Quantum Semidefinite Programming

Akshay Kumar
Venkata S Balivada
Vinay S Banakar

CS880 Final Report

Abstract

The quantum computing revolution is reshaping the landscape of computational power with the promise of significant speed-ups. Central to this transformation is the burgeoning field of quantum semidefinite programming (SDP). This paper provides a comprehensive exploration of the Quantum Arora-Kale algorithm, a groundbreaking approach that applies quantum mechanics principles to solve semidefinite programs more efficiently. We illuminate the inner workings of the algorithm by first scrutinizing its classical counterpart, the Arora-Kale algorithm, thereby delineating how the quantum version extracts speed-ups from quantum systems. In addition to a thorough overview of semidefinite programs and their duals, the Matrix Multiplicative Weight method, and the quantum Gibbs sampling technique, we highlight recent improvements in the quantum algorithm. Furthermore, we pose open problems that continue to challenge researchers and provide exciting avenues for future exploration. Our discussion draws primarily from the seminal 2017 paper "Quantum Speed-ups for Semidefinite Programming" and the 2005 paper "Fast Algorithms for Approximate Semidefinite Programming using the Multiplicative Weights Update Method". Through this paper, we aim to serve as a comprehensive guide for understanding quantum semidefinite programming, and to underscore its vast potential in propelling the field of quantum computing and optimization forward.

1 Introduction

Optimization forms the crux of numerous applications in science, engineering, and economics. In the realm of optimization, Semidefinite Programming (SDP) stands out as a powerful tool that has been extensively used in a multitude of fields such as machine learning, control theory, signal processing, and quantum information theory [1]. The strength of semidefinite programs (SDPs) lies in their versatility, coupled with the existence of efficient techniques for their resolution, as outlined in the study by [2].

SDP involves the optimization of a linear function subject to the constraint that an affine combination of symmetric matrices is positive semidefinite. Despite the rich applications and the considerable progress made in developing efficient algorithms, solving SDP problems can be computationally demanding due to their high-dimensional nature. Classical algorithms for solving SDPs, such as interior-point methods, while having polynomial time complexity, are not efficient for large-scale problems or cases requiring high precision [3].

In this era of accelerating technological advancements, Quantum Computing emerges as a beacon of hope to overcome these computational barriers. Quantum algorithms, due to their inherent nature, can provide significant speed-ups for specific computational tasks [4].

In the context of SDPs, the seminal work of Brandão and Svore (2017) [5] showcases the potential of quantum computing, by presenting a quantum algorithm for semidefinite programming. The promise of such a quantum speed-up for semidefinite programming could catalyze breakthroughs in the fields that heavily rely on SDP. Moreover, it opens up an exciting avenue for further research into the quantum algorithms for other types of optimization problems. This report aims to provide a comprehensive review of the quantum SDP solver introduced by Brandão and Svore [5]. We will delve into the details of the algorithm, elucidate its potential implications, discuss the conditions under which it outperforms classical counterparts, and explore possible future directions.

2 Semidefinite programs

Semidefinite programming (SDP) is a subclass of convex optimization, a field that, due to its tractability and broad applicability, has become a cornerstone of operational research and quantitative decision making. SDPs, in particular, are optimization problems with a linear objective function and constraints that are represented by linear matrix inequalities (LMIs).

In a standard form, a semidefinite program can be defined as follows:

$$\begin{aligned} \text{minimize } & \operatorname{tr}(CX) \\ \text{subject to } & \operatorname{tr}(A_i X) \leq b_i, \quad i = 1, \ldots, m \\ & X \succeq 0 \end{aligned}$$

where:

- C is an $n \times n$ hermitian matrix, the coefficients of our linear objective function.
- X is the $n \times n$ hermitian matrix of variables we want to solve for.
- A_i are $n \times n$ hermitian matrices for $i = 1, \ldots, m$.
- b_i are real numbers for $i = 1, \ldots, m$.

- tr(.) denotes the trace of a matrix.
- $X \succeq 0$ indicates that X is a positive semidefinite matrix, meaning all eigenvalues of X are nonnegative.

The dual form of an SDP provides additional insight into the problem and is defined as:

$$\begin{aligned} \text{minimize } & b^T y \\ \text{subject to } & \sum_{i=1}^{m} y_i A_i - C \succeq 0 \\ & y \geq 0 \end{aligned}$$

where:

- y is a vector in $\mathbb{R}^m$, and $b^T y$ is the linear objective function that we aim to minimize.
- y_i are the dual variables associated with the constraints of the primal SDP.
- The inequality $\sum_{i=1}^{m} y_i A_i - C \succeq 0$ signifies that the matrix $\sum_{i=1}^{m} y_i A_i - C$ must be greater than or equal to 0 in the positive semidefinite order.

