lambda_j} \neq 1$. Therefore it is a root of $\frac{X^K-1}{X-1} = X^{K-1} + X^{K-2} + \cdots + X^1 + 1$. This means that $\sum_{m=0}^{K-1} e^{2\pi i\lambda_j m} = 0$. If $e^{2\pi i\lambda_j} = 1$, so $\lambda_j = 0$, we get that $\sum_{m=0}^{K-1} e^{2\pi i\lambda_j m} = K$. Therefore we can write

$$\begin{aligned}
\sum_{t=1}^{K} g(t) &= \sum_{t=0}^{K-1} \frac{1}{N} \sum_{j=1}^{N} e^{2\pi i\lambda_j t} \\
&= \frac{1}{N} \sum_{j=1}^{N} \sum_{t=0}^{K-1} e^{2\pi i\lambda_j t} \\
&= \frac{1}{N} \sum_{j:\lambda_j \neq 0} \sum_{t=0}^{K-1} e^{2\pi i\lambda_j t} + \frac{1}{N} \sum_{j:\lambda_j = 0} \sum_{t=0}^{K-1} e^{2\pi i\lambda_j t} \\
&= \frac{1}{N} \sum_{j:\lambda_j \neq 0} K \\
&= \frac{K\beta_k}{N}
\end{aligned}$$

giving $\frac{1}{K}\sum_{t=1}^{K} g(t) = \frac{\beta_k}{N}$. □

5.6 Sampling the average of the $g(t)$

We now know we can find β_k by calculating and measuring $g(t)$ for all $t = 0, ..., K-1$. However, since K can get very large, calculating $g(t)$ for each $t = 0, ..., K-1$ will be slow. Instead, we can estimate the value of $\frac{1}{K}\sum_{t=1}^{K} g(t)$ by calculating $g(t)$ for randomly chosen t. This is done using Hoeffding's inequality.

Lemma 2. *We can estimate $\frac{1}{K}\sum_{t=0}^{K} g(t)$ up to probability δ with chance $1-\epsilon$ by calculating $Re(g(t))$ in total $\frac{2\log(\frac{2}{\epsilon})}{\delta^2}$ times.*

Proof. Let Y_i, for $i = 1, ..., p$, be random variables with the discrete uniform distribution from 0 to $K-1$. Define $X_i = \mathrm{Re}(g(Y_i))$. Then we have that $\mathbb{E}(X_i) = \frac{1}{K}\sum_{t=0}^{K-1} \mathrm{Re}(g(t))$.
We can now make use of Hoeffding's inequality: define $\overline{X} = \frac{1}{p}\sum_{i=1}^{p} X_i$. Then we have $\mathbb{E}(\overline{X}) = \frac{1}{K}\sum_{i=1}^{p} \mathbb{E}(X_i) = \frac{1}{K}\sum_{t=0}^{K-1} Re(g(t))$. These X_i are bounded by $[-1, 1]$. Hoeffding's inequality then states

$$\mathbb{P}\left(\left|\overline{X} - \mathbb{E}(\overline{X})\right| \geq \delta\right) \leq 2e^{-\frac{1}{2}p\delta^2}$$

If we choose $\epsilon = 2e^{-\frac{1}{2}p\delta^2}$, we get $\log(\frac{2}{\epsilon}) = \frac{1}{2}p\delta^2$, so $p = \frac{2\log(\frac{2}{\epsilon})}{\delta^2}$. Therefore, assuming we can calculate $P_{t,\alpha}(m)$ without error, we can calculate $\overline{X}$ such that the difference between $\overline{X}$ and $\frac{1}{K}\sum_{t=0}^{K-1} Re(g(t))$ is smaller than δ with probability more than $1-\epsilon = 1-2e^{-\frac{1}{2}\delta}$ by choosing $p = \frac{2\log(\frac{2}{\epsilon})}{\delta^2}$. □

Since $Re(g(t)) = P_{t,0}(0) - P_{t,0}(1)$, it takes on average $\frac{K-1}{2} + \frac{K-1}{2} = K-1$ executions of the controlled unitary $C-U$ to calculate X_i. Therefore the total number of executions of the controlled unitary $C-U$ in order to calculate $\overline{X}$ such that the difference between $\overline{X}$ and $\frac{1}{K}\sum_{t=0}^{K-1} Re(g(t))$ is smaller than δ with probability larger than $1-\epsilon$, is equal to $(K-1)\cdot\frac{2\log(\frac{2}{\epsilon})}{\delta^2}$.

