Yutong Du, Jean-Luc Lupien Physics C191
Final Project

# Quantum Speedups to Semidefinite Programming

May 5, 2024

Header styling inspired by CS 70: https://www.eecs70.org/

## 1 Introduction

Semidefinite programming (SDP), or more generally the problem of convex optimization, is a class of problems that has many practical applications, and is also incidentally one class of problems that classical computers are relatively weak at. In this report, we first establish the framework of a semidefinite program, discuss the leading solution philosophies, then explore how research into quantum SDP solvers has produced significantly faster algorithms than their classical counterparts. The investigation into these algorithms continues to push the boundaries of quantum computing, and is another shining example of quantum advantage.

### 1.1 Contribution Statement

For this paper, the contribution is as follows: Yutong Du was responsible for the introduction and description of linear and semidefinite programs. Yutong was also responsible for the QRAM section and the discussion of the main result (and subsequent sections including the discussion of spectral and matrix-vector approximation) of the quantum algorithms. Jean-Luc was responsible for the literature review and algorithm comparison which are reflected in sections 4.1 and 4.4.1. Jean-Luc also wrote the conclusion. All other sections are representative of combined work.

### 1.2 Linear Programs

To begin, let's look at a simpler problem: linear programming. In linear programming (LP), the goal is to find the optimal (either a maximum or minimum) value of a *linear objective equation*, which is an equation of the form:

$$c_1x_1 + c_2x_2 + c_3x_3$$

For convenience, we also employ a matrix vector representation – the above equation can also be written as $\mathbf{c}^\top\mathbf{x}$, where the vector $\mathbf{c} = \begin{bmatrix} c_1 & c_2 & c_3 \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$. Along with this equation, we also place any number of linear constraints on the variables $x_1, x_2$, and $x_3$. For instance, if we had 2 constraints:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 &\leq b_2 \\ x_1, x_2 &\geq 0 \end{aligned}$$

this can also be written using matrix-vector notation: $\mathbf{A}\mathbf{x} \leq \mathbf{b}$. Here, $\mathbf{A}$ is the matrix that gives us the coefficients of the constraint equations, $\mathbf{x}$ is the same vector as before, and $\mathbf{b}$ is the constraint value. Finally, we have a nonnegativity constraint on the variables, which can be written as $\mathbf{x} \geq 0$. Putting this all together, the following is the statement of linear programs:

$$\begin{aligned} \min_{x} \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b}, \quad \mathbf{x} \geq 0 \end{aligned}$$

Note that here we've written the problem in terms of a minimization, but it is always possible to convert a maximization problem into this form. Another thing to note is that in this formulation, the set of constraints $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ defines a *convex* region, also called the feasible region, in which the optimal value is to be found – this will be important later.

## 1.3 Semidefinite Programs

Now we move to the main focus of this project: semidefinite programs (SDPs). Here, instead of dealing with vectors as our variables, we now deal with positive semidefinite matrices. These are matrices that are symmetric and also have real eigenvalues, or more formally speaking, a matrix $\mathbf{M}$ is positive semidefinite if and only if:

1. $\mathbf{M}$ is symmetric
2. For any vector $\mathbf{v}$, we have $\mathbf{v}^\top \mathbf{M} \mathbf{v} \geq 0$.

[1]. With that in mind, the spirit of the problem is the exact same as an LP: we have an objective function to maximize, subject to a set of $d$ constraints:

$$\begin{aligned} \min \quad & \operatorname{Tr}\left(\mathbf{C}^\top \mathbf{X}\right) \\ \text{s.t.} \quad & \operatorname{Tr}\left(\mathbf{A}_i^\top \mathbf{X}\right) \leq b_i \quad \text{for } i = 1, 2, \ldots, d \\ & \mathbf{X} \succeq 0 \end{aligned}$$

Here, $\mathbf{C}, \mathbf{X}$ and $\mathbf{A}_i$ are all positive semidefinite matrices instead of simple vectors. Note that this is a more general class of problem than an LP; if $\mathbf{C}$ and $\mathbf{X}$ are diagonal, then this problem reduces immediately to an LP.

While LPs and SDPs share thematic similarities, the generalization of moving from vectors to semidefinite matrices introduces some key differences [2]:

1. Properties concerning duality are lost. For LPs the concept of duality states that for every LP, also called the *primal* LP, we can construct another "equivalent" LP called the *dual* LP. Moreover, the strong duality theorem states that if the primal LP has an optimal value, then the dual LP also has an optimal value, and the optimal value for the dual is equal to that of the primal.