The primal-dual pair of SDPs obeys weak and strong duality. The weak duality theorem ensures that the objective value of any feasible solution to the dual SDP provides an upper bound to the optimal value of the primal SDP. On the other hand, the strong duality theorem, under certain conditions, asserts that the optimal values of the primal and dual SDPs coincide. This duality plays a crucial role in the analysis and solution of semidefinite programs.

3 The Algorithm Analysis

This section aims to elucidate the foundational mechanisms underlying both the classical and quantum versions of the Arora-Kale algorithm for semidefinite programming (SDP). To grasp the quantum speed-up, it is crucial to first understand the classical Arora-Kale algorithm, which uses the Matrix Multiplicative Weights (MMW) method to construct an approximate solution to the SDP [6].

Further, we will analyze how the Arora-Kale algorithm maintains a feasible dual solution, which is a critical factor in its efficient performance [6]. Subsequently, we will explore the quantum version of this algorithm, which integrates quantum mechanics principles to achieve more efficient results [5].

The quantum version of the Arora-Kale algorithm leverages Gibbs sampling and a Hamiltonian oracle to expedite the computation process [5]. We will delve into these components and discuss how they contribute to the quantum speed-up for SDP. Through this analysis, we aim to reveal the commonalities and differences between the classical and quantum versions of the Arora-Kale algorithm, highlighting how quantum mechanics can significantly enhance computational efficiency in solving semidefinite programming problems.

3.1 Classical Arora-Kale Algorithm

The Arora-Kale algorithm [6] is a fast method for approximate semidefinite programming (SDP) that hinges on the Multiplicative Weights Update (MWU) method. It is known for its capability to provide an ϵ-approximate solution to the SDP in $O(\frac{n^2}{\epsilon^2}\log n)$ iterations, where n is the size of the SDP, thereby forming a crucial foundation for many practical optimization problems.

The algorithm starts with a uniform distribution over the constraints using I where $\mathrm{Tr}(\mathrm{X_{opt}}) \leq R$ and $\sum_{\mathrm{i}}(\mathrm{y_{opt}}) \leq r$ are upper bounds to the size of the optimal primal and dual solutions. The algorithm runs for maximum time T over each round and updates the weights based on the received cost vector y^t which satisfies the constraint. This updated weight distribution is used to generate a new feasible solution ρ to the SDP's. Eventually, the algorithm outputs the average of all the computed solutions.

```
Algorithm 1: Arora-Kale Algorithm for SDP's
Set $\rho^{(1)} = \frac{I}{n}$. $Let \varepsilon = \frac{\delta\alpha}{2R^2}$ $and$ $\varepsilon' = -\ln(1-\varepsilon)$. $Let$ $T = \frac{8R^2\ln(n)}{\delta^2}$
$For$ $t = 1, ..., T$
1.      $Run$ $ORACLE(\rho^t), if$ $fails$ $stop,$ $output$ $\rho^{(t)},$ $else$ $output$ $y^t$
2.      $Let$ $M^t = \frac{\sum_{j=1}^{m} y_j^t A_j - C + RI}{2R}$
3.      $Compute$ $W^{t+1} = \exp\left(-\varepsilon' \sum_{\tau=1}^{t} M^\tau\right)$
4.      $Set$ $\rho^{(t+1)} = \frac{W^{(t+1)}}{\mathrm{tr}(W^{(t+1)})}$
$Output:$ $\overline{y} = \left(\frac{\delta\alpha}{R}\right)\mathbf{e}_1 + \frac{1}{T}\sum_{t=1}^{T}\mathbf{y}^t$
```

Here it is important to understand Jaynes' principle [7] that states when we have incomplete information about a system, we should choose the probability distribution that maximizes entropy, subject to the constraints we do know. When this is applied to the primal form we get solution:

$$\mathbf{X} = exp(\sum_j \lambda_j A_j + \lambda_0 C)$$

When X does become optimum, we can get $\frac{X_{opt}}{\mathrm{tr}(\mathrm{X_{opt}})}$ which is nothing but Gibbs state with same expectation value for A_j, C. This can be used to define Hamiltonian as $\rho = \frac{exp(-H)}{\mathrm{tr(exp(-H))}}$, where H can be seen as expectation values for A_j, C.

Looking at the algorithm above we can see that there is a game between step 1, of ORACLE(ρ) and Hamiltonian in step 4, which in turn depends on step 2 M^t, both players try to get best objective value, one for primal and another player for dual problem. But the optimum is reached when the game becomes zero-sum. In the subsequent sections, we will dissect the Matrix Multiplicative Weight method and the feasible dual solution construction, both of which form the pillars of the Arora-Kale algorithm.

