Applications of Quantum Monte Carlo (SQA Undergraduate Research Scholarship)

Roy Lim
Supervisor: Dr Clément L Cannone

7 February 2023

ABSTRACT

Monte Carlo Method consists of a large group of computational algorithms that solve problems in a wide array of fields such as physics and quantitative finance. In 2022, a new algorithm was created with Grover's Algorithm as a base that allows for the mean of a random variable to be estimated with fewer or the same number of samples and a tighter error bound. This paper will focus on how an improvement in mean estimation can provide a speed-up in solving existing problems.

CONTENTS

1 INTRODUCTION

This paper's primary goal is to investigate the applications of Quantum Monte Carlo method as specified in Kothari & O'Donnell (2022) to take advantage of its speed-up.

Suppose that we want to estimate the means of some random variable **y** . The easiest method is simply to enumerate all the outcomes and their probability of occurring like so:

$$\mathbb{E}[X] = \sum_i x_i \times \Pr[X = x_i]$$

However, in practice, this method is too costly if the underlying distribution is unknown. Instead, it is far quicker to try to take a large number of samples and take the average of all those values, where the more samples we take, the more certain we are about our value for the mean. This is an example of a Monte Carlo method, a family of algorithms where a quantity can be estimated by taking samples. The logical continuation to the problem above would be, how many samples do we need to take and can we reduce the number of samples we need without compromising on the accuracy of our estimate?

2 PRELIMINARIES

This section includes some general information required to understand the rest of the paper.

2.1 Big Oh Notation

To evaluate the time or space complexity (i.e how long an algorithm takes or how much space it requires to run), we typically use what is known as Big Oh Notation in Computer Science. The most common is **Big Oh** $O(\cdot)$ which describes an upper bound or worst case performance for a specific algorithm. For example, if an algorithm has linear time or $O(n)$ time, this indicates that algorithm's run time with grow linearly as the input size n increases (i.e. if the input increases by a factor of 10, the run time will increase by 10 times). The notation only provides an estimate as it ignores any constant factor or any factors that grow slower than the leading term (for example, $O(n+\sqrt{n})$ may be simplified to just $O(n)$).

There are a few general ways of describing the run-time of algorithms, below is a list in order of increasing time:

- Constant Time $O(1)$ (operation does not change with input size);
- Sublinear Time $O(\sqrt{n}), O(\log n), \cdots < O(n)$;
- Polynomial Time $O(n^k)$ for $k \in \mathbb{Z}^+$;
- Exponential Time $O(2^n), O(3^n), \ldots$;
- Factorial Time $O(n!)$.

Big Ω Notation describes lower bound/best case complexity while **Big** Θ describes the complexity that can be describe by both $O(\cdot)$ and $\Omega(\cdot)$ (i.e. the algorithm has a tight upper and lower bound on its time or space complexity.

2.2 Basic Models for Quantum Computing

In Quantum Computing, use quantum bits (qubits), which can represent a superposition between states 0 and 1 instead of being either 0 or 1 like classical bits.

These qubits are manipulated using Gates and Oracles. Similar to classical computing Gates performs some logical operation on the bits, which causes the state of the qubits to change. This include Pauli's Matrices (which perform a logical operation on a singular qubit) and Hadamard

Oracles extend this concept of gates by using multiple gates to create a black box that takes in a set of inputs and gives some output.

To use a quantum algorithm, there therefore 4 distinct steps:

(i) Start with a set of inputs (for example n-bits);
(ii) Place them in a superposition of states (this can be done using a Hadamard Gate or other quantum logic gates);
(iii) Apply some algorithm that uses a sequence of gates or oracles;
(iv) Measure one or more qubits (this collapses the state into a classical output).

Note that qubits and their states are inherently probabilistic, so what the algorithm is doing is maximizing the probability of getting the correct answer. There is no guarantee that the answer is correct but there is a high probability that there is.

2.3 Monte Carlo Method

Monte Carlo refers to a set of algorithms that were invented shortly after the increase in computational power in the 1940s. These algorithms utilise statistical sampling to determine parameters that are often difficult to compute analytically, such as nuclear chain reactions, which was one of the first applications of Monte Carlo methods. Other scenarios include determining parameters where the trade-off between time and accuracy is required.

An example of Monte Carlo method can be found in estimating π. Suppose there is a quarter of a circle with a radius of 1 unit in a 1 unit by 1 unit cube. Afterwards, n number of points are randomly placed onto the cube. Given the area of a quarter of a circle is $\pi/4$ and the area of the cube is 1 unit, we expect that $\pi/4$ many dots out of the n dots to fall within the quarter circle. Therefore $\pi/4 =$ fraction of dots inside the circle $\implies \pi =$ $4 \times$ fraction of dots inside the circle

3 QUANTUM MONTE CARLO

3.1 Overview of Methodology

In a new paper (Kothari & O'Donnell (2022)), a new algorithm is hypothesised that can improve existing Monte Carlo methods for mean estimation provided quantum access and the "source code" for the random variable. In this case, the "source code" refers to some classical, randomized circuit or a unitary, quantum circuit that outputs a set of bits (note for the quantum circuit, it may output a super-position of qubits which can then be measured to yield a set of bits) that generated random samples of the random variable we are trying to estimate the mean of. The "source code" is required as created what is known as the "phase oracle" or ROT operator, which is a diagonal unitary that operates on the qubits. By modifying Grover's Algorithm (Grover (1996)) to include the ROT operator, each step of Grover's Algorithm will utilise both the REFL operator (commonly known as the diffusion operator) and the ROT operator, this incorporates complex phases into an algorithm which consequently allows for the algorithm to the mean $(\hat{\mu} = y_1 + y_2 + \cdots + y_n/n)$ of a random variable $\mathbf{y}$ (provided that the code for the generation of $\mathbf{y}$). The ability to estimate random variables means has other implications that will be outlined in section 3.3 while proof of how the quantum algorithm is as good if not better than classical algorithms can be found in section 3.2.