5.7 The aperiodic case

The above method works well, but only if we actually have a period K. If we do not have this period, there is another way to estimate $\frac{\beta_k}{N}$, by calculating $\frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt$. We do this by showing that $\frac{1}{T}\int_0^T \cos(2\pi\lambda_j t)dt$ goes to zero for non-zero λ_j when T is large, while $\frac{1}{T}\int_0^T \cos(2\pi\lambda_{j'} t)dt = 1$ for $\lambda_{j'} = 0$, leading to $\frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt$ approaching $\frac{\beta_k}{N}$ when T is large. This then gives us an estimate of the constant part of $\mathrm{Re}(g(t))$.
In the next subsection we show how we can approximate this integral using Hoeffding's inequality, much like the periodic case.

Lemma 3. *For $T \in \mathbb{R}_{>0}$ and $g(t)$ as before, we have*

$$\left|\frac{1}{T}\int_0^T Re(g(t))dt - \frac{\beta_k}{N}\right| \leq \frac{\lambda_{\max}}{2\pi T\lambda_{\min}}$$

so we can approximate $\frac{\beta_k}{N}$ with the integral $\frac{1}{T}\int_0^T Re(g(t))dt$ by taking a large T.

Proof. We start with showing how we can extract $\frac{\beta_k}{N}$:

$$\begin{aligned}
\frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt &= \frac{1}{T}\int_0^T \frac{1}{N}\sum_{j=1}^{N}\cos(2\pi\lambda_j t)dt \\
&= \frac{1}{NT}\sum_{j=1}^{N}\int_0^T \cos(2\pi\lambda_j t)dt \\
&= \frac{1}{NT}\sum_{j:\lambda_j\neq 0}\int_0^T \cos(2\pi\lambda_j t)dt + \frac{1}{NT}\sum_{j:\lambda_j=0}\int_0^T \cos(2\pi\lambda_j t)dt \\
&= \frac{1}{NT}\sum_{j:\lambda_j\neq 0}\int_0^T \cos(2\pi\lambda_j t)dt + \frac{1}{NT}\sum_{j:\lambda_j=0} T \\
&= \frac{1}{NT}\sum_{j:\lambda_j\neq 0}\int_0^T \cos(2\pi\lambda_j t)dt + \frac{\beta_k}{N}
\end{aligned}$$

In order to get the number of zero-eigenvalues, we need to show that

$$\lim_{T\to\infty} \frac{1}{NT} \sum_{j:\lambda_j \neq 0} \int_0^T \cos(2\pi\lambda_j t)dt = 0$$

To do this, for each non-zero λ_j we look at $T_j = \frac{c_j}{\lambda_j}$, where c_j is the largest integer such that $T_j \leq T$. This means that

$$\int_0^{T_j} \cos(2\pi\lambda_j t)dt = \frac{1}{2\pi\lambda_j}(\sin(2\pi c_j) - \sin(0)) = 0$$

for all j. Then we obtain

$$\begin{aligned}\frac{1}{NT} \sum_{j:\lambda_j \neq 0} \int_0^T \cos(2\pi\lambda_j t)dt &= \frac{1}{NT} \sum_{j:\lambda_j \neq 0} \int_0^{T_j} \cos(2\pi\lambda_j t)dt + \frac{1}{NT} \int_{T_j}^T \cos(2\pi\lambda_j t)dt \\ &= \frac{1}{NT} \sum_{j:\lambda_j \neq 0} \int_{T_j}^T \cos(2\pi\lambda_j t)dt.\end{aligned}$$