3.1.1 Matrix Multiplicative Weight Method

The Matrix Multiplicative Weights (MMW) method is a generalization of the classic Multiplicative Weights Update (MWU) method to the setting of semidefinite programming [6]. The MWU method is a fundamental technique in theoretical computer science, used for constructing approximate solutions to a variety of optimization problems. In the context of semidefinite programming, the MMW method forms the core of the Arora-Kale algorithm.

The MMW method works by maintaining a distribution over the constraints of the SDP. At each iteration, the method updates this distribution based on the discrepancy between the current solution and the constraint matrices. The updated distribution is then used to generate a new feasible solution to the SDP.

The Oracle in this context refers to a routine that computes the discrepancy between the current solution and the constraint matrices. Formally, given a solution $\rho^{(t)}$ and a distribution $M^{(t)}$, the Oracle computes the matrix $\sum_{i=1}^{m} M_i^{(t)}(A_i - \rho^{(t)})$, which measures the discrepancy between the current solution and the constraint matrices.

Now let's show that based on this explanation for any loss matrix M, there will be density function ρ of some form.

Theorem 3.1. *For any sequence of loss matrices $M^{(1)}, M^{(2)}, \ldots, M^{(T)}$, the matrix MW algorithm generates density matrices $\rho^{(1)}, \rho^{(2)}, \ldots, \rho^{(T)}$ such that*

$$\sum_{t=1}^{T} \mathbf{M}^t \cdot \rho^t \leq (\varepsilon + 1)\lambda_n \left(\sum_{t=1}^{T} \mathbf{M}^t \right) + \frac{\ln n}{\varepsilon}$$

Remark. The candidate solution $\mathbf{X^t}$ is just $N\rho^t$, where ρ^t is the density matrix at t^{th} round.

Proof. The proof is based on the potential function. We track the change in $Tr(W^(t)$ over time. The anlaysis is complicated by the fact that matrix multiplication is non-commutative, so $exp(\mathbf{A} + \mathbf{B}) \neq exp(\mathbf{A})exp(\mathbf{B})$ in general. However, Golden-Thomson inequality can be used [8, 9]: $\mathrm{Tr}(\mathrm{ex}^{\mathrm{A+B}} \leq \mathrm{Tr}(\mathrm{e}^{\mathrm{A}}\mathrm{e}^{\mathrm{B}})$

$$\begin{aligned}
\mathrm{Tr}(W^{(t+1)}) &= \mathrm{Tr}(\exp\left(-\varepsilon' \sum_{\tau=1}^{t-1} \mathbf{M}^{\tau}\right)) \\
&\leq \mathrm{Tr}(\exp\left(-\varepsilon' \sum_{\tau=1}^{t-1} \mathbf{M}^{\tau}\right) \exp\left(-\varepsilon' \mathbf{M}^{(t)}\right)) \\
&= \mathbf{W}^{(t)} \cdot \exp\left(-\varepsilon' \mathbf{M}^{(t)}\right) \\
&\leq \mathbf{W}^{(t)} \cdot (\mathbf{I} - \varepsilon' \mathbf{M}^{(t)}) \\
&= \mathrm{Tr}(\mathbf{W}^{(t)} \cdot (1 - \varepsilon' \mathbf{M}^{(t)} \cdot \mathbf{P}^{(t)})) \\
&\leq \mathrm{Tr}(\mathbf{W}^{(t)} \cdot \exp\left(\varepsilon' \mathbf{M}^{(t)} \cdot \mathbf{P}^{(t)}\right))
\end{aligned}$$

By induction, since $\mathrm{Tr}(\mathbf{W}^{(1)}) = \mathrm{Tr}(\mathbf{I}) = n$, we get

$$\mathrm{Tr}(\mathbf{W}^{(T+1)}) \leq n \ \exp\left(-\varepsilon' \sum_{t=1}^{T} \mathbf{M}^{(t)} \cdot \rho^{(t)}\right)$$

On the other hand, we have:

$$\begin{aligned}
\mathrm{Tr}(\mathbf{W}^{(T+1)}) &= \mathrm{Tr}(\exp\left(-\varepsilon' \sum_{t=1}^{T} \mathbf{M}^{(t)}\right)) \\
&\geq \exp\left(-\varepsilon' \lambda_n \sum_{t=1}^{T} \mathbf{M}^{(t)}\right)
\end{aligned}$$

This gives us:

$$\exp\left(-\varepsilon' \lambda_n \sum_{t=1}^{T} \mathbf{M}^{(t)}\right) \leq n \ \exp\left(-\varepsilon' \sum_{t=1}^{T} \mathbf{M}^{(t)} \cdot \rho^{(t)}\right)$$

Taking logarithms and simplifying, we get the required inequality:

$$\sum_{t=1}^{T} \mathbf{M}^{t} \cdot \rho^{t} \leq (\varepsilon + 1) \lambda_n \left(\sum_{t=1}^{T} \mathbf{M}^{t}\right) + \frac{\ln n}{\varepsilon}$$