The improvement can be summarised in **Theorem 1.1** and **Theorem 1.3** in Kothari & O'Donnell (2022) where its states:

For **Theorem 1.1**: *Given "the code" for a random variable* $\mathbf{y}$ *, the algorithm uses* $O(n)$ *samples and outputs an estimate* $\hat{\mu}$ *such that:*

$$\Pr\left[\left|\hat{\mu} - \mu\right| > \sigma/n\right] \leq 1/3$$

For **Theorem 1.3**: *Given "the code" for a random variable* $\mathbf{y}$ *(where* $\mathbb{E}[y^2] \leq 1$*, and a parameter* $\varepsilon > 0$*, in* $O(1/\varepsilon)$ *samples and distinguish (with confidence at least 2/3) between the cases*

(i) $|\mu| \leq \varepsilon/2$*;*
(ii) $\varepsilon \leq |\mu| \leq 2\varepsilon$*.*

3.2 Consequences

The algorithm provides an estimate for $\hat{\mu}$ where $\mu = \mathbb{E}[y]$ by running the code $O(n)$ times with a high probability of satisfying $|\hat{\mu} - \mu| \leq \sigma/n$, which is an improvement over the results achievable by a classical computer (being $|\hat{\mu} - \mu| \leq \sigma/\sqrt{n}$ with the same number of samples).

High probability in this context is the probability of not satisfying the inequality is less than 1/3. This can be improved from 1/3 to any $\delta > 0$ by repeating the algorithms $O(\frac{1}{\delta})$ times and taking the median (this is known as the Median Trick, more in the appendix 6.3).

This also means that any algorithms that utilise Chebyshev's Inequality can be improved as Chebyshev's Inequality is based on classical computation.

$$Pr(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2} (\textbf{Chebyshev's Inequality})$$

Letting $X = \hat{\mu} = y_1 + \cdots + y_n/n$, we can determine that:

$$Var[X] = \sigma^2/n$$

We can then modify Chebyshev's Inequality to be (letting $k = \sqrt{3}$ and $X = \hat{\mu}$).

$$Pr(|\hat{\mu} - \mu| \geq \sqrt{3}\sigma/\sqrt{n}) \leq 1/3$$

The method proposed in Kothari & O'Donnell (2022) concludes that the algorithm outputs an estimate $\hat{\mu}$ such that $Pr(|\hat{\mu} - \mu| \geq \sigma/n) \leq 1/3$. Hence it is clear that the algorithm provides lower error with the sample size of $O(n)$.

3.3 What problems can the algorithm be applied to?

Using the algorithms, there are a few things can we can achieve efficiently. General Mean Estimation can be done in $O(n)$ queries to the code such that the estimate $\hat{\mu}$:

$$|\hat{\mu} - \mu| \leq \sigma/n$$

This identity can be extrapolated to other problems such as:

- determining the mean of a Bernoulli random variable in $O(n)$ queries

- providing an estimate for μ with a specific bound, $|\hat{\mu} - \mu| \leq \sigma_{\text{bound}}/n$ for some σ_{bound}.
- verifying for some $\varepsilon \in (0,1]$ and $\sqrt{\mathbb{E}[y^2]} \leq 1$, whether $|\mu| \leq c\varepsilon$ or $\varepsilon \leq |\mu| \leq 2\varepsilon$ in $O(1/\varepsilon)$ queries. (note that we can choose $c \in (0,1)$), note that in **Theorem 1.1**, we let $c = 1/2$).

Due to the similarity between Grover's Algorithm and the algorithm from Kothari & O'Donnell (2022), if in the future Grover's Algorithm can be run, then modifications can be made such that mean estimation could also be done efficiently without requiring multiple different algorithms for different cases.

4 APPLICATIONS

The speed-up can be used in many classical problems to reduce the computational complexity of existing problems that already uses Monte Carlo method.

4.1 Pricing of Financial Derivatives and Evaluating Financial Risk

4.1.1 What are derivatives?

Derivatives are a type of contract whereby their values are derived from one or more underlying assets, such as the price of a stock or currencies. The goal of derivative pricing is to determine the value of entering a contract given the uncertainty in the payoff due to fluctuation in the price of the underlying asset.

The contract is issued between an issuer and a holder for a predefined period of time (the time when the derivative is no longer in effect/needs to be redeemed is known as the expiration date tho whether a payoff occurs is dependent on the type of derivative), where a payoff is defined as what the holder stands to gain from purchasing the derivative. An example of this is the European Call option, which gives the holder the option to buy a specific asset, such as a stock, upon expiration or maturity at time T at a specific price (known as the strike price K). If the asset price at time T (which will be denoted as S_T) is above the strike price, the holder may purchase the asset at the strike price and gain the difference between the strike and the actual price of the asset. In the case that the asset's price is below the strike price, the holder will instead choose not to buy at the strike price (as they would incur a loss). In other words, the payoff is:

$$f(S_T) = \max\{0, S_T - K\}$$

To determine the expected present value of a call option, we need to get the value of (note that e^{-rT} represent the discount factor to account for the time value of money):

$$\mathbb{E}[e^{-rT} f(S_T)]$$

One common way of pricing asset is via what is known as the Black-Scholes-Merton model Black & Scholes (1973) which utilises stochastic differential equations as shown below:

$$\frac{dS(t)}{S(t)} = rdt + \sigma dW(t)$$

where W is a standard Brownian Motion, r is the mean rate of return and σ is the volatility of the stock price. The solution of this stochastic differential equation is:

$$S(t) = S(0)\exp((r - \frac{1}{2}\sigma^2)t + \sigma W(t))$$

Using this, the different prices could be sampled and an estimate of the derivatives price can be obtained using Monte Carlo method.