We will now look at the integral, and give an upper bound. We have

$$\left| \int_{T_j}^T \cos(2\pi\lambda_j t)dt \right| = \left| \frac{\sin(2\pi c_j)}{2\pi\lambda_j} - \frac{\sin(2\pi\lambda_j T)}{2\pi\lambda_j} \right| = \left| \frac{\sin(2\pi\lambda_j T)}{2\pi\lambda_j} \right| \leq \frac{1}{|2\pi\lambda_j|} \leq \left| \frac{\lambda_{\max}}{2\pi\lambda_{\min}} \right|$$

since $\lambda_j \leq \frac{\lambda_{\min}}{\lambda_{\max}}$. Applying this inequality to the sum of the integrals, we get

$$\left| \frac{1}{NT} \sum_{j:\lambda_j \neq 0} \int_{T_j}^T \cos(2\pi\lambda_j t)dt \right| \leq \frac{1}{NT} \sum_{j:\lambda_j \neq 0} \left| \int_{T_j}^T \cos(2\pi\lambda_j t)dt \right| \leq \frac{1}{2\pi NT} \sum_{j:\lambda_j \neq 0} \frac{1}{|\lambda_j|} \leq \left| \frac{\lambda_{\max}}{2\pi T\lambda_{\min}} \right|$$

Combining this with the integral from earlier gives

$$\left| \frac{1}{NT} \sum_{j:\lambda_j \neq 0} \int_0^T \cos(2\pi\lambda_j t)dt \right| \leq \left| \frac{\lambda_{\max}}{2\pi T\lambda_{\min}} \right|$$

so we have an upper bound for the error. This means that

$$\left| \frac{1}{T} \int_0^T \mathrm{Re}(g(t))dt - \frac{\beta_k}{N} \right| \leq \frac{\lambda_{\max}}{2\pi T\lambda_{\min}}.$$

□

5.8 Estimating the integral

Since we cannot calculate the integral $\frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt$ directly, we will estimate it by calculating $\mathrm{Re}(g(t))$ for random choices of t. For this, we again use Hoeffding's inequality, in a way similar to the aperiodic case.

Lemma 4. *We can estimate* $\frac{1}{T}\int_0^T Re(g(t))dt$ *up to precision* δ *with probability* $1-\epsilon$ *by calculating* $g(t)$ $p = \frac{2\cdot\log(\frac{2}{\epsilon})}{\delta^2}$ *times, for random* t*.*

Proof. Let Y_j for $j = 1, ..., p$ be random variables with the continuous uniform distribution from 0 to T. Define $X_j = \mathrm{Re}(g(Y_i))$. Then we have that $\mathbb{E}(X_j) = \frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt$. Now we can apply Hoeffding's inequality: we define $\overline{X} = \frac{1}{p}\sum_{j=1}^p X_j$. Then we have that $\mathbb{E}(\overline{X}) = \frac{1}{p}\sum_{j=1}^p \frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt = \frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt$. Since the X_j are bounded by $[-1, 1]$, Hoeffding's inequality gives that

$$\mathbb{P}\left(\left|\overline{X} - \mathbb{E}[\overline{X}]\right| \geq \delta\right) \leq 2e^{-\frac{1}{2}p\delta^2}$$

Choosing $\epsilon = 2e^{-\frac{1}{2}p\delta^2}$, we get $\log(\frac{2}{\epsilon}) = \frac{1}{2}p\delta^2$, so $p = \frac{2\log(\frac{2}{\epsilon})}{\delta^2}$. Therefore, we can estimate $\frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt$ up to precision δ with probability $1-\epsilon$ by taking $p = \frac{2\cdot\log(\frac{2}{\epsilon})}{\delta^2}$. □

6 Complexity of the new algorithm

Now that we know the algorithm can calculate $\frac{\beta_k}{N}$, we are interested in the gate complexity of the circuit, especially when compared to existing algorithms, so we determine the time complexity of the aperiodic version of the algorithm.
For now, we assume Hamiltonian simulation without error.