$\square$

3.1.2 Feasible Dual Solution

A key aspect of the Arora-Kale algorithm is its ability to maintain a feasible dual solution throughout its iterations. This property is instrumental in ensuring that the algorithm remains efficient and effective. In the context of semidefinite programming, the dual of the standard SDP problem is expressed as follows:

$$\begin{aligned} \min \quad & b \cdot y & (1)\\ \text{s.t.} \quad & \textstyle\sum_{j=1}^{m} y_j A_j \geq C & (2)\\ & y \geq 0 & (3) \end{aligned}$$

The feasible dual solution is critical for the performance guarantee of the Arora-Kale algorithm. It allows the algorithm to provide an ϵ-approximate solution to the original SDP problem in $O\left(\frac{m^2 \log m}{\epsilon^2}\right)$ iterations. This is a significant improvement over general-purpose semidefinite programming solvers, particularly for large-scale problems.

Here we need to define the ORACLE(ρ) mentioned in the algorithm. Which takes in ρ to give y each round. And the ORACLE is defined as:

$$\mathcal{D}_\alpha := \{\mathbf{y} \in \mathbb{R}^m : \mathbf{y} \geq 0, \mathbf{b} \cdot \mathbf{y} < \alpha\}$$

such that.

$$\sum_{j=1}^{m} y_j \operatorname{tr}(A_j \rho) \geq \operatorname{tr}(C\rho)$$

Theorem 3.2. *Suppose the ORACLE never fails for $T = \frac{16R^4 \ln(n)}{\alpha^2 \delta^2}$ iterations. Then $\overline{y} = \left(\frac{\delta\alpha}{R}\right) \mathbf{e}_1 + \frac{1}{T} \sum_{t=1}^{T} \mathbf{y}^t$ is dual feasible with objective value at most $\alpha(1+\delta)$ with δ being the error in current objective to optimum objective value.*

Proof. This is a simple corollary of Theorem 3.1. And by the definition of ORACLE and the fact that $A_1 = I, b_1 = R$, we have

$$\overline{y}.b = \delta\alpha + \frac{1}{T} \sum_{t=1}^{T} \mathbf{y}^t.b \leq \alpha(1+\delta)$$

How is $\overline{y}$ dual feasible? It comes from the property of the ORACLE that $\overline{y} \geq 0$ □

3.2 Quantum Arora-Kale Algorithm

Once the classical algorithm and the trick it used are clear, the quantum version differs mainly in two aspects. 1. How to create the ORACLE which outputs $y^{(t)}$, 2. How to come up with a quantum method to calculate step 4.

```
Algorithm 2: Quantum Arora-Kale Algorithm for SDP's
Set $\rho^{(1)} = \frac{I}{n}$. Let $\varepsilon = \frac{\delta\alpha}{2R^2}$ and $\varepsilon' = -\ln(1-\varepsilon)$. Let $T = \frac{8R^2\ln(n)}{\delta^2}$
For $t = 1, ..., T$
1.        Run $ORACLE(\rho^t)$ by Gibbs sampler
2.        Let $M^t = \frac{\sum_{j=1}^{m} y_j^t A_j - C + RI}{2R}$ and sparsify
3.        Compute $W^{t+1} = \exp\left(-\varepsilon' \sum_{\tau=1}^{t} M^\tau\right)$
4.        Set $\rho^{(t+1)} = \frac{W^{(t+1)}}{\mathrm{tr}(W^{(t+1)})}$
Output : $\overline{y} = \left(\frac{\delta\alpha}{R}\right)\mathbf{e}_1 + \frac{1}{T}\sum_{t=1}^{T}\mathbf{y}^t$
```

3.2.1 Gibbs Sampling

Here the paper implements the ORACLE auxiliary algorithm by use of Jaynes' principle mentioned earlier. The output y^t is a Gibbss probability distribution over the two constraints, i.e.,

$$q_{\rho,\lambda,\mu}(i) := \frac{exp(\lambda tr(A_i\rho) + \mu\sum_i b_i)}{\sum_i exp(\lambda tr(A_i\rho) + \mu\sum_i b_i)'}$$

for real numbers λ, μ

```
Algorithm 3: Quantum ORACLE($\rho$)
Samples from distribution $\overline{q}_{\rho,k}$ such that $\|\overline{q}_{\rho,k} - q_{\rho,k}\| \leq \mu$ and real numbers $e_{i i=1 m+1}$
such that $|e_i - tr(A_i\rho)| \leq \mu, k > 0$
Let $M = 80log^{1+\zeta}(\frac{8R^2nm}{\varepsilon})$
1.        Sample $i_1, ..., i_M \in [m]^M$ independently from the distributions $\overline{q}_{\rho,k}$
2.        If $\frac{1}{M}\sum_{j=1}^{M} e_{i_j} \geq \frac{e_{(m+1)}}{kN} - (k+\mu)$ and $\frac{1}{M}\sum_{j=1}^{M} b_{i_j} \leq \frac{\alpha}{kN} + R(k+\mu)$,
          output samples from $\overline{q}_{\rho,k}$ for $\frac{y}{\|y\|_1}$ and the number $kN$ for $\|y\|_1$
Output: Samples from distribution $\frac{y}{\|y\|_1}$, and values of $\|y\|_1$
```