   With SDPs, strong duality fails. The duality gap, defined to be the difference between the optimal primal and dual problems, can be finite or even infinite. However, if both problems contain solutions in the semidefinite cone, then the primal and dual share the same optimal value.

2. There are no known finite algorithms for solving SDPs. As we'll see, all of the solution methods always approximate the solution, to some tolerance of $\epsilon$.

Despite these differences, however, SDPs are not that much harder to solve than LPs, and in fact many LP algorithms have been generalized to also work for SDPs as well.

## 2 Classical Algorithms

Before diving into the quantum algorithms for this problem, it is useful to first take a look at the leading philosophy developed to solve this problem classically. In particular, Interior Point Methods (IPM) are a incredible tool for solving SDPs (and therefore LPs). First introduced by Nesterov and Nemirovsky [3], these methods are widely used in mathematical and engineering contexts and have great practical performance.

### 2.1 Interior Point Methods

In an Interior Point Method, the SDP is rephrased as a functional problem: the objective function is denoted $f_\eta(x)$, defined as:

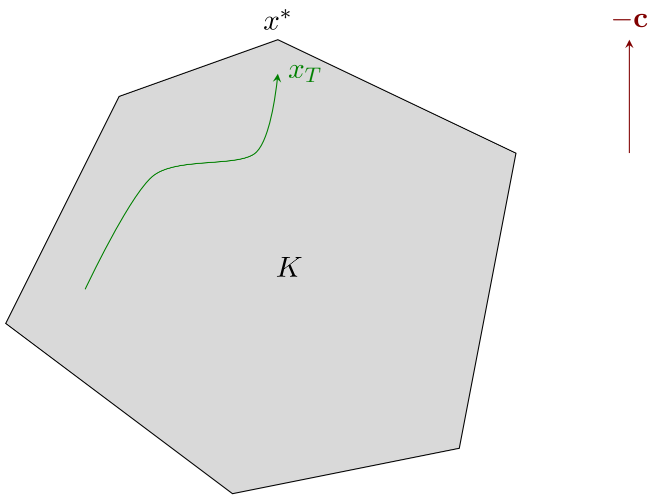
$$\min f_\eta(x) \quad \text{where} \quad f_\eta(x) = \eta c^\top x + f(x) \tag{1}$$

In this formulation, the original objective function $\mathbf{c}^\top \mathbf{x}$ corresponds to the first term, and $f(x)$ is a function known as a *barrier function*. Without defining them explicitly, one way to conceptually interpret $f(x)$ is that it's a function that tends toward infinity as we approach the barrier of the feasible region. In other words, $f(x)$ is the function that encodes the information about the constraints, i.e. $\mathbf{A}_i^\top \mathbf{X}$ for SDPs. There are multiple ways this barrier can be implemented, and as it turns out, different ways of writing $f(x)$ gives rise to drastically different runtime bounds.

The parameter $\eta$ controls the contribution of the $c^\top x$ term in $f_\eta(x)$. When $\eta$ is small, the constraints dominate, and the objective function dominates when $\eta$ is large. In an IPM, we initialize $\eta = 0$ and compute an approximate solution to equation (1) using iterative Newton steps. The value of $\eta$ is then increased and another Newton step is performed. These two steps are repeated until $\eta$ becomes sufficiently large to approximate the solution to the problem within a tolerance $\epsilon$. The point $x_{n+1}$ obtained after $n+1$ iterations is given by: $x_{n+1} = x_n + n(x)$, where $n(x)$, the Newton step, is defined as:

$$n(x) = -H(x)^{-1} g(x). \tag{2}$$

Here, $H(x)$ and $g(x)$ are the Hessian matrix and gradient vectors respectively. Below we've included a diagram showing the path the IPM method takes from $x_0$ to $x_T$ in $T$ steps (this diagram is inspired by [4]):



In theory, our point $x_n$ equals $x^*$ when $\eta \to \infty$ and $T \to \infty$. As this is not possible in a finite process, the best we can hope to do is to get approximately close to $x^*$, which echoes what we mentioned earlier about the lack of a finite algorithm for solving SDPs.

However, we can leverage the inherent properties of SDPs and LPs to bound the number of iterations needed given a desired precision $\epsilon \in \mathbb{R}$. It is proven in [5] that the number of Newton step iterations is $O(\log \frac{1}{\epsilon})$. Notably, this bound holds for both classical and quantum methods. Therefore, speedup is achieved by reducing the time taken to perform each Newton step not in reducing the total number of iterations.