6.1 Error of the average of random variables

In order to determine the error of the averages of several measurements, we will need the following lemma.

Lemma 5. *For* $j = 1, ..., p$*, let* X_j *and* Y_j *such that* $\mathbb{P}(|X_j - Y_j| \geq \delta) \leq \epsilon$*, and define* $\overline{X} = \frac{1}{p}\sum_{j=1}^p X_j$ *and* $\overline{Y} = \frac{1}{p}\sum_{j=1}^p Y_j$*. Then* $\mathbb{P}(|\overline{X} - \overline{Y}| \geq \delta) \leq p\epsilon$*.*

Proof. Let X_j and Y_j as in the lemma. Applying Boole's inequality gives

$$\mathbb{P}\left(\bigcup_{j=1}^p \{|X_j - Y_j| \geq \delta\}\right) \leq \sum_{j=1}^p \mathbb{P}(|X_j - Y_j| \geq \delta) \leq p\epsilon$$

Therefore, we can write

$$1 - \mathbb{P}\left(\bigcup_{j=1}^p \{|X_j - Y_j| \geq \delta\}\right) = \mathbb{P}\left(\bigcap_{j=1}^p \{|X_j - Y_j| \leq \delta\}\right) \geq 1 - p\epsilon$$

Since $\mathbb{P}(\{X \leq x\} \cap \{Y \leq y\}) \leq \mathbb{P}(X + Y \leq x + y)$, we obtain

$$\begin{aligned} 1 - p\epsilon &\leq \mathbb{P}\left(\bigcap_{j=1}^{p} \{|X_j - Y_j| \leq \delta\}\right) \\ &\leq \mathbb{P}\left(\sum_{j=1}^{p} |X_j - Y_j| \leq p\delta\right) \\ &\leq \mathbb{P}\left(\left|\sum_{j=1}^{p} X_j - Y_j\right| \leq p\delta\right) \\ &\leq \mathbb{P}\left(|\overline{X} - \overline{Y}| \leq \delta\right) \end{aligned}$$

This then gives $\mathbb{P}\left(|\overline{X} - \overline{Y}| \geq \delta\right) \leq p\epsilon$, proving the lemma. □

6.2 Sampling $P_{t,0}$

Let $t \in \mathbb{R}$ be fixed. For $j = 1, ..., q$ we define the stochasts $Z_j^{(t)}$ i.i.d. as $\mathbb{P}(Z_j^{(t)} = m) = P_{t,0}(m)$ for $m \in \{0, 1\}$, so $\mathbb{E}(Z_j^{(t)}) = P_{t,0}(1)$. We now want to know how many times we need to sample $Z_j^{(t)}$ to estimate $P_{t,0}(0)$ up to a certain precision.
Define $\overline{Z}^{(t)} = \frac{1}{q}\sum_{j=1}^{q} Z_j^{(t)}$. We then have $\mathbb{E}(\overline{Z}^{(t)}) = \mathbb{E}(Z_j^{(t)}) = P_{t,0}(m)$. Hoeffding's inequality then gives us

$$\mathbb{P}(|\overline{Z}^{(t)} - \mathbb{E}(Z_j^{(t)})| \geq \delta_1) = \mathbb{P}(|\overline{Z}^{(t)} - P_{t,0}(1)| \geq \delta_1) \leq \epsilon_1$$

Since $\mathrm{Re}(g(t)) = P_{t,0}(0) - P_{t,0}(1) = 1 - 2P_{t,0}(1)$, we have that

$$\mathbb{P}(|(1 - 2\overline{Z}^{(t)}) - \mathrm{Re}(g(t))| \geq 2\delta_1) \leq \epsilon_1$$

if we take $q = \frac{\log(\frac{2}{\epsilon_1})}{2\delta_1^2}$. Sampling $Z_j^{(t)}$ has gate complexity $\mathcal{O}\left(\frac{k^2\binom{n}{k+1}}{N} + tn^2\right)$, so the gate complexity of estimating $P_{t,0}$ with precision δ_1 (or $\mathrm{Re}(g(t))$ with precision $2\delta_1$) and probability ϵ_1 has complexity