As we can see, the quadratic speed-up provided can be used to price simple derivatives. However, is there a better use case? The answer is yes! First, lets take a look at path-dependence.

A *path* refers to the different prices that an underlying asset undertakes over its lifetime. An European Call option is what is known as path-independent, meaning only the final price of the underlying affects the value of the derivative (the path or the different prices that the asset takes before expiration do not affect its values). These derivatives are typically easier to price using analytical methods (for example, Binomial or Trinomial Tree Method).

However, path-dependent derivatives' payoff is dependent on the price of the underlying asset for some or part of its life. As a result, they are more difficult to price.

4.1.2 Monte Carlo Method used in pricing Path-Dependent Derivatives

Some path-dependent derivatives are computationally difficult to price and as a result, could benefit from the speed-up facilitated by a quantum computer, as discussed in Chakrabarti et al. (2020). The value of a derivative is determined at the present time ($t = 0$) by finding the expected value of discounted payoff at the expiration date (or when the derivative is no longer in effect, which could be earlier than the expiration date for some exotic options). Note that the payoff must be discounted to account for the opportunity cost of investing in a risk-free asset.

A description of how a quantum algorithm could be used to determine the price of a derivative is in Chakrabarti et al. (2020) using a quantum algorithm in Montanaro (2015), which has since been improved upon in our new algorithm in Kothari & O'Donnell (2022). To briefly summarise, let $\omega \in \Omega$ as the discrete space of the paths and $f(\omega)$ be the discounted payoff given that path ω was followed. The algorithm will then take a sample of the paths to compute the expectation:

$$\mathbb{E}(f) = \sum_{\omega \in \Omega} p(\omega) f(\omega)$$

Since computing all the paths is computationally too expensive, we will instead use an estimator:

$$\mathbb{E}(f) \approx \frac{1}{M} \sum_{\omega=1}^{M} f(\omega)$$

By generating a large sample of size M, we can utilise the algorithm to provide a more accurate estimate of a value of a derivative. In classical computing, the estimator's error is $O(1/\sqrt{M})$, where M is the number of paths sampled. Suppose we had the "source code" and quantum access, this error can be reduced to $O(1/M)$, significantly decreasing the number of samples required to get an accurate representation of the value of a derivative.

A brief summary of common path-dependent derivatives has been included in the appendix (6.4).

4.1.3 Monte Carlo Method used in determining Value-At-Risk (VaR) and Conditional Value-At-Risk (CVaR)

Value-At-Risk (VaR) is a statistic used to measure the extent of financial loss/ measure a portfolio's (collection of assets) exposure to risk over a specific time frame (known as a time-horizon Δt). VaR

refers to the maximum possible loss in a given period of time that has a probability of occurring that exceeds a certain probability. For example, a one-day 95% VaR of 1 million refers to the probability of incurring a loss greater than 1 million in a given day is 5%. In addition, there is what is known as Condition Value-At-Risk (CVaR), which is defined as the expected loss for losses greater than VaR. CVaR can be difficult to estimate as it is sensitive to extreme losses (in other words, a small chance to lose a lot of money can disproportionately affect the value of CVaR). The relationship of the two can be summarised in the following formula for a loss x and probability of loss $p(x)$:

$$CVaR = \frac{1}{1-c}\int_{-1}^{VaR} xp(x)dx$$

c refers to a cut-off point that depends on VaR.

Monte Carlo method is one of the ways that can be used to determine VaR and has the advantage of making minimal assumptions about the underlying dynamics that affect the price of the portfolio (for example, one of the common methods Variance-Covariance Method assumes that the portfolio's value shifts linearly or quadratically by the change in market risk factors (i.e. the variance in value caused by fluctuations in the markets interest rates, currency, commodity price and more), which may cause inaccuracies as stated in Glasserman et al. (2000)).

The classical method of using Monte Carlo is as follows (with more reasoning in Glasserman et al. (2000))

(i) Generate N different samples of portfolios in Δt.
(ii) Re-evaluate the value of the portfolio at the end of the period.
(iii) Calculate the fraction of the samples where the loss in value of the portfolio exceeds a certain amount x.
(iv) Repeat step (iii) for different values of x.

Since the probability can be interpreted as a Bernoulli random variable, we can utilise the quantum algorithm (3.3) to get an estimate $\hat{\mu}$ that has tight bounds in $O(n)$ many queries (shown below) with a high probability:

$$|\hat{\mu} - \mu| \leq \sigma/n$$

By computing VaR, by extension, we can calculate CVaR.

4.2 Property Testing Algorithms

4.2.1 Preliminaries

Property Testing Algorithms are a type of algorithm made to infer some global feature of an object usually from a subset of data by having some sort of query access to the data set (GoldReich (2017)). These algorithms tend to be really efficient by providing an estimate that is close to the exact answer (with a higher probability), leading to a much faster run time/input size required (such as going from polynomial time with an exact answer to a sub-linear time for an approximate answer). For example, if the goal is to find the average age of a population, finding out everyone's age is too costly to be practical. Instead, taking a sample and providing an estimate is more efficient.

Property Testing algorithms are commonly utilised in fields where the input is large, where sometimes a loss of accuracy is a natural consequence of difficult and large problems. As a result, it is preferable to use Property Testing Algorithms instead of seeking an exact answer. Examples of use cases span Computational Biology, Astronomy, the study of the Internet, and NP-hard properties in Theoretical Computer Science (TCS). Property Testing Algorithms may also be used in cases where efficiency is prioritised over the accuracy or to save time computing some final decision by pruning unneeded outcomes.

