Quantum Computing and Machine Learning:
A Survey

Austin Feydt, *Case Western Reserve University*

12 December 2017

Abstract

As technology advances, it seems that we are on track to see quantum computers become commercialized in the near future. But what exactly is a quantum computer, and how will it impact current classical computations, such as machine learning algorithms? In this survey, we take a look at the fundamentals of quantum computation, and focus in on Grover's algorithm for optimizing a search, one of the most well-studied quantum algorithms. We will see how it can potentially speed up some well-known algorithms, and what other quantum advancements can be made to increase efficiency in machine learning problems.

1 Introduction and Motivation

Classical computers, ones that we use every day, encode information bit by bit. Each bit can be exactly one of two possible values, either 0 or 1. Thus, an n-bit word in a classical computer is simply a string of *n* zeroes and ones. However, conceived after years of studying quantum physics, a quantum bit (qubit) is capable of so much more than a classical bit. We can think of a qubit as simply an atom in one of two different states, denoted as 0 or 1. Thus, two qubits can attain four well-defined states (00, 01, 10, or 11). So, how does this make qubits superior to classical bits? Well, it turns out that a qubit, just like an electron, can exist simultaneously as 0 and 1, with distinct probabilities for each of the states, denoted by a numerical coefficient. Thus, a qubit gives us 2 bits of information: it's state, and the probability of that state. And in general, n qubits together give us 2^n bits of information, an exponential increase compared to classical computers. And because of this superposition aspect, quantum computers can operate on all possible states simultaneously, allowing for much faster computations [8].

We now know that quantum computers offer much faster computations than classical computers, as they can operate on all possible states at the same time, rather than one at a time. What kinds of computations could benefit from a speed boost? In the realm of machine learning, at least, optimization plays a crucial role in a large number of modern machine learning algorithms. The idea of finding optimal parameters to minimize a predefined loss function, or to maximize probabilistic models is a common thread between many aspects of machine learning. Thus, it seems natural to attempt to use quantum computing to speed up these optimization processes. And as it turns out, there exists a well-defined and long-studied quantum algorithm that, for certain machine learning algorithms, offers a significant increase in efficiency (known as Grover's algorithm)[9][1]. A conceptual picture illustrating qubit and bit operations can be seen in Figure 1.

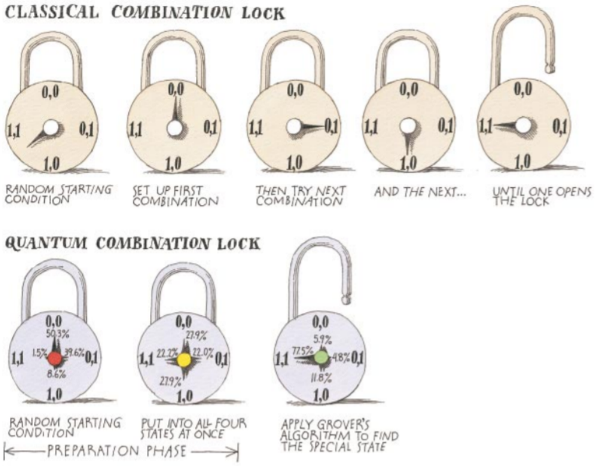

CLASSICAL COMBINATION LOCK
0,0
1,1
0,1
1,0
RANDOM STARTING CONDITION
SET UP FIRST COMBINATION
THEN TRY NEXT COMBINATION
AND THE NEXT...
UNTIL ONE OPENS THE LOCK
QUANTUM COMBINATION LOCK
50.3%
1.5%
39.6%
8.6%
27.9%
22.2%
22.0%
27.9%
5.9%
77.5%
4.8%
11.8%
RANDOM STARTING CONDITION
PUT INTO ALL FOUR STATES AT ONCE
APPLY GROVER'S ALGORITHM TO FIND THE SPECIAL STATE
PREPARATION PHASE

Figure 1: A comparison between two bits and two qubits

For many of the algorithms we consider later, we are given a set of data points, and then the algorithm must quantize this data and perform quantum calculations to output the desired classifier back in classical dimensions. As we will learn later, quantum algorithms output probabilistic answers, in which the quantum state returned is based on the probabilities of each state at the end of the algorithm. Thus, there is some fuzziness, which can lead to incorrect outputs. However, a complexity class defined specifically for quantum algorithms addresses this error, stating that a quantum classifier can make mistakes and still be considered efficient (but, more about that later...) As we will also soon learn, there are certainly **many** limitations to actually implementing these quantum machine learning algorithms, thus they are analyzed with as much theoretical rigour as is available for the area of quantum computing thus far[10].

2 Algorithms and Methods

2.1 Background on Quantum Computing

As already mentioned, the qubit in quantum computing is the quantum analogue of the bit in classical computing. However, while a bit can only either exist in the 0 on 1 state, a qubit exists in a **superposition** of both states. This is inspired by studying electrons in quantum physics (in which the particle can simultaneously exist in two different orbitals of the same atom.) Mathematically, we can describe a qubit using the famous Dirac notation. With this notation, we denote the state ”0” as ”spin-down,” which is $|0\rangle$ in Dirac notation. Likewise, we denote the state ”1” as ”spin-up,” which can be written as $|1\rangle$ in Dirac notation. Thus, we can express an arbitrary qubit as

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1}$$

In Equation 1, our particle is decomposed into a linear combination of both states, where α and β are complex numbers, known as the **amplitudes** of each fundamental state. We require that α and β exist such that $|\alpha|^2 + |\beta|^2 = 1$[1]. Thus, we can see exactly why qubits have an advantage over regular bits: a qubit gives us 2 bits of information (because we must supply not just the state, but also the probabilities of each state), whereas a bit simply gives 1 bit of information (”0” or ”1”) [14].