Each Newton step is an incredibly costly computation, since it requires calculating a matrix inverse and a matrix multiplication. Classically, the fastest known matrix multiplication algorithm runs in $O(n^{2.37})$ [1], and the fastest matrix inverse algorithms run close to $O(n^3)$, both of which are incredibly slow.

With this in mind, there are several approaches we can take to easing the computational burden here. Perhaps the most obvious way is to find a faster way to perform matrix multiplication and inversion, which is a valid approach, however it's hard to imagine that this will generate appreciable speedup as it does nothing to affect the number of iterations we have to compute. On the other hand, a more promising approach, and one that we will focus on in subsequent sections, lies in finding ways to shrink the matrices $H$ and $g$ to make them more tractable, by way of approximations. This lessens the computational burden in two ways: first, a smaller $H$ and $g$ means calculating $H^{-1}$ and multiplying it with $g$ is faster, and second, approximations reduce the number of times we need to compute Newton's step.

In sum, there are three main steps that incur a high computational cost when using an IPM which are the following:

1. Calculating and storing the Hessian matrix $H(x)$ and the gradient $g(x)$
2. Calculating the inverse of the Hessian matrix $H(x)^{-1}$.
3. Calculating the matrix-vector product $H(x)^{-1}g(x)$.

If we consider a constant tolerance $\epsilon$, the best classical runtime achieved is somewhere on the order of $\widetilde{O}(nd + d^3)$. As we'll see, the quantum algorithm we focus on has an optimal runtime of $\widetilde{O}(\sqrt{n}\,\mathrm{poly}(d))$, which is much faster.

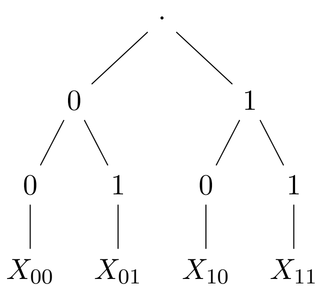
## 3 Quantum Random Access Memory

A crucial subroutine in many quantum solvers is the quantum implementation of classical random access memory, also called QRAM. We will now discuss this subroutine and its importance to quantum solvers. In essence, QRAM provides an $O(n) \to O(\log n)$ speedup in query length, and is one of the major subroutines used to achieve a better runtime.

The philosophy behind QRAM is relatively simple: instead of accessing a single memory location, QRAM uses superposition to access multiple memory locations at once. This can be implemented in different ways, one of which is the "bucket-brigade" implementation described in [6]. To explain how this works,

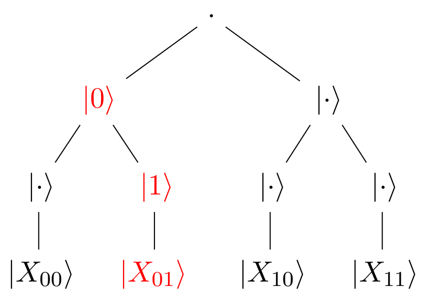
[1]This algorithm is also considered a "galactic algorithm", or in other words, extremely impractical, and in practice the slower (but much easier) Strassen's algorithm is used instead, which has a runtime of roughly $O(n^{2.8})$.

consider a classical RAM with $n$ bits, which encodes for $2^n$ memory locations. Now, since there are $2^n$ locations, we can imagine each memory location being a leaf in a full binary tree with $n$ layers:



Then, the data at location $X_{ab}$ can be specified by the binary string $ab$, where the value of $a$ and $b$ trace the path from the root node to the leaf in question. Note that the path is traced from most to least significant bit – for instance, $X_{01}$ is encoded by the bits $a = 0, b = 1$.

Quantumly, the idea is more or less the same, except now, each node is replaced by a three-state system, sometimes called a qutrit. The states are $|\cdot\rangle$ , $|0\rangle$ and $|1\rangle$. The $|\cdot\rangle$ state is referred to as a "wait" has the special property that whenever a state, either $|0\rangle$ or $|1\rangle$ is received, the state transforms into the received state. That is, if it receives $|0\rangle$, then $|\cdot\rangle \rightarrow |0\rangle$, and the same goes for $|1\rangle$. Then, to access a particular cell, we specify the path to that cell by feeding in the path one qubit at a time, just like we did classically; the diagram below shows an example where we try to access $|X_{01}\rangle$:



the red markers indicates the path taken to $|X_{01}\rangle$. Now, how is this faster than the classical approach? Well, the key comes when we consider superpositions of states. What if, instead of feeding in just $|0\rangle$ or $|1\rangle$, we instead fed $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ twice? Well, then both paths would be activated due to superposition, and as a result instead of getting a single qubit as earlier, we would get a superposition of all four memory states:

$$\frac{1}{2}(|X_{00}\rangle + |X_{01}\rangle + |X_{10}\rangle + |X_{11}\rangle)$$