Property Testing algorithms utilise *Proximity-Oblivious Testers (POTs)* in order to verify properties. Let:

- Set of functions defined over $\{1,...,n\}$ be Π_n
- Set of functions $\Pi = \cup_{n\in\mathbf{N}}\Pi_n$
- Proximity Parameter be $\varepsilon \in (0,1]$ (whereby the proximity parameter will let us determine whether an object has a property or is at least $\varepsilon-$ far from having said property).
- Threshold Frequency be $\tau \in (0,1]$.
- Detection Probability be $\rho : (0,1] \to (0,1]$ be a monotonically non-decreasing function of the proximity parameter.

Let a POT be a probabilistic oracle machine for Π, denoted as T, which will satisfy the following:

(i) T will accept inputs $f \in \Pi$ with probability at least τ. $\Pr[T^f(n) = 1] \geq \tau$
  - If $\tau = 1$, then we say T has one-sided error, otherwise it is two-sided.

(ii) T will reject inputs $f \notin \Pi$ with probability at least $1 - \tau + \rho(\varepsilon)$

For the remainder of this section, we will refer to $\rho(\varepsilon)$ as ρ.

Suppose that we have a POT with query complexity q (to determine each input f). What will be the total complexity of our tester q' such that we have a high probability of knowing whether our object has a property or is $\varepsilon-$ far from having said property? There are two cases:

- **Case 1**: $\tau = 1$ (i.e. one-sided error)
- **Case 2**: $\tau \in (0,1)$ (i.e. two-sided error)

In **Case 1**, we can simplify the problem down to determining whether the output of T follows a Bernoulli random variable with mean 0 or at least ρ. Let's denote the output of T as $X \sim Bern(\mu)$, where X is the probability of being rejected by T. If $\mu = 0$, this means that the probability of any query being rejected is zero, which makes sense since a mean of zero suggests that the object has the property. Otherwise, if $\mu = \rho$, we expect the first query to be rejected to be $1/\rho$-th (by considering a geometric distribution). As a result, in **Case 1**, we should only accept that object has the property we want if every query returns true for $O(1/\rho)$ queries.

In **Case 2**, it is more difficult than in Case 1 as even if the object has said property, there is a probability of being rejected. Similar to Case 1, we let the output of T be a Bernoulli random variable with mean either τ or $\tau - \rho$, where X is the probability of being accepted by T. Classically, according to Cheyshev's Inequality:

$$\Pr[|\hat{\mu} - \mu| \leq \sqrt{3}\sigma/\sqrt{n}] \leq 1/3$$

Meaning if we want to differentiate between the two different outcomes, we need the error to be lower than the gap between the two means ρ. In other words,

$$\sqrt{3}\sigma/\sqrt{n} \leq \rho$$

For the inequality above to be true (noting that the variance of a Bernoulli random variable is capped 6.5),

$$\begin{aligned}\sqrt{n} &\geq \sqrt{3}\sigma/\rho \\ n &\geq 3\sigma^2/\rho^2 \\ &\geq 3/2 \times \rho^2\end{aligned}$$

This means that to differentiate between $p = \tau$ or $p = \tau - \rho$, we need $O(1/\rho^2)$ many queries.

The question now follows, can the results above be improved? The answer is yes!

4.2.2 *How can Property Testing Algorithms be improved with Quantum Access?*

In **Case 1**, we were able to determine that the number of queries require is $O(1/\rho)$. According to Theorem 3 in Brassard et al. (1998), an efficient quantum algorithm can determine for mean for a Bernoulli random variable $X \sim Bern(\rho)$ in $O(1/\sqrt{\rho})$ many queries instead of $O(1/\rho)$ many.

In **Case 2**, according to Theorem 1.1 from Kothari & O'Donnell (2022), an efficient quantum algorithm with access to the "source code" can determine the mean with an error with σ/n with high probability. Let $\sigma/n = \rho$, hence we can see the number of queries required is $O(1/\rho)$ (which is an improvement from $O(1/\rho^2)$).

Using the improvements, specified earlier, we can expect that the improvements can lead to other existing problems being solved more efficiently. Some problems that are solved using POT include (but are not limited to):

- Boolean Functions (i.e. functions that output true or false)
- Real Functions
- Graph Properties (e.g. whether the graph is bipartite or not)

One example that showcases the improvement is in checking whether a n-bit string x is sorted. Lets consider a n-bit string to be sorted if for any bit x_i, $x_i \leq x_{i+1}$ for $i \in [0, n-1)$. We can see clearly that this task grows as the size of the string $n = |x|$ increases. To combat this, can we use a 2-Query POT that can solve this problem in sub-linear time? According to Proposition 1.5 in GoldReich (2017), a good randomized algorithm can determine where the string is sorted or $\varepsilon - far$ from being sorted in $O(1/\varepsilon)$-time. In the following paragraph, we will prove that is the case if we utilise our quantum algorithm.

Let our arbitrary string x be a $\delta_S(x)$-far from being sorted. Let x' be the sorted version of x, where (note $wt(x)$ means the Hamming Weight 6.6):

$$x' = 0^{n-wt(x)}1^{wt(x)} = \underbrace{0\ldots0}_{n-wt(x)}\ \underbrace{1\ldots1}_{wt(x)}$$

Let D_0 be the set of places in the first $n - wt(x)$ bits that is 1 (i.e. positions where x' and x differ) and let D_1 be the number of places in the last $wt(x)$ bits that is 0. Note that $|D_0| = |D_1|$. From here, we known that the distance $\delta_S(x) \leq (|D_0| + |D_1|)/2$, we can rearrange the following formula to get:

$$|D_i| \geq n/2 \times \delta_S(x)$$

for $i \in \{0, 1\}$.

If we choose any pair of $i, j \in [0, n)$ such that $i < j$ (note there are $\binom{n}{2}$ many such pairs),

$$\begin{aligned}\Pr(x_i > x_j) &\geq \frac{|D_0| \times |D_1|}{\binom{n}{2}} \\ &= \delta_S(x) * (n/2)^2/(n(n-1)/2) \\ &\approx \delta_S(x)^2/2\end{aligned}$$

Let $\rho(\varepsilon) = 0.5 \times \delta_S(x)^2$.