By converting our complex coefficients α and β, we can further express our qubit as

$$|\psi\rangle = cos(\frac{\theta}{2})\,|0\rangle + e^{i\phi}sin(\frac{\theta}{2})\,|1\rangle \tag{2}$$

which is a familiar wave equation with parameters $0 \leq \theta \leq \pi, \quad 0 \leq \phi \leq 2\pi$. These two constraints on θ and ϕ describe a point on a three dimensional sphere with radius 1, known as the Bloch sphere, as seen in Figure 2. The Bloch sphere gives a geometric explanation to qubit operations, which will be explored further later [14].

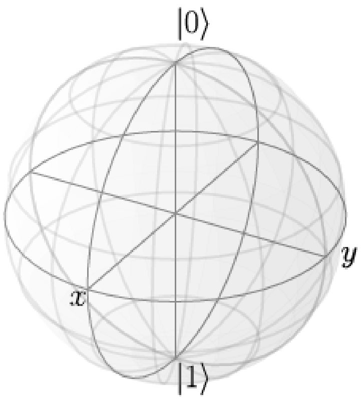


Figure 2: The Bloch sphere

When left unobserved, the $|\psi\rangle$ exists in a superposition of both fundamental states. However, as soon as the state is observed in any capacity, it immediately collapses into $|0\rangle$ with probability $|\alpha|^2$, or into $|1\rangle$ with probability $|\beta|^2$. This observation is known as **irreversible** because doing so erases all memory of the values of α and β. In other words, there is no way of recovering the actual state the particle was in prior to observing[1].

Observing a qubit is the only operation that is irreversible. All other possible operations in quantum mechanics are reversible, and thus ”safe” to perform on a qubit without fear of losing amplitudes. These reversible operations, known as **gates**, are analogous to logical gates in classical computations. And these gates can be combined to form quantum circuits, just like classical gates. However, it is important to note that, just like classical logic gates, quantum gates can either act on a single qubit, or on a collection of entangled qubits, also known as a **register**. For example, we may write a two-qubit register as

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle \tag{3}$$

where, similarly as before, $|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$. By generalizing to an n-qubit register, we can conclude that it will have 2^n amplitudes when left unobserved, which is exponentially more information than an n-bit register [6].

One such quantum gate, which is a fundamental gate used in more complicated algorithms, is the *Walsh-Hadamard* gate **H**, as seen in Figure 3. It simply maps $|0\rangle$ to $\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$ and maps $|1\rangle$ to $\frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle$. In simple terms, H maps a quantum state to a superposition of both fundamental states, with equal probability of both states. This gate acts as a crucial component to many more complicated quantum circuits, such as circuits used in Grover's Algorithm [1].

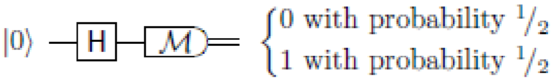


Figure 3: A quantum circuit consisting of the Walsh-Hadamard gate, and a measurement gate

2.2 Grover's Algorithm

Back in 1996, Lov Grover, a researcher at Bell Labs, proposed the following problem:

Imagine a phone directory containing N names arranged in completely random order. In order to find someone's phone number with a probability of $\frac{1}{2}$ *, any classical algorithm (whether deterministic or probabilistic) will need to look at a minimum of* $\frac{N}{2}$ *names. Quantum mechanical systems can be in a superposition of states and simultaneously examine multiple names. By properly adjusting the phases of various operations, successful computations reinforce each other while others interfere randomly. As a result, the desired phone number can be obtained in only* $O(\sqrt{N}$ *steps. The algorithm is within a small constant factor of the fastest possible quantum mechanical algorithm [9].*

Grover proposed an algorithm where, when given a Boolean function f as a black-box, such that there is some unique x_0 where $f(x_0) = 1$, we can find the unique value in sub-linear time using quantum superpositioning. And while this may seem impossible, it ends up being a rather simple and powerful algorithm.

To begin, Grover's algorithm uses Walsh-Hadamard gates on the all-zero state $|000\ldots0\rangle$ to create an equal superposition of all possible states, with amplitude value $\frac{1}{\sqrt{2^n}}$. Then, the algorithm continues by repeating the **Grover Iteration**. This iteration, in layman's terms, uses a series of special, nontrivial quantum transformations which make the desired x_0 exhibit a much higher amplitude than all the other states. Thus, whenever the algorithm stops and we observe our qubit register, we have a high probability of the register collapsing to the desired state.[3]

In order to guarantee a high amplitude on the desired state, the Grover Iteration must repeat roughly $\frac{\pi}{4}\sqrt{n}$ times. In more general cases, where there are t values of x such that $f(x) = 1$, rather than just 1 such value, the necessary number of iterations of the Grover Iteration shrinks to roughly $\frac{\pi}{4}\sqrt{\frac{n}{t}}$, with slight modifications in the original algorithm [1].

2.3 A Conceptual Look at Grover's Iteration

As we mentioned, the heart of Grover's algorithm is the repeated Grover Iteration, which performs transformations on the amplitudes of each state to increase the amplitude of the desired state. This is done in two separate steps, which each happen in each iteration of Grover's algorithm [10].