3.2.2 Hamiltonian Oracle

Given a Hamiltonian H, in order to run a Gibbs sampler algorithm, we need an oracle for its entries. We also need the notion of a probabilistic oracle. This is an oracle that with high probability outputs the right entry of the Hamiltonian, but with small probability might output a wrong value. Consider a quantum state ρ and two real numbers λ and μ. The Hamiltonian is defined as:

$h(\rho, \lambda, \mu) := \sum_{i=1}^{m} r_i |i\rangle\langle i|$, with $r_i := \lambda\,\mathrm{tr}(A_i\rho) + \mu b_i$, So let's prove that the Hamiltonian along with phase estimation can give the right entry with high probability and small error.

Theorem 3.3. *Given a s-sparse $n \times n$ Hermitian matrix A with $\|A\| \leq 1$ and the density matrix ρ, with probability larger than $1-p_e$, one can compute* $\mathrm{tr}(\rho \mathrm{A})$ *with additive error ε in time $\mathcal{O}(s\varepsilon^{-2}\log\left(\frac{ns}{p_e\varepsilon}\right)^4)$ using $\mathcal{O}(\log\left(\frac{1}{p_e}\right)\varepsilon^{-2})$*

Proof. Using the Hamiltonian simulation technique mentioned in the, and phase estimation algorithm, the time taken is $(s\log(\frac{ns}{\varepsilon}))$ to measure to accuracy $\frac{\varepsilon}{2}$ the energy of A_i in the state ρ. Now using the Chernoff bound, repeating the process $\mathcal{O}(\frac{1}{\varepsilon^2})$ times allows us to obtain an estimation for $\mathbf{tr}(\rho\mathbf{A_i})$ to accuracy ε □

Now we have both Gibbs sampler and Hamiltonian simulation, this can be used to get a full Quantum Arora-Kale Algorithm

3.3 Quantum SDP time Complexity

Let's combine the Gibbs sampler with sparsity for fast quantum sparse access and Hamiltonian simulation to find the basic time complexity $\tilde{O}(n^{1/2}m^{1/2}s^2 poly(R,r,\delta))$.

1. Implementing the Gibbs sampler to get y^t and using amplitude amplification to boost the probability of the y^t which satisfies the constraints to get solve time $\tilde{O}(s^2n^{\frac{1}{2}}m^{\frac{1}{2}})$
   (a) There is a feasible y^t of the form $y^t = Nq^t$ with $q^t := \frac{exp(h)}{tr(exp(h))}$
   (b) To simulate an oracle to the entries of h. One can do it by measuring ρ^t with A_i
   (c) To prepare each ρ^t takes time $\tilde{O}(s^2n^{\frac{1}{2}})$.
   (d) To sample from q^t requires $\tilde{O}(m^{\frac{1}{2}})$ calls to Oracle for h. So total time is $O(s^2n^{\frac{1}{2}}m^{\frac{1}{2}})$
2. Sparsify M^t to be a sum of $\tilde{O}(log(m))$ terms, shown by using Lemma 15 in [5]
3. Using the Hamiltonian along with amplitude amplification to prepare: $\overline{\rho}^t = \frac{\exp\left(-\varepsilon' \sum_{\tau=1}^{t} M^\tau\right)}{tr(\exp\left(-\varepsilon' \sum_{\tau=1}^{t} M^\tau\right))}$ in time $\tilde{O}(s^2n^{\frac{1}{2}})$.

Theorem 3.4. *The Quantum SDP algorithm runs in time $\tilde{O}(n^{1/2}m^{1/2}s^2R^{3/2}/\delta^{18})$.*

Proof. Method to prepare an ε-approximation to $e^{\beta H}/\mathrm{tr}(e^{\beta H})$ for a sparse Hamiltonian H with $H \leq 1$ using $\tilde{O}(\dim(H)\beta s/\varepsilon)$ calls to the oracle and two-qubit gates. This Gibbs sampler is to step 1 of the quantum algorithm.