Therefore, we have provided that this algorithm has a quadratic detection probability, or $\rho(\varepsilon) = \varepsilon^2$ According to classical computing, we would need $O(1/\rho(\varepsilon)) = O(1/\varepsilon^2)$ for many queries. From earlier, we can solve this problem with a quantum computer in only $O(1/\sqrt{\rho(\varepsilon)}) = O(1/\varepsilon)$ many queries. We know this is optimal as according to proposition 1.5 in GoldReich (2017), the optimal solution to this problem requires $O(1/\varepsilon)$ queries for a randomized algorithm. This means that our quantum algorithm can at a minimum be as good as the best classical computing algorithm.

4.3 Determining Graph Parameters: Total Number of Edges

Graphs are a common data structure within computer science due to their ability to model many real-world relationships depending on what the vertices and edge weights of graphs represent. In very large graphs, a common way of representing edges is by using an adjacency list, where each vertex has a list of vertices. Suppose now we want to check how many edges are within a large graph. Can we do this in less than $O(|V|)$ or $O(|E|$ time?

4.3.1 *Preliminaries*

Suppose that we have a graph G with a set of vertices V (where $V = \{1, \ldots, n\}$, $n = |V|$ will therefore denote the total number of vertices within the graph) and a set of edges E (let $|E| = m$ denotes the total number of edges) so that we can define a graph as pair $G = (V, E)$. Two vertices u and v are said to be adjacent if there is an edge $e = \{u, v\}$ between them and $\{u, v\} \in E$.

Let degree be denoted by d_v, the degree is the number of edges connected to vertex v. We define $\prec$ to denote total order on V such that $u \prec v$ if $d_u < d_v$ or when $d_u = d_v$, $u < v$.

Suppose that we have a graph that allows us to perform a check with a constant query complexity $O(1)$:

- where two vertices $u, v \in V$ are adjacent or each other;
- the degree d_v for any $v \in V$.

The naive solution to get all the edges would be simply to check each vertex, which ends up with an $O(n)$ solution.
We can speed this up by providing an estimate with some proximity parameter/error ε. According to Hamoudi (2021) and Seshadhri (2015), we can do this in $O(\frac{n}{\varepsilon\sqrt{m}})$ time by utilising the algorithm below:

(i) Choose any arbitrary vertex $v \in V$ and any vertex $w \in V$ that is a neighbour of v (i.e. $\{u, v\} \in E$;
(ii) If $v \prec w$, output nd_v, otherwise repeat algorithm.

This algorithm leads to a sample of outputs whose mean is equal to the number of edges m in the graph. This means by estimating the mean of the samples, we can provide an estimate for the number of edges within a graph.

4.3.2 *How can Graph Parameters be determined whilst utilising a quantum speed-up?*

According to Hamoudi (2021), by utilising a quantum algorithm that estimates the mean and limiting $\varepsilon, \delta \in (0, 1/2)$, we can deduce an

estimate $\hat{m}$:

$$Pr(|\hat{m}-m| \leq \varepsilon m) \geq 1-\delta$$

with a total query complexity of $O^*(\frac{\sqrt{n}}{m^{1/4}}$. (Note that $O^*(\cdot)$ is a non-standard notation that ignores the log factors). According to Table 1 in Kothari & O'Donnell (2022), the quantum algorithm used in Hamoudi (2021) has a complexity of $O^*(n) = O(n\log^{3/2} n \log\log n)$ while the algorithm in Kothari & O'Donnell (2022) has a total query complexity of $O(n)$, leading to a speed-up by removing the log factors.

4.4 Determining Probability Distribution Given Dual Access Model

Suppose that we have a set of data where we have done some pre-computation already, specifically the probability mass function such that we have the ability to query for the probability for any specific outcome. Now, suppose we want to determine what type of distribution the data follows with a minimal number of queries/samples. How can we do this efficiently?

4.4.1 Preliminaries

The model I described above where I can obtain the probability mass function $D(\cdot)$ that is defined over $[n] = \{1,\dots,n\}$ is what is known as a **Dual Access Model**. In a Dual Access Model, we have dual oracles, which can perform the following actions:

(i) SAMP_D: Can return an element $i \in [n]$ with probability $D(i)$;
(ii) EVAL_D: Return the probability $D(j)$ for any element $j \in [n]$.

Now, we want to test the distribution D for uniformity (i.e. is it a uniform distribution over $[n]$) and identity (i.e. if $D = D^*$ for some known distribution D^*. For this case, we will look at the test for uniformity (note that the results can then be extrapolated to test for identity as well).

To measure the similarity between two distributions, we will use **total variation distance**, which for two distributions D_1 and D_2 will be defined as:

$$d_{TV}(D_1,D_2) = max_{S\subseteq[n]}(D_1(S)-D_2(S)) = \frac{1}{2}\sum_{i\in[n]}|D_1(i)-D_2(i)|$$

Suppose we have a Dual Access Model and a specific algorithm with the following inputs:

- Distribution on $n = \{1,\dots,n\}$;
- $\varepsilon_1, \varepsilon_2$ such that $0 \leq \varepsilon_1 < \varepsilon_2 \leq 1$.

This algorithm will make m calls to the oracles and either:

- if $d_{TV}(D_1,D_2) \leq \varepsilon_1$ (i.e. if the distributions are "close"), the algorithm will accept with probability at least 2/3;
- if $d_{TV}(D_1,D_2) \geq \varepsilon_2$ (i.e. if the distribution is "far"), the algorithm will reject with probability at least 2/3.