The first step of Grover's Iteration is to apply the black box function $f(x)$ via a quantum black box map

$$U_f : |x\rangle \to (-1)^{f(x)} |x\rangle \tag{4}$$

where x is our current qubit register. Conceptually, what this mapping is doing is flipping **only** the target state to the negative of it's current state, and leaving all other $2^n - 1$ states alone. This operation U_f is known as the phase shift, and is responsible for isolating the target state from all of the other states using the black box function. It is also important to note that this shift causes the average amplitude (which is originally $\frac{1}{\sqrt{2^n}}$) to slightly shift down, since the target amplitude is now negative[10].

Following the phase shift comes the second operator of the Grover Iteration. Recall that at the beginning of Grover's algorithm, all states are set to the same amplitude using the Walsh-Hadamard gate. Following the first U_f operator, this average is now slightly less than $\frac{1}{\sqrt{2^n}}$. Thus, the second operator, denoted as $U_{\psi^\perp}$, takes advantage of this inconsistency to the mean by what is known as an ”inversion about the mean,” in which each state's amplitude is transformed so that it is as far above the average amplitude as it was below the average amplitude prior to the transformation (and vice-versa for amplitudes originally above the average amplitude). Since the first operator drops the target amplitude below the average, this second operator will overcompensate by driving the target amplitude way above the average, while slightly shifting all other amplitudes below the average. After a necessary number of iterations, these two operators combined drive the target amplitude to a value very close to a probability 1, so that measuring the state at the end of Grover's algorithm almost guarantees that the qubit register will collapse into the desired state [9]. A visualization of Grover's algorithm, applied to a 3-qubit register, is outlined in Figure 4, which highlights what, conceptually, is going on with Grover's iteration.

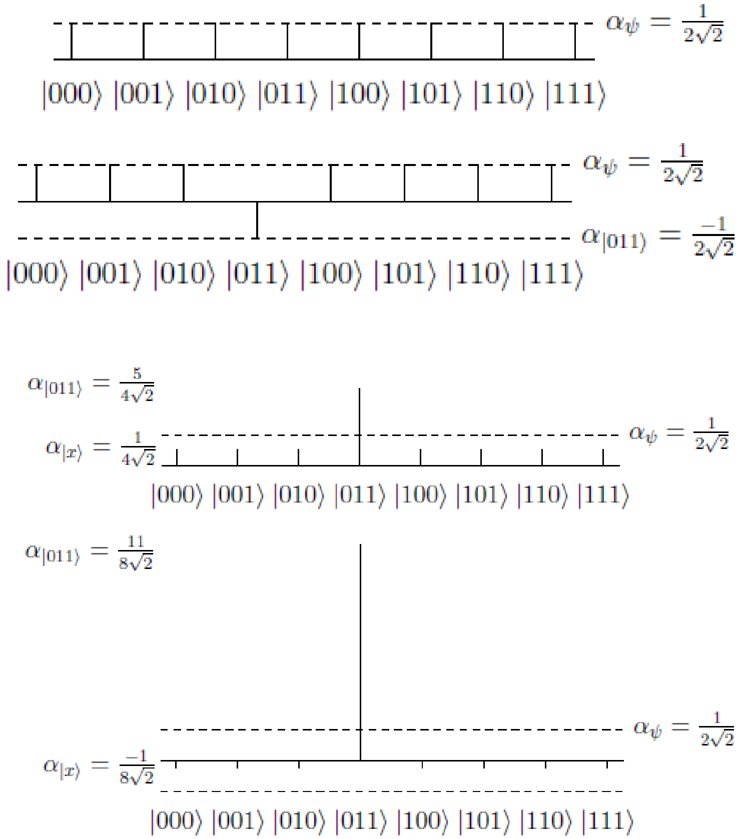


Figure 4: Top: Walsh-Hadamard gate applied to initialized data; Second: U_f applied to states (notice $|011\rangle$ is the desired state); Third: $U_{\psi^\perp}$ applied to above states; Bottom: State amplitudes after a second Grover iteration

2.4 Applications of Grover's Algorithm

2.4.1 Quantum Divisive Clustering

Clustering, a classic machine learning algorithm, involves categorizing data points within geometric clusters, in hopes of discovering interesting, meaningful clusters of data. We define our data set D_n comprised of n points, such that $D_n = \{x_1, \ldots, x_n\}$. Each example x_i has d attributes, so $x_i \in \mathbb{R}^d$. Thus, our goal is to partition our example set D_n into disjoint subsets, called **clusters**, in hopes that ”similar” points are placed in the same cluster, and ”not similar” points are placed in different clusters. Similarity is determined by the class labels of each example in the set. Clustering also relies on some notion of ”distance” between examples, which is dependent on the dimensions of the examples [1].

However, for quantum clustering, we discard this framework in favor of a black box model, in which the ”distance” formula is solely available through a black box function (similarly to the magical $f(x)$ function in Grover's algorithm). We will refer to this black box function as the ”distance oracle,” whose circuit can be seen in Figure 5. It is with this distance oracle that we are able to implement a modified version of Grover's algorithm to quickly find clusters [1].