In general, for a QRAM of $n$ qubits, if we gave $|+\rangle^{\otimes n}$ as an input (in succession), then the output would just be a superposition of all the memory states. This is the power of QRAM – with only $n$ queries, we managed to access $2^n$ different memory locations, or equivalently, for $n$ memory locations we only need $\log n$ queries. This is why QRAM introduces an $O(n) \rightarrow O(\log n)$ speedup.

As we'll see, QRAM is one of the central subroutines that are assumed to be implemented in the algorithm we will focus on. This is also true for many other quantum algorithms out there – they assume access to QRAM, and as a result they are able to claim a $O(n) \rightarrow O(\log n)$ query time basically for free.

## 4 Quantum IPM Algorithms

Now that we are familiar with optimization problems and challenges in implementing an efficient algorithm classically, we move to investigating how quantum algorithms can be used to generate speedups. In particular, we will focus on the paper by Apers and Gribling [7], who propose a new quantum IPM algorithm that bypasses many of the challenges encountered by the algorithms that precede it.

Before we begin discussing the algorithm, it is useful to first take a look at some other algorithms from the literature that attempt to speedup convex optimization.

### 4.1 Optimization Approaches from the Litarature

Because of the inherent efficiency in terms of Newton iterations obtained by IPMs, most algorithms use the same general architecture as classical IPMs. A few such techniques are listed here. A seminal work in this space was presented by Brandao in [8] which leverages a quantum matrix multiplicative weight approach to achieve speedups for solving SDPs. Specifically, supposing quantum access to the matrices encoding the SDP, this algorithm achieves $\widetilde{O}(\sqrt{d} \cdot \text{poly}(\log d, \log n, r, 1/\epsilon)$, where $r$ denotes the maximum rank of input matrices and $\epsilon$ is the required precision. An improvement to this algorithm is presented in [9] which uses an approximate quantum linear solver to identify inexact search directions. This paper addresses the largest issue associated with inexact search techniques which is the possibility of obtaining infeasible solutions. Recently, a quantum implementation of the fast SDP solver [10] was presented in [11] which exhibits very fast $O(d^{1.5}n^2 \log(1/\epsilon)$ time complexity. This complexity is achieved under very strict assumptions however and notably depends on a condition-number.

A different approach to optimization of LPs is presented in [12] which encodes the central path of an IPM into a Hamiltonian. The evolution of this Hamiltonian is then performed using quantum Schrodinger simulation which will converge with high probability to the optimal solution of the LP. This technique presents two distinguishing features: it does not depend on a bounded condition number of the input matrices and it does not require multiple iterations to reach the desired solution. However, this comes at the expense of a high number of required quantum gates. Indeed, the number of quantum gates required is $O((n + d)\text{nnz}(A) \log(n)\text{polylog}(d, n))$, where $\text{nnz}(A)$ denotes the number of non-zero elements of $A$. This makes actual use of this algorithm extremely impractical given the current difficulties with creating large-scale quantum circuits.

### 4.2 Main Result

One of the main obstacles in obtaining a good quantum SDP solver is the assumption of a bounded condition-number. This is highlighted by approaches presented in [8, 9, 10] which are all dependent on a bounded condition-number. In essence, a condition number bound implies that the runtime of the algorithm was highly dependent on the values to the input matrices; this is problematic because it means that the runtime of such algorithms varies wildly depending on the values to the input matrix. However, this algorithm we will focus on removes that dependence entirely, and as such is seen as a monumental achievement in the search for a faster quantum IPM algorithm.

In particular, this algorithm develops two main subroutines [7]:

a) An efficient algorithm to spectrally approximate the Hessian.

b) An efficient algorithm to approximate a matrix-vector product.

The implementation of these algorithms allow us to calculate an approximate Hessian $Q(x)$ and gradient $\tilde{g}(x)$, which is then used to approximate the update step via $\tilde{n}(x) = -Q(x)^{-1}\tilde{g}(x)$. To be a bit more specific, it turns out that in this particular case $\tilde{g}(x)$ can be calculated directly from $Q(x)$, which is why we don't need an efficient algorithm to calculate $\tilde{g}(x)$ alone.