$$\mathcal{O}\left(\frac{k^2\binom{n}{k+1}}{N} + qtn^2\right) = \mathcal{O}\left(\frac{k^2\binom{n}{k+1}}{N} + \frac{tn^2\log\left(\frac{2}{\epsilon_1}\right)}{2\delta_1^2}\right) = \mathcal{O}\left(\frac{k^2\binom{n}{k+1}}{N} + \frac{tn^2\log\left(\frac{2}{\epsilon_1}\right)}{\delta_1^2}\right).$$

The reason the complexity for the state preparation is not multiplied by q, is that we run the circuit with the same input, so the state preparation only has to be executed once.

6.3 Estimating the integral

In order to estimate $\frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt$, we estimate $\overline{X}$. Applying the lemma to $X_j = \mathrm{Re}(g(Y_j))$ and $1 - 2\overline{Z}^{Y_j}$: since $\mathbb{P}(|(1 - 2\overline{Z}^{(Y_i)} - \mathrm{Re}(g(Y_i))| \geq 2\delta_1) \leq \epsilon_1$, we get

$$\mathbb{P}\left(\left|\overline{X} - \frac{1}{p}\sum_{j=1}^{p} 1 - 2\overline{Z}^{(Y_j)}\right| \geq 2\delta_1\right) \leq p\epsilon_1$$

Calculating $\overline{Z}^{(Y_j)}$ has a gate complexity of $\mathcal{O}\left(\frac{k^2\binom{n}{k+1}}{N} + \frac{Tn^2\log(\frac{2}{\epsilon_1})}{\delta_1^2}\right)$. Since we need to calculate this p times, we find a total of $\mathcal{O}\left(p\left(\frac{k^2\binom{n}{k+1}}{N} + \frac{Tn^2\log\left(\frac{2}{\epsilon_1}\right)}{\delta_1^2}\right)\right)$ to calculate $\overline{X}$.

6.4 Putting it together

If we want $\left|\overline{X} - \frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt\right| \leq \delta_2$ with chance $1-\epsilon_2$, we need to choose $p = \frac{2\log(\frac{2}{\epsilon_2})}{\delta_2^2}$. This gives

$$\mathbb{P}\left(\left|\overline{X} - \frac{1}{p}\sum_{j=1}^{p} 1 - 2\overline{Z}^{(Y_j)}\right| \geq 2\delta_1\right) \leq p\epsilon_1 = \frac{2\epsilon_1\log(\frac{2}{\epsilon_2})}{\delta_2^2}$$

and

$$\mathbb{P}\left(\left|\overline{X} - \frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt\right| \geq \delta_2\right) \leq \epsilon_2$$

Together, this gives

$$\mathbb{P}\left(\left|\frac{1}{p}\sum_{j=1}^{p} 1 - 2\overline{Z}^{(Y_j)} - \frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt\right| \geq 2\delta_1 + \delta_2\right) \leq \frac{2\epsilon_1\log(\frac{2}{\epsilon_2})}{\delta_1^2} + \epsilon_2$$

We can calculate this with a complexity of $\mathcal{O}\left(\frac{\log(\frac{2}{\epsilon_2})}{\delta_2^2}\left(\frac{k^2\binom{n}{k+1}}{N} + \frac{Tn^2\log(\frac{2}{\epsilon_1})}{\delta_1^2}\right)\right)$.