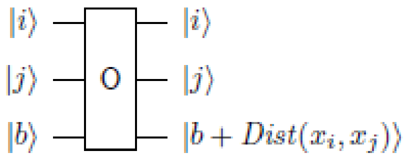
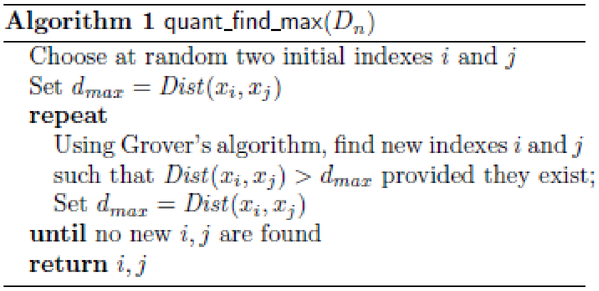


Figure 5: Illustration of the distance oracle: *i* and *j* represent indices of points in D_n, and $Dist(x_i, x_j)$ represents the black box distance between the points. Note that b is simply an external register set to 0 to compute *Dist* unitarily

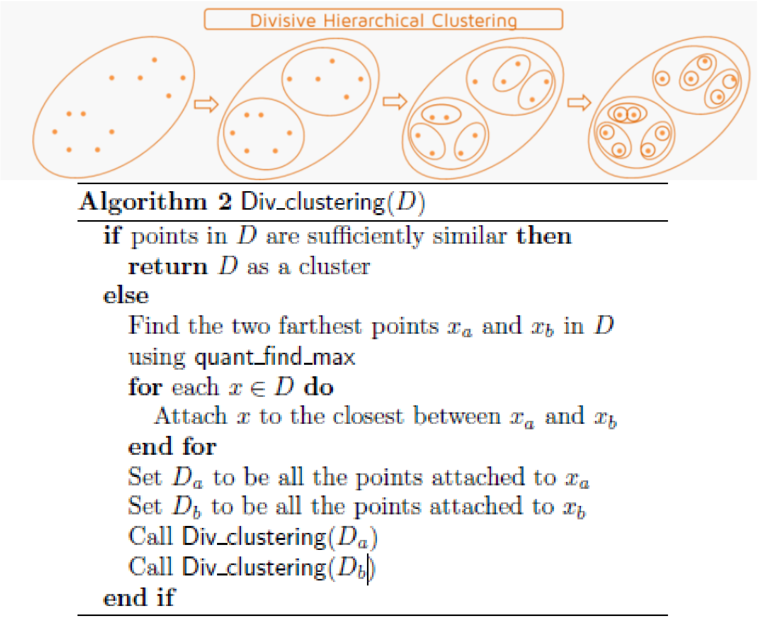
The foundation of quantum clustering is being able to quickly find points in D_n with the largest distance, as computed by the distance oracle. Thus, we can define a slick algorithm that poses this problem as a search problem, and uses Grover's algorithm to find these ideal points quickly (see Figure 6). In this algorithm, we start with a guess distance by sampling two points from D_n at random. It then utilizes Grover's algorithm to search for two points that have a larger distance than the current max distance. It repeats Grover's algorithm until it is unable to find a larger distance, and returns the maximum distance d_{max}. Classically an exhaustive search, the number of queries required for this quantum algorithm to converge has been show to be in the order of $O(n)$ [7].



```
Algorithm 1 quant_find_max($D_n$)
  Choose at random two initial indexes $i$ and $j$
  Set $d_{max} = Dist(x_i, x_j)$
  repeat
    Using Grover's algorithm, find new indexes $i$ and $j$
    such that $Dist(x_i, x_j) > d_{max}$ provided they exist;
    Set $d_{max} = Dist(x_i, x_j)$
  until no new $i, j$ are found
  return $i, j$
```

Figure 6: Utilizing Grover's algorithm to quickly find best points in clustering problem

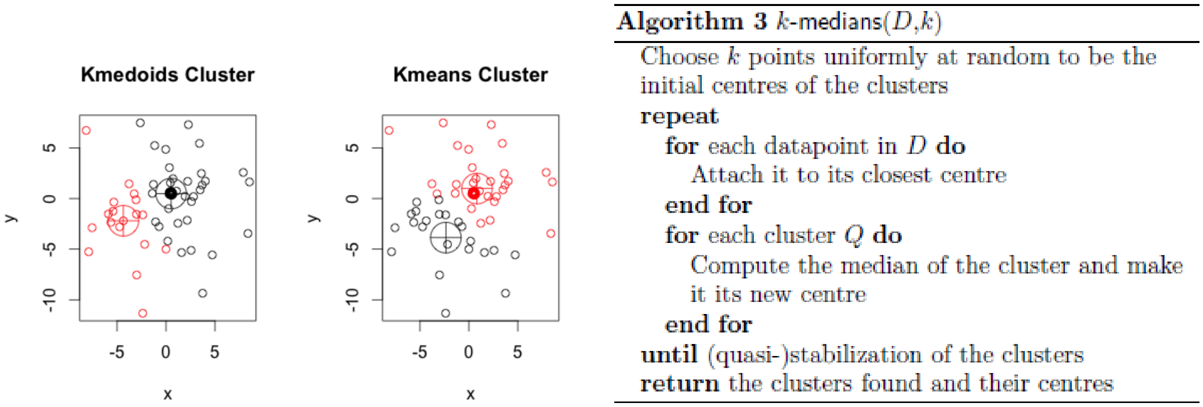
Divisive Clustering is known as one of the simplest ways to build a hierarchy of clusters. It is a recursive algorithm in which we partition our examples D_n into two clusters (based on the two points with maximum distance as the two centers), and continue partitioning each sub-cluster until all of the examples in the sub-cluster are "similar enough" (this may involve some form of error forgiveness if the classes are too inseparable, or by allowing clusters to consist of single points). See Figure 7 for a visual interpretation of Divisive Clustering and for the pseudocode[1]. We will analyze the classical and corresponding quantum run-times in **Theoretical Results**.



```
Algorithm 2 Div_clustering($D$)
  if points in $D$ are sufficiently similar then
    return $D$ as a cluster
  else
    Find the two farthest points $x_a$ and $x_b$ in $D$
    using quant_find_max
    for each $x \in D$ do
      Attach $x$ to the closest between $x_a$ and $x_b$
    end for
    Set $D_a$ to be all the points attached to $x_a$
    Set $D_b$ to be all the points attached to $x_b$
    Call Div_clustering($D_a$)
    Call Div_clustering($D_b$)
  end if
```