In order to estimate the running time of each of these applications, One should estimate the associated β and sparsity. In step 2, the associated β is upper bounded by $O(1/\varepsilon) = O(R^2/\delta)$ and the sparsity is one. In step 3, the associated β is upper bounded by $\varepsilon' T = \tilde{O}(R/\delta)$, while the sparsity is upper bounded by TQ times the sparsity of each of the input matrices, which gives $\tilde{O}(sR^9/\delta^6)$.

Finally, since to query an element of the Hamiltonian algorithm need to query each of the A_i matrices appearing in the decomposition, we have a cost of Ts for querying an element of the Hamiltonian. Therefore, we have the bounds $G_h \leq \tilde{O}(\sqrt{m}R^2/\delta)$ and $G_M \leq \tilde{O}(\sqrt{n}s^2R^9/\delta^6)$.

Giving total time $\tilde{O}(n^{1/2}m^{1/2}s^2R^{3/2}/\delta^{18})$ by using Theorem 4 from [5]. □

4 Lower Bounds

We discuss lower bounds for the SDP problem in two scenarios. In the first scenario, we assume that the sizes of the primal and dual solutions are bounded by R and r respectively. We also assume that the approximation error ϵ is also a constant. We use the adversary method to provide a lower bound for the problem in section 4.1. Then, we move on to discuss the case where the above parameters are not constant in section 4.2. Finally, we compare the two lower bounds and discuss the implications of these lower bounds.

4.1 Lower Bounds: Constant Case

As mentioned above, we consider the case where the sizes of primal and dual solutions are bounded by a constant and we also assume a constant approximation error. First, we prove a lower bound on the following problem

Theorem 4.1. *Given a bit string of length n and a promise that exactly one of the bits are set (hamming weight of 1), identifying the index of the set bit requires at least $O(\sqrt{N})$ queries.*

Remark. Notice that this problem is different from the OR problem where **at most** one bit is set. Moreover, this is a search problem and not a decision problem.

Proof. We attempt the proof using the adversarial method discussed in the class. Recall the quantum circuit setup shown in class in figure 1. Here, instead of distinguishing between the zero and one solution case in the original OR problem, we attempt to distinguish between two separate bit strings that have ones in different positions of the string. Using arguments similar to the one in notes, we require the statistical distance between the final qubit states after application of the quantum circuit corresponding to the two different bit strings to be at least $1 - 2\epsilon$, where ϵ is the tolerable error. In other words, we require that

$$1 - 2\epsilon \leq \left\| \left|\psi_{x_1}^{final}\right\rangle - \left|\psi_{x_2}^{final}\right\rangle \right\|_2$$

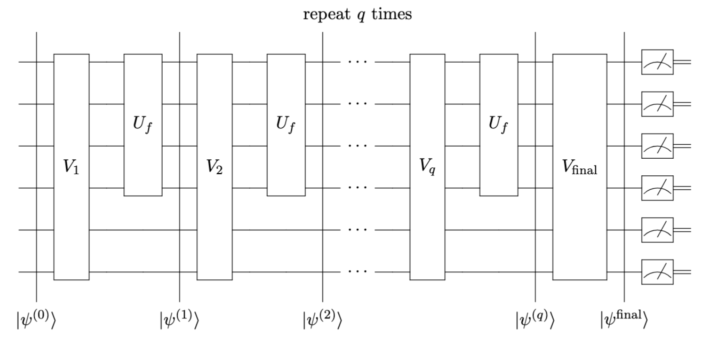

Figure 1: Model of a blackbox algorithm with q queries

Now, we figure out how much difference two distinct binary strings of hamming weight 1, can cause a difference in the i^{th} query. The i^{th} query transforms the quantum states in the following way

$$\left|\psi_{x_1}^{(i)}\right\rangle = (U_{\delta_{x_1}} \otimes I)V_i \left|\psi_{x_1}^{(i-1)}\right\rangle$$
$$\left|\psi_{x_2}^{(i)}\right\rangle = (U_{\delta_{x_1}} \otimes I)V_i \left|\psi_{x_2}^{(i-1)}\right\rangle$$

Therefore,

$$\begin{aligned}
\left\|\left|\psi_{x_1}^{(i)}\right\rangle - \left|\psi_{x_2}^{(i)}\right\rangle\right\|_2 &= \left\|(U_{\delta_{x_1}} \otimes I)V_i \left|\psi_{x_1}^{(i-1)}\right\rangle - (U_{\delta_{x_2}} \otimes I)V_i \left|\psi_{x_2}^{(i-1)}\right\rangle\right\|_2 \\
&= \left\|(U_{\delta_{x_1}} \otimes I)V_i \left|\psi_{x_1}^{(i-1)}\right\rangle - (U_{\delta_{x_2}} \otimes I)V_i \left|\psi_{x_1}^{(i-1)}\right\rangle\right\|_2 \\
&+ \left\|(U_{\delta_{x_2}} \otimes I)V_i \left|\psi_{x_1}^{(i-1)}\right\rangle - (U_{\delta_{x_2}} \otimes I)V_i \left|\psi_{x_2}^{(i-1)}\right\rangle\right\|_2 \\
&= \Delta + \left\|\left|\psi_{x_1}^{(i-1)}\right\rangle - \left|\psi_{x_2}^{(i-1)}\right\rangle\right\|_2
\end{aligned}$$