Last of all, we add the error between $\left|\frac{1}{T}\int_0^T \mathrm{Re}(g(t))dt - \frac{\beta_k}{N}\right| < \frac{1}{2\pi T\frac{\lambda_{\min}}{\lambda_{\max}}} = \frac{\lambda_{\max}}{2\pi T\lambda_{\min}} = \delta_3$. We get this by taking $T = \frac{\lambda_{\max}}{2\pi\lambda_{\min}\delta_3}$, so we get

$$\mathbb{P}\left(\left|\frac{1}{p}\sum_{j=1}^{p} 1 - 2\overline{Z}^{(Y_j)} - \frac{\beta_k}{N}\right| \geq 2\delta_1 + \delta_2 + \delta_3\right) \leq \frac{2\epsilon_1\log\left(\frac{\epsilon_2}{2}\right)}{\delta_2^2} + \epsilon_2$$

with time complexity

$$\mathcal{O}\left(\frac{\log(\frac{2}{\epsilon_2})}{\delta_2^2}\left(\frac{k^2\binom{n}{k+1}}{N} + \frac{\lambda_{\max}n^2\log(\frac{2}{\epsilon_1})}{\lambda_{\min}\delta_1^2\delta_3}\right)\right).$$

In order to simplify this, we take $\delta = 2\delta_1 + \delta_2 + \delta_3$ and $\epsilon = \frac{2\epsilon_1\log\left(\frac{\epsilon_2}{2}\right)}{\delta_2^2} + \epsilon_2$, equally splitting the precision and the error over the terms. This means we get $\delta_1 = \frac{\delta}{4}$, $\delta_2 = \delta_3 = \frac{\delta}{2}$ and $\frac{2\epsilon_1\log\left(\frac{\epsilon_2}{2}\right)}{\delta_2^2} = \epsilon_2 = \frac{\epsilon}{2}$. Since $\frac{2\epsilon_1\log\left(\frac{\epsilon_2}{2}\right)}{\delta_2^2} = \frac{8\epsilon_1\log\left(\frac{\epsilon_2}{2}\right)}{\delta^2} = \frac{\epsilon}{2}$, we get $\epsilon_1 = \frac{\epsilon\delta^2}{16\log\left(\frac{\epsilon_2}{2}\right)} = \frac{\epsilon\delta^2}{16\log\left(\frac{\epsilon}{4}\right)}$. This gives

$$\mathbb{P}\left(\left|\frac{1}{p}\sum_{j=1}^{p} 1 - 2\overline{Z}^{(Y_j)} - \frac{\beta_k}{N}\right| \geq \delta\right) \leq \epsilon$$

with time complexity

$$\mathcal{O}\left(\frac{\log(\frac{1}{\epsilon})}{\delta^2}\left(\frac{k^2\binom{n}{k+1}}{N} + \frac{\lambda_{\max}n^2\log\left(\frac{\log\left(\frac{\epsilon}{4}\right)}{\epsilon\delta^2}\right)}{\lambda_{\min}\delta^3}\right)\right).$$

6.5 Summary

By first applying the state preparation (where we choose the classical version, without Grover's algorithm), then applying Hamiltonian simulation and finally applying the modified phase estimation explained in section 5, we get the following theorem.

Theorem 1. *We can estimate β_k/N up to precision δ with chance ϵ with gate complexity*

$$\widetilde{\mathcal{O}}\left(\frac{\log(\frac{1}{\epsilon})}{\delta^2}\left(\frac{k^2\binom{n}{k+1}}{N}+\frac{\lambda_{\max}n^2\log\left(\frac{\log(\frac{\epsilon}{4})}{\epsilon\delta^2}\right)}{\lambda_{\min}\delta^3}\right)\right)=\widetilde{\mathcal{O}}\left(\left(\frac{k^2\binom{n}{k+1}}{N}+\frac{\lambda_{\max}n^2}{\lambda_{\min}\delta^3}\right)/\delta^2\right)$$

given that we know $\lambda_{\max}$ and $\lambda_{\min}$.

By the Gershgorin circle theorem, we have $\lambda_{\max}\in\mathcal{O}(n)$, giving a total complexity of

$$\widetilde{\mathcal{O}}\left(\left(\frac{k^2\binom{n}{k+1}}{N}+\frac{n^3}{\lambda_{\min}\delta^3}\right)/\delta^2\right)$$

This means that if $\lambda_{\min}\in\Omega\left(\frac{1}{\text{poly}(n)}\right)$, and $\binom{n}{k+1}/N\in\mathcal{O}(\text{poly}(n))$, this is polynomial, which are the same conditions for the algorithm to be polynomial as the original algorithm. The number of qubits required is around $2n+r+2$, where r is the number of qubits required for the entries of the matrix B for Hamiltonian simulation, since we add one extra qubit for the phase estimation.