Figure 7: Top: A visual evolution of Divisive Clustering; Bottom: Pseudo-code for recursive Quantum Divisive Clustering

2.4.2 Quantum k-medians Clustering



```
Algorithm 3 k-medians($D$,$k$)
  Choose $k$ points uniformly at random to be the
  initial centres of the clusters
  repeat
    for each datapoint in $D$ do
      Attach it to its closest centre
    end for
    for each cluster $Q$ do
      Compute the median of the cluster and make
      it its new centre
    end for
  until (quasi-)stabilization of the clusters
  return the clusters found and their centres
```

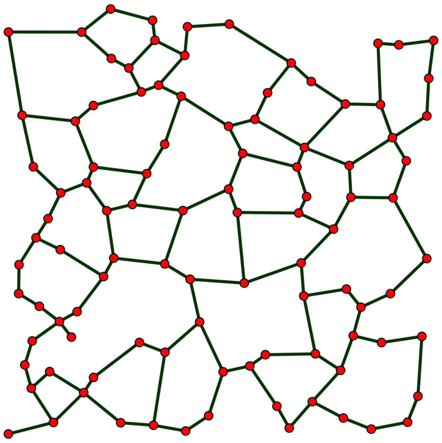
Figure 8: Left: A visual example of k-medians (k-mediods) and k-means; Right: Pseudo-code for iterative Quantum k-medians

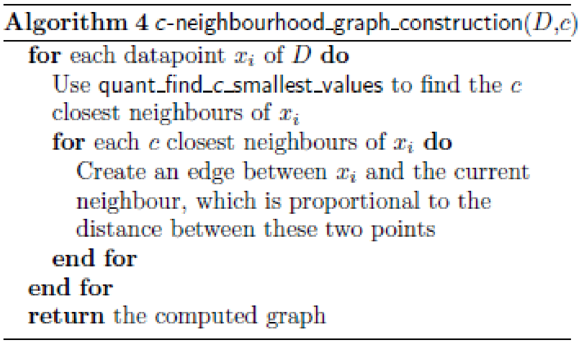
Another popular classical clustering algorithm is k-medians clustering. It is derived from a much more well-known algorithm, known as k-means clustering. Whereas divisive clustering was recursive, k-medians is an iterative algorithm consisting of two distinct steps. In the first step, each example in D_n is attached to the closest cluster center (all k cluster centers are initially randomized). Then, in the second step, each center is updated by choosing the point in the cluster that is the median of all the cluster points. These steps continue until the cluster centers stabilize (or at least approach stabilization). The parameter

k can be altered depending on the complexity of the example set, and how many clusters the implementer actually wants to consider [1].

In order to consider a quantum version, we must choose k-medians rather than k-means. In k-means clustering, it is possible that the center of the cluster will not actually be a point in D_n (since it is computed as the average of all points in the cluster, rather than the median point). Our distance oracle is configured to only measure distance between points D_n, thus we will be in trouble if we try to feed the oracle one of these ”mean” points, that don’t actually exist in D_n. Thus, we must implement k-medians clustering in a quantum setting [1]. A comparison of k-medians and k-means can be found in Figure 8, along with the already-outlined pseudocode. We will again analyze the classical and corresponding quantum run-times in **Theoretical Results**.

2.4.3 Quantum C-Neighborhood Graph





```
Algorithm 4 c-neighbourhood_graph_construction($D$,$c$)
  for each datapoint $x_i$ of $D$ do
    Use quant_find_c_smallest_values to find the $c$ closest neighbours of $x_i$
    for each $c$ closest neighbours of $x_i$ do
      Create an edge between $x_i$ and the current neighbour, which is proportional to the distance between these two points
    end for
  end for
  return the computed graph
```

Figure 9: Left: A visual example of a c-neighborhood graph (c=3); Right: Pseudo-code for iterative Quantum c-neighborhood graph construction

One common algorithm implemented in a variety of unsupervised learning algorithms is the construction of a neighborhood graph. Unlike the previous two clustering examples, we now consider our example set D_n as vertices in a complete graph. This means that there is an edge between every node, and a weight on each edge corresponding to the distance between each vertex. The number of edges is $\sum_{i=1}^{n} i = \frac{n(n-1)}{2}$. This many edges is far more than necessary, thus we want to transform it into a c-neighborhood graph to cut down on the number of edges to consider. This can be done by keeping, for each vertex, only the edges connecting it to its c closest neighbors (as determined by the weights on each edge). To do this, we simply loop through every example in D_n. For each example, we find it’s c closest neighbors, and create an edge between them with a new weight that is proportional to the distance (so that, when exploring later nodes, we don’t overlook already created edges). At this point, it may no longer be a surprise, but this is easily converted into a quantum algorithm. The pseudocode for the quantum algorithm, as well as an example of a c-neighborhood graph, can be found in Figure 9 [2]. Efficiency analysis is detailed in **Theoretical Results**. (It should be worth noting that, although *quant_find_c_smallest_values* is not explicitly defined anywhere, it is just a quick modification to the existing *quant_find_max algorithm*.)