## 4.3 Quantum Spectral Approximation

As this is one of the main subroutines the paper leverages, it is one that is important to look at if one is to understand the entire algorithm. As outlined in Theorem 3.1 of [7], the following is the formal statement of the spectral approximation:

**Theorem 1** (Quantum Spectral Approximation). *Consider query access to a matrix $B \in \mathbb{R}^{n\times d}$ with row sparsity $r$. For any $0 < \epsilon \leq 1$, there is a quantum algorithm that, with high probability, returns a matrix $\tilde{B}^{\widetilde{O}(d/\epsilon^2)\times d}$ satisfying*

$$(1-\epsilon)\tilde{B}^\top\tilde{B} \preceq B^\top B \preceq (1+\epsilon)\tilde{B}^\top\tilde{B}$$

*while making $\widetilde{O}(\sqrt{nd}/\epsilon)$ row queries to $B$, and taking time $\widetilde{O}(r\sqrt{nd}/\epsilon + d^\omega)$.*

Intuitively, this theorem is best understood as the following: given a matrix $B$ in $\mathbb{R}^{n\times d}$, we can use this algorithm to find a matrix $\tilde{B}$ such that $\tilde{B}^\top\tilde{B}$ is within $\epsilon$ of the true value of $B^\top B$.[2]

The reason this spectral approximation is phrased in this way is in part due to the form of the Hessian matrix for $f(x)$. In summary, because all the Hessian matrices considered in the paper are of the form $B^\top B$, then the above algorithm perfectly allows us to approximate $H$.

The method in which this is done is actually quite complicated – the authors make use of an algorithm called the *repeated halving algorithm* developed in [13]. In essence, the algorithm works in two parts: first, a set of matrices $A_1, \ldots, A_L$ is recursively constructed by subsampling $A$; in other words, to generate $A_{i+1}$, we keep each row in $A_i$ with probability $\frac{1}{2}$. This is then combined another subsampling procedure that allows us to construct a matrix $B_i$ from each $A_i$. In the paper, this is denoted as the step $B \overset{w,\epsilon}{\leftarrow} A$. Here, instead of each row being kept with uniform probability, there is a distribution $p_i$ instead, which is dependent on the dimension of $A$ along with other factors. This process is recursively done, and once all the computation is done we are left with a matrix $B$ which approximates $A$ in the sense that $B^\top B \approx_\epsilon A^\top A$. This notation effectively means that $B^\top B$ is approximately equal to $A^\top A$, to a tolerance of $\epsilon$.

Quantumly, this latter process of $B \overset{w,\epsilon}{\leftarrow} A$ is interesting, because its quantum implementation is actually rather simple. In particular, this step is actually implemented via Grover search [14], which is used to select the rows in $A$ that we want to use to construct $B$. It should also be noted that this step in particular is one that benefits greatly from QRAM access, since accessing multiple rows of $A$, which is necessary to construct $B$ would be made far more efficient with QRAM as we can access all these rows at once.

This result is then used in conjunction with calculations of leverage scores and Lewis weights in order to fully compute an approximation to $H(x)$. However, the process by which the Lewis weights are calculated

[2]More accurately, we should say that if $A \preceq B$ then this is the same as $0 \preceq B - A$, or in other words the matrix $B - A$ is positive semidefinite. However, the analogy of "values" suffices for a conceptual understanding.

are relatively complicated, and we will not analyze them in detail in this report. However, we will point to [7] for further reading.

## 4.4 Gradient Estimation

Perhaps the more interesting of the two subroutines introduced is this efficient method of calculating the matrix-vector product used to define the gradient. In particular, the algorithm presented in the paper of focus [7] uses a quantum matrix-vector approximation algorithm to estimate the gradient of the barrier function. Before we delve into the specifics of this algorithm, it is useful to look historically at what's been done already.

### 4.4.1 Other Gradient Estimation Algorithms

Gradient estimation has been an area of major research for quantum algorithms and is pivotal for most quantum SDP solvers. One of the first such algorithms is Jordan's algorithm [15] which supposes black-box access to a function and can compute the gradient of said function in constant time. This algorithm is of little practical use for SDP solvers, however, as we have analytical access to the functions in question. Another approach that does make use of analytical information is a parameter shift technique for calculating gradients of quantum functions. Given a function encoded in a quantum circuit, parameter-shift rules are employed to calculate the gradients from the same circuit architecture as that of the original quantum function. This result is presented in [16]. This technique is difficult to apply to general optimization problems, however, as the barrier function would have to be encoded within quantum circuits prior to running the optimization. This incurs a higher time cost rather than just storing and accessing the constraint matrices through QRAM.