According to Canonne & Rubinfeld (2014), this can be done in $m = \Theta(\frac{1}{\gamma^2})$ many queries to the Dual Access Model (provide the queries take $O(1)$ time). See appendix section 6.7 for a more detailed proof. Let D be some unknown distribution where we have a dual oracle for and $\mathscr{U}$ be a uniform distribution on $[n]$. For each query, we get sample s_i and use it to compute X_i where:

$$X_i = (1-\frac{1}{nD(s_i)})\mathbf{1}_{\{D(s_i)>\frac{1}{n}\}}$$

Note $\mathbf{1}_{\text{boolean function}}$ is 1 when the boolean function is true, and 0 otherwise.

From here, we will define $\hat{d} = \frac{1}{m}\sum_{i=1}^{m} X_i$, where $\mathbf{E}[\hat{d}] = d_{TV}(D,\mathscr{U})$ and $\mathbf{V}(\hat{d}) \leq 1/\sqrt{m}d_{TV}(D,\mathscr{U})$. The upper bound of variance can be determined by noting that $X_i \in [0,1]$.

$$\begin{aligned}\mathbf{V}(\hat{d}) &= 1/m^2 \times \mathbf{V}(X_1+\cdots+X_m)\\ &= 1/m^2 \times m \times \mathbf{V}(X_i)\\ &\leq 1/m \times \mathbf{E}(X_i^2)\\ &\leq 1/m \times \mathbf{E}(X_i)^2\\ &= 1/m \times d_{TV}(D,\mathscr{U})^2.\end{aligned}$$

Using $m = \Theta(\frac{1}{\gamma^2})$ many queries, we can guarantee that:

$$\Pr[|\hat{d}-d_{TV}| \geq \gamma] \leq 1/3$$

Lets see how our quantum algorithm can improve this.

4.4.2 Algorithm Speed-up with Quantum Access

According to Theorem 1.1 in Kothari & O'Donnell (2022), we know that:

$$\Pr\left[\left|\hat{\mu}-\mu\right| > \sigma/n\right] \leq 1/3$$

We will make the following substitutions:

- $\hat{\mu} = \hat{d}$;
- $\mu = d_{TV}(D,\mathscr{U})$;
- $\sigma = 1/\sqrt{m} \times d_{TV}(D,\mathscr{U})$;
- $n = m$.

There we can conclude that:

$$\Pr\left[\left|\hat{d}-d_{TV}(D,\mathscr{U})\right| > d_{TV}(D,\mathscr{U})/m^{3/2}\right] \leq 1/3$$

To get the same error bound, let:

$$\begin{aligned}d_{TV}(D,\mathscr{U})/m^{3/2} &= \gamma\\ m^{3/2} &= d_{TV}(D,\mathscr{U})/\gamma\end{aligned}$$

Therefore $m = O(1/\gamma^{2/3}) < O(1/\gamma^2)$. Hence by having quantum access and "source code", we can provide more than a quadratic speed-up to testing a probability distribution for uniformity and identity.

5 CONCLUSION

To summarise, in this paper we were able to accomplish:

- List what I did during this time

5.1 Future Work

Plenty of applications of quantum algorithms in classical problems could lead to a speed-up. These include:

- [ToDo: add something here]

In our research, we attempted to look at multiple potential applications (which I will list below). Although there is not an obvious link, this topic should be explored further.

5.1.1 Randomized Algorithms

Randomized algorithms are a type of algorithm that incorporates some degree of randomness as a part of their logic. A good question is why not just use a deterministic algorithm? While it is common for deterministic and randomized algorithms to have similar worst-case performances, the hope is that on average, the randomised algorithms will outperform deterministic algorithms for specific problems. An example of an algorithm with both a deterministic and randomized version is the commonly used *Quick Sort*.

Quick Sort is an example of a divide-and-conquer algorithm where the algorithm can sort a list of objects in $O(n\log(n))$ time. The sorting algorithm chooses an example as the pivot and partitions the list into two separate lists (where all elements in one list are less than the pivot and the rest are placed in the second list) and sorts them recursively again using the same algorithm. There are varying methodologies for choosing what the pivot is as it has a large impact on the performance of the algorithm (for instance, if the largest/smallest element was chosen, the time complexity degrades to $O(n^2)$). Deterministic ways of choosing the pivot include attempting to determine a median value. These methods may add additional costs to quick sorting (for instance, determining the median is very costly) and may perform adversely on specific types of data sets (such as if the data set was already sorted). Randomized quick sort differs as the pivot is chosen on random (or rather, using the computer pseudo-random number generator), which sometimes leads to a better-expected performance overall data sets. To further clarify, randomized quick sort's expected run time remains consistent across inputs while deterministic quick sort does not.

While quick sort is just one example, there are other randomized algorithms that are more efficient than deterministic ones and may even be simpler to implement compared to their deterministic counterparts.

5.1.2 Data Stream Algorithms

Data Stream Algorithms refer to a specific type of algorithm that is able to take in a large number of inputs (from an input stream σ) and use a small amount of space (ideally in sub-linear/logarithmic space) to process the data (Chakrabarti (2020)). Examples of how these algorithms can be used are:

- detecting denial-of-services attacks (by determining two different IPs where one IP sends significantly more than another IP)
- analysing a large data set without all of the data being loaded into RAM to determine the most frequent items, distinct items and other statistical problems

Typically, these algorithms will compute an estimate or an approximation $\mathscr{A}(\sigma)$ (output of a randomized streaming algorithm $\mathscr{A}$ given an input of σ) of the true value $\phi(\sigma)$. We say that the algorithm (ε,δ)-estimates ϕ if

$$Pr\left[\left|\frac{\mathscr{A}(\sigma)}{\phi(\sigma)}-1\right|\geq\varepsilon\right]\leq\delta$$

Note that if $\varepsilon=0$, then the algorithm is exact (will compute the actual answer at the expense of space and time). If $\delta=0$, then the algorithm is deterministic.