3 Theoretical Results (and Lack of Empirical Results)

3.1 Turing Machines and Computational Complexity

In order to see the possible power of quantum computing, it would be helpful to compare it's computational power to classical computing. The basis of computational complexity theory is built upon the *Church-Turing Thesis*, which states that ”a computing problem can be solved on any computer that we could hope to build, if and only if it can be solved on a very simple ’machine’ named a *Turing machine*.”[10] Despite the implication of the name, a Turing machine is not a physical machine, but a mathematical model. A Turing machine consists of a finite set of states, and an infinitely long ’tape.’ This tape has symbols from a finite alphabet written on it, and is read one at a time. The final component is a transition function, which specifies the next state of the Turing machine based on the current state and the current symbol being read. These components together can be seen in Figure 10. Thus, a problem is computable by a Turing machine if and only if it can be computed by some type of realistic device (finite dimensional, but can run for an infinite amount of time if never stopped)[10].

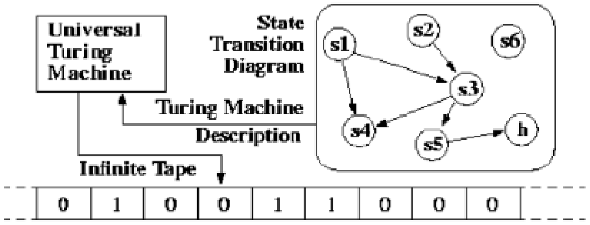


Figure 10: All components of a simple Turing Machine

Recall that any problem in *P* is a problem that can be solved in polynomial time. An equivalent definition of a *P* problem, then, is a decision problem that can be solved by specifically a Turing machine in polynomial time. Similarly, problems in *NP* take exponential time on a deterministic Turing machine. The hardest of the NP problems fall under the subset class of *NP*-complete problems, known as *NPC*. All of these concepts, stemming from Turing machines, boil down to the long-open problem of if $P = NP$ or not[14].

However, the original Church-Turing Thesis only says whether a problem is computable or not. It says nothing on the efficiency of computation. In order to extend the original Church-Turing Thesis to say something about the efficiency of computations, we must of course provide some definition of efficiency. We say that a machine's solution to a problem is **efficient** if the required amount of resources (time or space) used by the simulation is *polynomial*. Also, we must introduce a new kind of Turing machine, known as a *probabilistic* Turing machine. This special Turing Machine is able to make decisions based on a random binary choice (it is said that they have a built-in 'coin-flipper'). This small source of randomness is so powerful that there are certain problems in which we have solved efficiently with probabilistic Turing machines, but **not** deterministic Turing machines (such as finding square roots in finite fields of prime size). [14][10].

With the concept of probabilistic Turing machines, we are able to introduce new complexity classes, such as Bounded-error Probabilistic Polynomial Time (*BPP*). This class consists of problems that can be solved in polynomial time, specifically by a probabilistic Turing machine (with a chance that the solution is incorrect). However, the probability of correctness must be greater than $\frac{2}{3}$, which is where the Bounded-error in the name manifests. And although the number of problems in *BPP* that are not in *P* has shrunken considerably over the years, it has not yet been proven that $P = BPP$[14]

Recently, with the introduction of quantum computing, there have been a number of new complexity classes added to the polynomial hierarchy. Most notably is the Bounded-error Quantum Polynomial time, or *BQP*. It is basically just the quantum extension of *BPP*, or the set of all problems that can be solved in polynomial time by a probabilistic *quantum* Turing Machine, with the same error bound as the *BPP* class. And while it is suspected that $P = BPP$, it is not the same for *BQP*. Rather, some suspect that $P \subset BQP$, which implies that there exist some problems that quantum computers can solve in polynomial time, but that classical computers are not capable of![14] This is the driving inspiration behind mainstreaming quantum computing: to essentially convert *NP* problems to *BQP* problems. However, it also suggests volatility in the usage of quantum computers. If sometime in the future, it is found that $P = BQP$, then essentially quantum computing becomes useless, as $P = BPP = BQP$. This would mean that any classically-probabilistic Turing Machine could do whatever a quantum-probabilistic Turing Machine could also do, including solving those coveted NP problems in polynomial time. A helpful graphic representing a potential polynomial hierarchy can be seen in Figure 11, along with popular problems in each area.