Hence, the 2-norm difference can only increase in Δ and we can bound Δ^2 using methods similar to those employed in class. We express the superposition as $V_i \left|\psi_{x_1}^{(i)}\right\rangle = \sum_z \alpha_i |z\rangle$. Unlike the OR problem, the value of Δ^2 now has two terms each corresponding to respective x_js. To elaborate,

$$\begin{aligned}\Delta^2 &= \left\| (U_{\delta_{x_1}} \otimes I)V_i \left|\psi_{x_1}^{(i-1)}\right\rangle - (U_{\delta_{x_2}} \otimes I)V_i \left|\psi_{x_1}^{(i-1)}\right\rangle \right\|_2^2 \\ &= \left\| ((U_{\delta_{x_1}} \otimes I) - (U_{\delta_{x_2}} \otimes I)) \sum_i \alpha_z \left|z\right\rangle \right\|_2^2 \\ &= \sum_{bu} \left|\alpha_{x_1bu} - \alpha_{x_1\bar{b}u}\right|^2 + \sum_{bu} \left|\alpha_{x_2bu} - \alpha_{x_2\bar{b}u}\right|^2 \\ &= 2(\sum_u \left|\alpha_{x_10u} - \alpha_{x_11u}\right|^2 + \sum_u \left|\alpha_{x_20u} - \alpha_{x_21u}\right|^2) \\ &\leq 4(\sum_u (\left|\alpha_{x_10u}\right|^2 + \left|\alpha_{x_10u}\right|^2) + \sum_u (\left|\alpha_{x_20u}\right|^2 + \left|\alpha_{x_20u}\right|^2)) \\ &= 4(\Pr\left[i^{th} \text{ query at } f_{x_1} \text{ is } x_1\right] + \Pr\left[i^{th} \text{ query at } f_{x_1} \text{ is } x_2\right])\end{aligned}$$

Adding the differences across all queries, we have

$$\left\|\left|\psi_{x_1}^{final}\right\rangle - \left|\psi_{x_2}^{final}\right\rangle\right\|_2 \leq 2\sum_{i=1}^{q} \sqrt{\Pr[i^{th} \text{ query at } f_{x_1} \text{ is } x_1] + \Pr[i^{th} \text{ query at } f_{x_1} \text{ is } x_2]}$$

We sum this result across all possible combinations of x_1 and x_2.
We now have,

$$\begin{aligned}\sum_{x_1}\sum_{x_2}(1-2\epsilon) &\leq 2\sum_{x_1}\sum_{x_2}\sum_{i=1}^{q} \sqrt{\Pr[i^{th} \text{ query at } f_{x_1} \text{ is } x_1] + \Pr[i^{th} \text{ query at } f_{x_1} \text{ is } x_2]} \\ &= 2\sum_{i=1}^{q}\sum_{x_1}\sum_{x_2} \sqrt{\Pr[i^{th} \text{ query at } f_{x_1} \text{ is } x_1] + \Pr[i^{th} \text{ query at } f_{x_1} \text{ is } x_2]} \\ &\leq 2\sum_{i=1}^{q} \sqrt{N^2}\sqrt{(2N+1) + (2N+1)} \\ &= 2qN\sqrt{4N+2}\end{aligned}$$

$$\implies N^2(1-2\epsilon) \leq 2qN\sqrt{4N+2}$$

$$\implies \frac{N}{\sqrt{4N+2}}(\frac{1}{2} - \epsilon) \leq q$$

The first line follows from the distance needing to be large enough to distinguish between the states. The second line follows from a simple change in order of sums. The third line follows from an application of Cauchy-Swartz inequality. The rest is rearranging terms. Hence, we require at least $O(\sqrt{N})$ even for the known weight Quantum Search Problem.

$\square$

Remark. Can we use other methods discussed in the lecture to arrive at a similar result? Observe that the General Adversary Method cannot be used here since this is a search problem and not a decision problem. Perhaps, a reformulation as a decision problem can help? It is also not clear to us if we can apply the polynomial method to this problem since unlike the OR problem, we do not have the symmetricity here. To elaborate, we are already under the promise that we have a hamming weight of one and we want the exact index of the set bit. Permutations of input most definitely change the output of the function we are considering here. Using the polynomial method may require ingenuity and a more rigorous analysis both of which are beyond our reach :P. So, we leave this as an open problem.