ACKNOWLEDGEMENT

[ToDo: include thank you] Supervisor + SQA

6 APPENDIX

In the appendix, we will go through any tricks that I found useful or any topic that I found interesting that may have some or little to no tie to the actual paper.

6.1 Challenges for Quantum Computing

For our algorithm to be implemented, there are several challenges that must be first overcome. In particular, the creation of error-free quantum hardware with sufficient capacity to allow for large computations.

6.1.1 Brief History

Quantum Computing is a relatively new field of study, being hypothesised by a series of respected physicists such as Richard Feynman (Feynman (1982)), and Yuri Manin around and before the 1980s. Although it was theorised that quantum computing would be beneficial in simulating real-world systems that are fundamentally random/governed by quantum mechanics, interest in the topic quickly rose when Peter Shor developed Shor's Algorithm (Shor (1997)), which provide an efficient way of factoring large integers (let N be the integer, the problem went from having an $O(N)$ time solutions to a $O(log(N))$ solution). This was significant as some of the most prominent cryptographic techniques relied upon large prime integers to be difficult to factor (examples include the RSA (Rivest et al. (1978))). In 1996, Lov Grover invented an algorithm (Grover (1996)) that searched a database for specific items in $O(\sqrt{N})$ time instead of $O(N)$ time (where N is the size of the database). Although the speed-up isn't as significant as in Shor's Algorithms case, Grover's Algorithm is far more versatile, as shown by our Quantum Monte Carlo method being created by modifying Grover's Algorithm. While there were algorithms discovered earlier, none of the preceding algorithms had the same impact as Shor and Grover's algorithm.

Nowadays, it has been recognised that quantum computers may be used to solve a multitude of problems. An overview is given in Hamoudi (2021), where the author describe a number of applications that include but are not limited to:

- Mean Estimation (determining an accurate estimate for the mean with less samples or with a limited, potentially random stopping time);
- Estimating Graph Parameters (e.g. number of edges in an adjacent list model of a graph and number of triangles in a graph);
- Optimization Problems (producing independent samples from a probability distribution);
- Big Data Problems (reducing the amount of memory required to reach a usable result with a reasonable time-space trade off).

6.1.2 Ongoing Challenges

In 2022, the largest quantum computer is made by IBM, called "IBM Osprey", with a staggering 433 qubits (this is almost triple the number of qubits IBM was able to create in the previous year).

However, there are several engineering problems that must be solved for quantum computers, specifically the hardware required to build one.

Firstly, more qubits are required to have reliable, error-free quantum computers that can store a large number of information. This is a very active region of research, with IBM making an ambitious claim to achieve over 4000 qubits by 2026 and other firms having similar goals.

Secondly, Quantum Decoherence refers to the phenomenon where objects lose their quantum mechanical nature, effectively eliminating any advantage of using a quantum computer. This could be resolved partially by having more qubits and using error-correction algorithms to improve redundancy, but this leads to less computing power and increased run time. This additional run-time may be enough to offset the potential savings gained by using a quantum algorithm. In addition, decoherence leads to a hard limit on how long an algorithm can run, limiting the size of inputs when one of quantum computing's primary benefits is its ability to process big data more efficiently than classical computers theoretically. This has spurred research into developing algorithms for Noisy Intermediate Scale Quantum (NISQ) Processors that can develop passable results despite coherence.

6.2 Beyond Big Oh Notation

While Big Oh Notation is the most common, there are several other ways we can describe the time or space complexity.

There is also known as amortized time, which is different from average case. If an algorithm is ran multiple times and has an amortized time of $O(1)$, that means over a sufficient number of runs (but not necessarily in a small number of runs), the average time is $O(1)$. A good example of this is allocating more space to store a variable. For example, a data structure may allocate space for n objects and doubles it storage when it tries to store more than its current capacity. For the first n insertions, the time taken to insert is $O(1)$ but when tries to insert the $n+1$-th element, it doubles its capacity, which is an $O(n)$ operation. Despite the expensive act of increasing its capacity, on average the insertion is extremely fast.

6.3 The Median Trick

As previously mentioned, the median trick can improve the accuracy of a quantum algorithm. Suppose an algorithm can provide an estimation for X that is within $\pm\varepsilon$ with a probability greater than 90%. By running the algorithm $k = O(\log(\frac{1}{\delta})$ times and taking the median of the k outputs, the probability for estimate for X to be within $\pm\varepsilon$ becomes $1-\delta$. This can be proven using Chernoff Bounds (Andoni (2015)).

6.4 Examples of Path-Dependent Derivatives

Path-dependent derivative's price are dependent on parts or all of the prices an underlying asset(s) take for a specific period of time. Path-dependent derivatives tend to usually have some knock-out condition (meaning their payoff will depend on when the derivative terminates) (e.g. Auto-callable Options) or require their entire price history to evaluate (e.g. Asian Options). As a result, path-dependent derivatives may benefit more than path-independent derivatives when it comes to a speed-up via Quantum Monte Carlo since they are more commonly priced using numerical methods (especially Auto-Callables and TARFs).

An **Asian Option** has the payoff be the difference between the average (arithmetic) value of an asset $\bar{S}$ and the strike price K:

$$Payoff = \max\{0, \bar{S} - K\}$$

Geometric Average Options are similar in concept to an Asian option, with the average stock price being the Geometric Average of the stock price instead.

Barrier Options are similar to regular options but will be knocked out if the underlying asset crosses a pre-determined threshold.

An **Auto-callable Option** is a set of options that provide different payouts at different dates. If any of the payments are taken, it voids all future payoffs.

A **Target Accural Redemption Forward (TARF)** is any derivative whose payoff is capped at a specified amount. If the payout of a specific TARF exceeds a certain, pre-determined amount, the derivative is knocked off (meaning the maximum payout is capped).