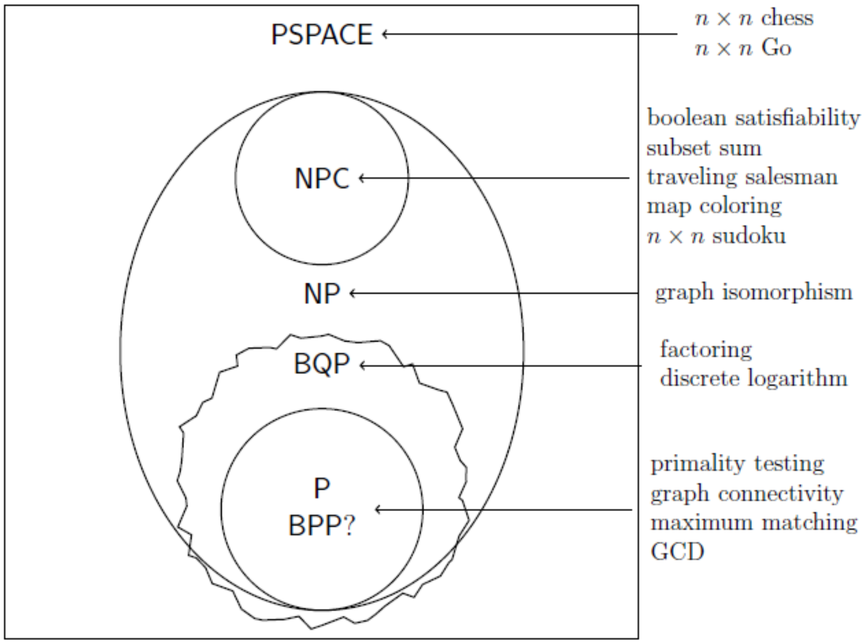


Figure 11: A possible polynomial hierarchy

3.2 Efficiency of Select Quantum Algorithms

We will first consider Divisive Clustering, to see how much faster, if at all, the quantum version is. Clearly, the most costly part of *Div_clustering* is finding the farthest points in each sub-cluster. As we mentioned before, this would classically take $O(n^2)$ comparisons (comparing every possible pair in our example set). While quadratic run-time is not necessarily horrible, it is definitely not ideal for extremely large datasets. This makes Divisive Clustering a perfect candidate to quantize. Thanks to our predefined algorithm *quant_find_max* and utilization of Grover's algorithm, we are able to improve the run-time of Divisive Clustering to $O(n)$.

To compare the efficiency of classical and quantum k-medians clustering, we must make some assumptions about the current distribution of the clusters. For simplicity, we will assume each cluster has approximately $\frac{n}{k}$ points inside them. For each cluster, classically finding the median would require a time of approximately $\Theta((\frac{n}{k})^2$, implying a total runtime of $\Theta(\frac{1}{k}n^2)$ to find all medians. From Dürr and Høyer's work in 1996, we have seen that, in a quantum setting, we can calculate a single median in $\Theta(\frac{n}{k}\sqrt{\frac{n}{k}})$ time [7]. Thus, calculating all k medians in a quantum setting can be done with a runtime of $\Theta(\frac{1}{\sqrt{k}}n^{3/2}$, which is in the order of $\sqrt{n/k}$ faster than the classical approach.

Based on the previous work, it is easy to see that for quantum c-neighborhood graph construction, we can also find an example's c closest neighbors in $\Theta(\sqrt{cn})$ by using *quant_find_c_smallest_values*. Thus, assuming c is a constant leads to a total cost of $\Theta(n^{3/2})$ for the quantum twist. If we are classically trying to construct such a graph, it clearly would take $\Theta(n^2)$, as it would require nested loops over all examples in D_n. Thus, similarly as for k-medians, we see a slight improvement in efficiency by switching to the quantum version. However, there is a significant improvement to be made to the classical algorithm. If we have access to all d-attributes used to describe each $x_i \in \mathbb{R}^d$, then it is possible to decrease construction time to $\Theta(clogn)$ using what are known as kd-trees[4]. Thus, perhaps a quantum approach is not better than it's classical counterpart.

Both quantum clustering algorithms and the c-neighborhood graph constructor are all highly dependent on the distance oracle, which is a magical blackbox function that we have been assuming to exist all of this time. In the real world, we are most likely not going to just be given this black box, but instead would have to construct our own black box (given our training set D_n). However, there has not been much progress towards finding an efficient, generalized quantum circuit that does what our distance oracle has been doing all along. Therefore, right now, quantum clustering cannot truly be implemented [1].

Also, the current quantum implementation of k-medians clustering is only $\sqrt{n/k}$ faster than the classical approach, which is not nearly fast enough to be a viable choice. This is mostly due to the amount of time it takes to add all states the calculate the median. In Grover's original paper, he proposed an algorithm to estimate the average over set of values, but this is not reliable due to precision issues[9]. There are, however, some promising methods based on amplitude estimation that could be further explored [5].

For all of these comparisons, it is only fair to consider the best possible classical algorithm available and the quantum counterpart. Especially for the case of c-neighborhoods, it is clear that although the quantized method out-performs a sub-optimal classical method, it can be beaten by an optimal classical method. More research must go into providing lower bounds for these already well-defined classical problems if we hope to actually determine if a quantum version is any better [1].

4 Discussion and Conclusion

We have only scratched the surface of the theoretical power of quantum computing in machine learning. Contingent on the assumption that $BQP \neq BPP$, quantum algorithms have the capability to solve some NP-hard problems that classical computers just cannot tackle. This is because quantum computations are able to operate on all possible states of the incoming qubit-register, whereas classical computers can only focus in the incoming state. In the specific case of applying Grover's algorithm to classical machine learning algorithms, we see that in most cases, a notable efficiency upgrade is achieved. However, there are certain uncertainties holding these quantum algorithms back. First: how can we efficiently create an oracle quantum circuit, and how can generalize the black box oracle across any given example set? Second, how can we improve existing quantum gates to make all aspects of them quantum- such as calculating a median of points probabilistically?