Gradients can also be estimated though shadow tomography as seen in [17]. However, this process requires many copies of an initial quantum state which would render the process too complex for use in optimization.

### 4.4.2 Quantum Matrix-Vector Approximation

Now we turn to the algorithm outlined in [7]. As mentioned, it is effectively a gradient estimation algorithm that uses previous quantum matrix-vector approximation algorithms, and is outlined in Theorem 5.1 of [7]:

**Theorem 2** (Approximate matrix-vector product)**.** *Assume query access to a vector $v \in \mathbb{R}^n$ and a matrix $B \in \mathbb{R}^{n \times d}$ with row sparsity $r$. There is a quantum algorithm that returns a vector $\tilde{y}$ satisfying*

$$\|\tilde{y} - B^\top v\|_{(B^\top B)^{-1}} \leq \delta$$

*while making $\widetilde{O}(\sqrt{n}d\|v\|_\infty/\delta)$ row queries to $B$ and $v$, and taking time $\widetilde{O}(r\sqrt{n}d^2\|v\|_\infty + d^\omega)$.*

In other words, this algorithm takes in an input $B \in \mathbb{R}^{n \times d}$ and $v \in \mathbb{R}^n$, and finds a vector $\tilde{y}$ that gets arbitrarily close (because we choose the value of $\delta$) to the value of $B^\top v$, which is the matrix vector product. In the context of our IPM algorithm, this is the algorithm that finds an approximation to the quantity equivalent to the quantity $Q(x)^{-1}\tilde{g}(x)$.

Similar to the spectral approximation, the implementation of this algorithm is also rather complex, and uses other algorithms such as the quantum multivariate mean estimation from [18] to arrive at a result. Despite this, the main points are relatively easy to understand. First, the input matrix $B$ is spectrally approximated as $\tilde{W}$ using the spectral approximation we talked about earlier as a subroutine. This outputs $\tilde{W}$, which we then convert to $\tilde{W}^{-1/2}$ classically. Now, this quantity is used in conjunction with $v$ to generate a random variable $X$ from which the quantum multivariate mean subroutine is used, ultimately returning a vector $\tilde{\mu}$. Then, we return $W^{1/2}\tilde{\mu} = \tilde{y}$ as the approximation.

While the details as to *why* the algorithm works is rather technical, the important point is the following: the quantum multivaraite mean algorithm is incredibly efficient, and the authors get around having to explicitly approximation $B^\top v$ by using this algorithm to output an estimate $\tilde{\mu}$ which we use to output $\tilde{y}$.

## 4.5 Putting it all Together

Now, we are ready to put everything together. Recall in section 2.1, we discussed that depending on the way we encode our barrier function $f(x)$, the resulting time (and space) complexity of the quantum algorithm is different. In [7], the authors detail the implementation of their quantum algorithm for three different barriers, of which we will focus on the fastest one: the Lewis weight barrier.

The idea of a Lewis weight barrier is inspired by [19], which defines $f(x)$ as follows:

$$f(x) = \ln\det\left(A^\top S_x^{-1} W_x^{1-\frac{2}{p}} S_x^{-1} A\right) \quad \text{where} \quad W_x = \mathrm{Diag}(w^{(p)}(S_x^{-1}A))$$

Where $A$ is the input matrix, and $S_x$ is a matrix calculated based on the Lewis weights, which as mentioned, is a process that we do not fully understand, and will not discuss in detail here. The reason this approach is used is because the gradient and Hessian matrices take on a relatively nice form:

$$g(x) = -A^\top S_x^{-1} W_x \mathbf{1} \quad H(x) = A^\top S_x^{-1} W_x S_x^{-1} A$$

Then, leveraging the forms of these matrices, we can apply the spectral and the matrix-vector approximation algorithms here to approximate these quantities. This is also done in a very clever way: for $H$, instead of spectrally approximating $H$, we spectrally approximate the quantity $B = W_x^{1/2} S_x^{-1} A$ outputting a matrix $\tilde{B}$, with the guarantee that $\tilde{B}^\top \tilde{B} \approx H(x)$. A similar approach is done to approximate $g(x)$, where we allow $B = W_x^{1/2} S_x^{-1} A$ and $v = W_x^{1/2}\mathbf{1}$, whose product is equal to $g(x)$.