6.5 Upper Limit for Variance of Bernoulli Random Variable

Let $X \sim Bern(p)$ where $p \in [0,1]$. This means that:

$$\mathbb{E}[X] = p = \mu$$
$$\mathbb{V}[X] = p(1-p) = \sigma^2$$

If we differentiate variance with respect to p,

$$\frac{d\sigma^2}{dp} = 1 - 2p$$
$$\frac{d^2\sigma^2}{dp^2} = -2;$$

Since variance has a turning point at $p = 1/2$ and the second derivative is negative, this implies variance is maximized at $p = 1/2$.

$$\begin{aligned}\sigma^2 &= p(1-p)\\ &\leq 1/2 \times (1 - 1/2)\\ &= 1/4\end{aligned}$$

Hence we can conclude that the following identity for Bernoulli Random Variables:

$$\sigma^2 \leq 1/4$$

6.6 Hamming Weight

The Hamming Weight $wt(x)$ refers to the number of locations that hold the value one; that is:

$$wt(x) = |\{i \in [|x|] : x_i = 1\}| = \sum_{i=1}^{|x|} x_i$$

Hamming Weights are widely used in cryptography, information theory and other fields of computer science. For example, in one paper (Fatih Balli & Banik (2021)), it is shown that Hamming Weight has a high correlation with power consumption of cryptographic algorithms.

6.7 Math for Determining Probability Distribution

In this section, we will cover the proof of why the number of samples according to classical computing is $\Theta(1/\gamma^2) = \Theta(1/(\varepsilon_2 - \varepsilon_1)^2)$ as stated in section 4.4.

Suppose that we had m samples of a random variable $Y_1, \ldots, Y_m$ where

$Y_i \in [0,1]$ for any $i \in [1,m]$. Let $\mathbf{E}[Y_i] = p_i$ and $\sum_{i=1}^{m} p_i = P$ and let any $\gamma \in (0,1]$. By using Chernoff's Bounds, Canonne & Rubinfeld (2014) was able to deduce the following identities:

$$\Pr[\sum_{i=1}^{m} Y_i > P + \gamma m], \Pr[\sum_{i=1}^{m} Y_i < P - \gamma m] \leq \exp(-2\gamma^2 m)$$

We will now simply this his expression by letting $\sum_{i=1}^{m} Y_i/m = \hat{d}$ and $P/m = d_{TV}(D, \mathscr{U})$.

$$\begin{aligned} \Pr[\sum_{i=1}^{m} Y_i > P + \gamma m] &= \Pr[\frac{\sum_{i=1}^{m} Y_i}{m} > \frac{P}{m} + \gamma] \\ &= \Pr[\hat{d} - d_{TV}(D, \mathscr{U}) > \gamma] \\ \therefore \Pr[\hat{d} - d_{TV}(D, \mathscr{U}) > \gamma] &\leq \exp(-2\gamma^2 m) \end{aligned}$$

Using a similar argument, we can deduce that:

$$\Pr[\hat{d} - d_{TV}(D, \mathscr{U}) < \gamma] \leq exp(-2\gamma^2 m)$$

Hence we can conclude that:

$$\Pr[|\hat{d} - d_{TV}(D, \mathscr{U})| > \gamma] \leq exp(-2\gamma^2 m)$$

In this case, we want a high probability of the event to be true, we want to find a number of samples m such that the probability is less than 1/3.

$$\begin{aligned} \exp(-2\gamma^2 m) &= 1/3 \\ \exp(2\gamma^2 m) &= 3 \\ 2\gamma^2 m &= \ln 3 \\ m &= \frac{\ln 3}{2\gamma^2} \end{aligned}$$

Hence the number of samples required for a classical computer to test for uniformity is $m = \Theta(1/\gamma^2) = \Theta(1/(\varepsilon_2 - \varepsilon_1)^2)$.

REFERENCES

Andoni A., 2015, Algorithmic Techniques For Massive Data Lectures Notes, http://www.mit.edu/~andoni/F15_AlgoTechMassiveData/files/scribe2.pdf

Black F., Scholes M. S., 1973, Journal of Political Economy, 81, 637

Brassard G., HØyer P., Tapp A., 1998, in Larsen K. G., Skyum S., Winskel G., eds, Automata, Languages and Programming. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 820–831

Canonne C., Rubinfeld R., 2014, Testing probability distributions underlying aggregated data, doi:10.48550/ARXIV.1402.3835, https://arxiv.org/abs/1402.3835

Chakrabarti A., 2020, Data Stream Algorithms, https://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf

Chakrabarti S., Krishnakumar R., Mazzola G., Stamatopoulos N., Woerner S., Zeng W. J., 2020, arXiv e-prints, p. arXiv:2012.03819

Fatih Balli A. C., Banik S., 2021, Some Applications of Hamming Weight Correlation, https://eprint.iacr.org/2021/611.pdf

Feynman R. P., 1982, International Journal of Theoretical Physics, 21, 467

Glasserman P., Heidelberger P., Shahabuddin P., 2000, Master. Risk, 2

GoldReich O., 2017, Introduction To Property Testing, https://www.wisdom.weizmann.ac.il/~oded/PDF/pt-v3.pdf

Grover L. K., 1996, A fast quantum mechanical algorithm for database search, doi:10.48550/ARXIV.QUANT-PH/9605043, https://arxiv.org/abs/quant-ph/9605043

Hamoudi Y., 2021, PhD thesis, https://yassine-hamoudi.github.io/files/other/PhDthesis.pdf

Kothari R., O'Donnell R., 2022, arXiv e-prints, p. arXiv:2208.07544

Montanaro A., 2015, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 471, 20150301

Rivest R. L., Shamir A., Adleman L., 1978, Commun. ACM, 21, 120–126

Seshadhri C., 2015, CoRR, abs/1505.01927

Shor P. W., 1997, SIAM Journal on Computing, 26, 1484