The previous two questions are both implementation questions. We have not, however, even considered the physical capabilities of quantum computers so far. Right now, quantum computing is on the verge of being actually useful. Technology and hardware are advancing at a quick pace, and researchers are able to build systems that can handle more and more qubits at once. In fact, just last month IBM announced that developed the first 50-qubit quantum computer. At first glance, the article title seems extremely impressive. However, upon reading it further, one quickly realizes that the news is a bit too good to be true, as the quantum computer can only preserve the 50-qubit register's state for 90 microseconds. This is a record for the quantum computing industry, but still quite a useless amount of time.[11].

One reason for the difficulty in keeping a qubits state is the sheer sensitivity of the qubit. We must remember that a qubit is literally a single particle, thus any form of interference, no matter how small, can completely alter it's superposition or even lead to an early measurement, completely collapsing the superposition. Also, qubits must start in their low-temperature ground states ($|0\rangle$ to be useful for most algorithms. When we send these qubits through quantum gates, they begin to heat up. Thus, an extremely strong cooling unit is necessary to reset these qubits (and standard computer fans cannot do the job quickly). Parallel advancements are being made to improve cooling systems for these quantum computers, with what are known as nanofridges. These nanofridges utilize qubits in an very interesting way, forcing them to behave as resonators, omitting photons to boost other electrons in the system. As the resonator omits more and more photons, it cools down, and also removes heat from the entire system. Thus, it can cool multiple qubits at once. [12]

The concept of quantum computing itself was by far the most compelling area of my research. It truly is a whole other side of mathematics and computational theory, with some parallels, but also a lot of counter-intuitive ideas as well. Perhaps one of the most interesting questions that quantum physics brings up is: why does observing a particle collapse it's super position? Why is physical detection incapable of detecting superpositions? This delves into the much more philosophical questions of quantum physics, like the Heisenberg Uncertainty Principle and Schrödinger's cat. Is there some deterministic way of detecting superpositions that we are not aware of yet? And if so, how incredibly life-altering would this become? It's a fascinating field, one that deserves more than a few months of researching, especially since my studies were extremely surface level (enough to understand ML applications).

However, most of the machine learning applications to quantum computing that I stumbled upon were either far too complicated, or rather boring.

1. **Far too complicated to comprehend at a level that would appropriate for this paper:** Quantum computing uses many advanced linear algebra topics (Lie Algebra, tensor products, functional approximations) that are not intuitive at first glance, and would be incredibly hard to portray in this paper without devoting pages of explanations for them. However, the resulting exponential efficiency boosts, like that of kernelized least-squares SVMs[13], are beautiful theoretical results.

2. **Rather boring, but provide tangible and easy to understand results:** The topics I touched upon were ones that were rigorously defined and simple enough to properly understand and to connect to other topics touched upon in the

survey (like complexity theory). The clustering algorithms I found are all extensions of Grover's algorithm, one of the most famous and well-defined quantum algorithms in existence. However, the efficiency boosts were almost always polynomial increases, if that.

All in all, quantum computing is a truly fascinating realm of computer science, and offers many promising improvements to machine learning algorithms. However, we will not see any of these improvements until hardware demands are met, and the few theoretical gaps that exist are filled in.

References

[1] Aïmeur, Esma, et al. ”Quantum Clustering Algorithms.” *Proceedings of the 34th International Conference on Machine Learning*, 2007.

[2] Aïmeur, Esma, et al. ”Quantum Speed-up for Unsupervised Learning.” *Machine Learning*, vol. 90, no. 2, 2012, pp. 261287.

[3] Anguita, Davide, et al. Quantum Optimization for Training Support Vector Machines. *Neural Networks*, vol. 16, no. 5-6, 2003, pp. 763770

[4] Bentley, J. L. ”Multidimensional binary search trees used for associative searching.” *Communications of the ACM*, 18(9), 509517.

[5] Brassard, G. et al ”Quantum amplitude amplification and estimation.” *Contemporary Mathematics*, 305, 5374, (2002)

[6] Ciliberto, Carlo, et al.Quantum Machine Learning: a Classical Perspective. 8 Aug. 2017.

[7] Dürr, C. and Høyer, P. ”A quantum algorithm for finding the minimum” (1996).

[8] Gershenfeld, Neil and Isaac L. Chaung. ”Quantum Computing with Molecules.” *Scientific American, Inc,* June 1998

[9] Grover, Lov K. ”A Fast Quantum Mechanical Algorithm for Database Search.” 19 Nov. 1996

[10] Kaye, Phillip, et al. ”An Introduction to Quantum Computing.” *Oxford University Press*, 2007.

[11] Knight, Will. IBM Announces a Trailblazing Quantum Machine. *MIT Technology Review,* 13 Nov. 2017, www.technologyreview.com/s/609451/ibm-raises-the-bar-with-a-50-qubit-quantum-computer/.

[12] Ouellette, Jennifer. Nanofridge Could Keep Quantum Computers Cool Enough to Calculate. New Scientist, Daily News, 8 May 2017, www.newscientist.com/article/2130210-nanofridge-could-keep-quantum-computers-cool-enough-to-calculate/

[13] Rebentrost, Patrick et al. ”Quantum support vector machine for big data classification.” 10 July 2014

[14] Wittek, Peter. ”Quantum Machine Learning: What Quantum Computing Means to Data Mining.” *Academic Press*, 2014.