This last step highlights the major contribution of this paper: the authors were able to find a way to separate $H(x)$ and $g(x)$ in an incredibly clever way, such that we can apply these algorithms like the spectral and matrix-vector approximation in order to approximate $H$ and $g$ with an extremely fast runtime. In addition to these algorithms, all of these algorithms benefit from QRAM, as they all (at some point in their execution) demand query access to objects multiple times, which is made significantly more efficient with the implementation of QRAM. Then, combining all these algorithms and structures together, we finally arrive at a runtime of

$$\widetilde{O}(\sqrt{n}\,\mathrm{poly}(d, \log(n), 1/\epsilon))$$

which is incredibly fast. Notably, the dependence on $O(\sqrt{(}n))$ rather than $O(n)$ in the classical setting, is the most striking improvement in complexity. This quantum runtime is considered significantly faster than the $O(nd + d^3)$ classical runtime due to the polylog depedence in parameters $d$ and $n$.

However, because this implementation depends on many sampling steps for each iteration of the IPM method, the constant multiplicative factor in front of this complexity result is non-negligible. This makes this algorithm impractical for small calculations where classical methods would have a much faster actual runtime. Furthermore, current quantum computing frameworks do not allow for large-scale implementations which means that it impossible to solve large SDPs with this method. This greatly limits the current applicability of such methods to real-world problems. The hope is to eventually be able to implement large-scale quantum circuits which will be able to solve large SDPs impossible to do classically.

## 5 Conclusion and Future Directions

Overall, this exploration highlights the speedup that can be gained from utilizing quantum algorithms to solve SDPs compared to their classical counterparts. Although the inherent structure of the IPM algorithm is very efficient, quantum algorithms can greatly speedup matrix estimation, linear system solving, and matrix accessing steps integral to the Newton step. Utilizing such quantum algorithms therefore enables considerable speedup when used in conjuction with classical frameworks.

Despite these speedups, there is still much more room for this field to advance. For instance, perhaps the most obvious area of improvement, is to find an experimental way to implement such an algorithm. It is well known that many quantum algorithms, for instance Shor's algorithm, are theoretically faster than classical algorithms, but remain non-functional to this day because we have not yet figured out how to build a fault-tolerant way to implement these algorithms.

This same principle is also what hinders these SDP algorithms. Specifically, as mentioned, many of these algorithms leverage QRAM in order to demonstrate an enormous speedup, and this algorithm we investigated is no different. However, as far as we know, there are no known fault-tolerant ways of implementing QRAM, and as a result such algorithms remain only theoretically possible [20].

There is also further direction on how to improve this particular algorithm outlined in [7]. They suggest that because their gradient approximation runs in $\widetilde{O}(\sqrt{n}d)$ whereas the theoretically optimal bound is $\widetilde{O}(\sqrt{nd})$, there is potential for improvement in this area. However, they haven't been extremely specific on how this may be accomplished, and instead just suppose that alternate, more efficient algorithms are possible.

Finally, classical IPMs have a $\log(1/\epsilon)$ dependence, meaning they can achieve very high precision very quickly. The quantum algorithm of Apers ([7]), however, has a $1/\epsilon$ dependence meaning precision is only reached linearly. This dependency problem has been addressed for specific instances such as in [10] where $\log(1/\epsilon)$ dependence is achieved under strict assumptions and a condition-number bound. The next frontier in this space would be to simultaneously achieve good dependence in $1/\epsilon$, $n$ and $d$ without requiring a condition-number bound.

Regardless though, it will be incredibly interesting tracking the advancements in this field over the next several years.

## References

[1] Rajat Mittal. “Lecture 7: Positive Semidefinite Matrices”. In: ().

[2] Robert M Freund. “Introduction to Semidefinite Programming (SDP)”. In: ().

[3] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[4] Christopher Musco. “Lecture 17: Interior Point Methods”. In: ().

[5] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge, UK ; New York: Cambridge University Press, 2004. 716 pp. ISBN: 978-0-521-83378-3.

[6] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. “Quantum Random Access Memory”. In: *Phys. Rev. Lett.* 100.16 (Apr. 21, 2008), p. 160501. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.100.160501. arXiv: 0708.1879 [quant-ph]. URL: http://arxiv.org/abs/0708.1879 (visited on 05/04/2024).

[7] Simon Apers and Sander Gribling. *Quantum Speedups for Linear Programming via Interior Point Methods*. Apr. 11, 2024. arXiv: 2311.03215 [quant-ph]. URL: http://arxiv.org/abs/2311.03215 (visited on 05/03/2024). preprint.