7 Conclusions and Further Research

We were able to reduce the number of ancilla qubits required for the phase estimation step of the algorithm from $n+\left\lceil\log_2(\frac{\lambda_{\max}}{\lambda_{\min}})\right\rceil+\left\lceil\log_2\left(2+\frac{1}{2\epsilon}\right)\right\rceil$ to $n+1$, so with $\frac{\lambda_{\max}}{\lambda_{\min}}\in\mathcal{O}(n)$, we go from $n+\log(n)$ to $n+1$. This brings the total cost of the number of required qubits down from around $2n+r+1+\left\lceil\log_2(\frac{\lambda_{\max}}{\lambda_{\min}})\right\rceil+\left\lceil\log_2\left(2+\frac{1}{2\epsilon}\right)\right\rceil$ down to around $2n+r+2$. However, this change comes at the cost of a higher complexity of the resulting algorithm. The time complexity of the original algorithm, with precision δ, is equal to

$$\widetilde{\mathcal{O}}\left(\left(\frac{k^2\binom{n}{k+1}}{N}+\frac{n^2}{\lambda_{\min}}\right)/\delta^2\right),$$

but the time complexity of the new algorithm, again with precision δ, is equal to

$$\widetilde{\mathcal{O}}\left(\left(\frac{k^2\binom{n}{k+1}}{N}+\frac{n^3}{\lambda_{\min}\delta^3}\right)/\delta^2\right).$$

Do note that the new algorithm does not require more gates, in fact, it requires less, but the circuit does need to run more often to get the same level of precision.
Some questions, however, still remain open. For example, we do not know if $\lambda_{\min}\in\mathcal{O}(1/\text{poly}(n))$,

which is a requirement for the algorithm to be polynomial. On top of that, it is also unknown if the Dirac operator satisfies the conditions required for other methods of Hamiltonian simulation, which would reduce the number of required qubits even further.
However, despite these unknowns, this new algorithm reduced the required number of qubits, bringing it very close to number of qubits we are expecting to see in quantum computers in the near future, and bringing us one step closer to being able to implement this algorithm on an actual quantum computer.

References

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, and et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505510, Oct 2019.

[GCD20] Casper Gyurik, Chris Cade, and Vedran Dunjko. Towards quantum advantage for topological data analysis. 2020.

[Ghr08] Robert Ghrist. Barcodes: The persistent topology of data. *BULLETIN (New Series) OF THE AMERICAN MATHEMATICAL SOCIETY*, 45, 02 2008.

[GK19] Sam Gunn and Niels Kornerup. Review of a quantum algorithm for betti numbers. 2019.

[LC17] Guang Hao Low and Isaac L Chuang. Optimal hamiltonian simulation by quantum signal processing. *Physical review letters*, 118(1):010501, 2017. `arXiv:1606.02685`.

[LGZ16] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. Quantum algorithms for topological and geometric analysis of data. *Nature communications*, 7(1):1–7, 2016. `arXiv:1408.3106`.

[NC00] M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information*. Cambridge Series on Information and the Natural Sciences. Cambridge University Press, 2000.

[OTT19] Thomas E OBrien, Brian Tarasinski, and Barbara M Terhal. Quantum phase estimation of multiple eigenvalues for small-scale (noisy) experiments. *New Journal of Physics*, 21(2):023022, Feb 2019.

[Pre18] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, Aug 2018.

[QCM12] Barry Quinn, I. Clarkson, and Robert Mckilliam. Estimating period from sparse, noisy timing data. *2012 IEEE Statistical Signal Processing Workshop, SSP 2012*, pages 193–196, 08 2012.