[8] Fernando G. S. L. Brandão et al. *Quantum SDP Solvers: Large Speed-ups, Optimality, and Applications to Quantum Learning*. Apr. 22, 2019. arXiv: 1710.02581 [quant-ph]. URL: http://arxiv.org/abs/1710.02581 (visited on 05/03/2024). preprint.

[9] Zeguan Wu et al. “An Inexact Feasible Quantum Interior Point Method for Linearly Constrained Quadratic Optimization”. In: *Entropy* 25.2 (Feb. 10, 2023), p. 330. ISSN: 1099-4300. DOI: 10.3390/e25020330. URL: https://www.mdpi.com/1099-4300/25/2/330 (visited on 05/05/2024).

[10] Baihe Huang et al. *A Faster Quantum Algorithm for Semidefinite Programming via Robust IPM Framework*. Feb. 6, 2023. arXiv: 2207.11154 [quant-ph]. URL: http://arxiv.org/abs/2207.11154 (visited on 05/03/2024). preprint.

[11] Baihe Huang et al. *Solving SDP Faster: A Robust IPM Framework and Efficient Implementation*. Nov. 18, 2021. arXiv: 2101.08208 [cs, math]. URL: http://arxiv.org/abs/2101.08208 (visited on 05/03/2024). preprint.

[12] Brandon Augustino et al. *A Quantum Central Path Algorithm for Linear Optimization*. Nov. 7, 2023. arXiv: 2311.03977 [quant-ph]. URL: http://arxiv.org/abs/2311.03977 (visited on 05/03/2024). preprint.

[13] Michael B. Cohen and Richard Peng. *$\ell_p$ Row Sampling by Lewis Weights*. Dec. 1, 2014. arXiv: 1412.0588 [cs, math]. URL: http://arxiv.org/abs/1412.0588 (visited on 05/03/2024). preprint.

[14] Simon Apers and Ronald de Wolf. *Quantum Speedup for Graph Sparsification, Cut Approximation and Laplacian Solving*. May 8, 2023. arXiv: 1911.07306 [quant-ph]. URL: http://arxiv.org/abs/1911.07306 (visited on 05/04/2024). preprint.

[15] Stephen P. Jordan. “Fast Quantum Algorithm for Numerical Gradient Estimation”. In: *Phys. Rev. Lett.* 95.5 (July 28, 2005), p. 050501. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.95.050501. arXiv: quant-ph/0405146. URL: http://arxiv.org/abs/quant-ph/0405146 (visited on 05/03/2024).

[16] David Wierichs et al. “General Parameter-Shift Rules for Quantum Gradients”. In: *Quantum* 6 (Mar. 30, 2022), p. 677. ISSN: 2521-327X. DOI: 10.22331/q-2022-03-30-677. arXiv: 2107.12390 [quant-ph]. URL: http://arxiv.org/abs/2107.12390 (visited on 05/03/2024).

[17] Amira Abbas et al. *On Quantum Backpropagation, Information Reuse, and Cheating Measurement Collapse*. May 22, 2023. arXiv: 2305.13362 `[quant-ph]`. URL: http://arxiv.org/abs/2305.13362 (visited on 05/05/2024). preprint.

[18] Arjan Cornelissen, Yassine Hamoudi, and Sofiene Jerbi. "Near-Optimal Quantum Algorithms for Multivariate Mean Estimation". In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. June 9, 2022, pp. 33–43. DOI: 10.1145/3519935.3520045. arXiv: 2111.09787 `[quant-ph, stat]`. URL: http://arxiv.org/abs/2111.09787 (visited on 05/04/2024).

[19] Yin Tat Lee and Aaron Sidford. "Solving Linear Programs with O( Rank) Linear System Solves". In: ().

[20] Amira Abbas et al. *Quantum Optimization: Potential, Challenges, and the Path Forward*. Dec. 4, 2023. arXiv: 2312.02279 `[quant-ph]`. URL: http://arxiv.org/abs/2312.02279 (visited on 04/17/2024). preprint.

[21] Michael B. Cohen et al. *Uniform Sampling for Matrix Approximation*. Aug. 21, 2014. arXiv: 1408.5099 `[cs, stat]`. URL: http://arxiv.org/abs/1408.5099 (visited on 05/04/2024). preprint.

[22] Zak Tonks. *On Fast Matrix Inversion via Fast Matrix Multiplication*. Jan. 3, 2019. arXiv: 1901.00904 `[cs]`. URL: http://arxiv.org/abs/1901.00904 (visited on 05/04/2024). preprint.