Now that we have proven the lower bound on the known initial weight Quantum Search Problem, let's solve the problem using a semidefinite program to establish a lower bound on the SDP problem. This result was already proved by Brandao and Svore in their seminal paper [5]. Here, we present a proof that is more inline with concepts discussed in class.

Theorem 4.2. *Solving SDPs requires at least $O(\sqrt{n}+\sqrt{m})$ queries to solve in the case where the sizes of primal and dual solutions are bounded by a constant when we can tolerate a constant error approximation in the optimal value. Here, $n \times n$ is the size of constraint and cost matrices and m is the number of constraints.*

Proof. The proof outline is as follows. We consider two functions f_1 and f_2 each corresponding to n and m bit strings where each bit corresponds to the value of the respective function. Each of these strings have a hamming weight one. In other words, inverse of 1 has a single element under both functions. The problem of finding the set bits of f_1 and f_2 is equivalent to solving the known initial weight Quantum Search Problem defined above. Hence, the lower bound of the said problem is $O(\sqrt{n}+\sqrt{m})$.

We now solve this problem by defining an SDP such that the size of the primal and dual solutions can be at most size 1. We also specify the bound on constraints as $b_j = 1, \forall j \in [m]$. Moreover, we construct the matrices C and A such that $C_{ii} = 1, 0$ otherwise and $(A_j)_{ii} = 1, 0$ otherwise. In other words, the cost matrix is a diagonal matrix with only the i^{th} element set. And the constraint matrices are all zero, except that required by the constraint on size of the solution which is identity and the j^{th} matrix is the same as the cost matrix.

Hence, the primal problem turns into

$$\begin{aligned}
&\max\ x_{ii} \\
&\sum_{k=1}^{n} x_{kk} \leq 1 \\
&x_{ii} \leq 1
\end{aligned}$$

This problem is clearly maximized when $x = |i\rangle\langle i|$.

Coming to the dual solution, it reduces to

$$\begin{aligned}
&\min y_j \\
&y_1 \geq 0 \\
&y_1 + y_j \geq 1
\end{aligned}$$

The solution of this problem is clearly $y_j = 1$. Thus we know the index i and index j from sampling the primal and dual solutions thus solving the two-function quantum search problem.

□

Remark. This version of SDP reduces to an LP problem, so the lower bound holds for linear programming as well. Additionally, the lower bound is tight and has been achieved by the more recent algorithms.

Remark. The lower bound in the classical case is $O(n + m)$ because of how unordered search works in the classical setting. While we cannot even store the matrices in this time in the classical setting, we are unsure whether this bound can be tightened in the sparse setting. So, we treat the tightness of the classical lower bound as an open problem still.

Finally, we assume in the above proof that we can efficiently construct the cost and constraint matrices for the problem we defined. Let's see how we can do that by using a similar approach to our Q4 problem in HW2. Cost matrix is a diagonal matrix whose diagonal is the bit string represented by the function f_1. To get the block encoding for the cost matrix, we first need to construct a sparse access for the cost matrix. Since, the matrix is diagonal, each row and column has a single element at index i. This implies that the sparsity $s = 1$ and,

$$\begin{aligned}
O_{row} &: |i\rangle|0\rangle \longrightarrow |i\rangle|i\rangle \\
O_{col} &: |0\rangle|j\rangle \longrightarrow |j\rangle|j\rangle \\
O_C &: |i\rangle|i\rangle|x\rangle \longrightarrow |i\rangle|i\rangle|x \oplus f_1(i)\rangle \\
O_C &: |i\rangle|j\rangle|x\rangle \longrightarrow |i\rangle|j\rangle|x\rangle, \forall j \neq i
\end{aligned}$$

We can also obtain sparse access to constraint matrices with a bit more work. We can think of the set of constraint matrices as a list of 2 dimensional matrices. The diagonal of the j^{th} matrix is given by the scalar multiple of the bit string f_1 multiplied $f_2(j)$. We can extend the notion of sparse matrices to 3 dimensions with the first index denoting the constraint index j. We still have sparsity $s = 1$, the row and columns accesses are same as before, and the matrix access operator becomes,

$$O_A : |j\rangle\,|i\rangle\,|i\rangle\,|x\rangle \longrightarrow |i\rangle\,|i\rangle\,|x \oplus (f_1(i) * f_2(j))\rangle$$
$$O_A : |j\rangle\,|i\rangle\,|k\rangle\,|x\rangle \longrightarrow |i\rangle\,|k\rangle\,|x\rangle\,, \forall k \neq i$$

Thus, we can create the required block encodings from the given sparse accesses to the matrices using the Gram matrix trick mentioned in the class.

4.1.1 Other Interesting Questions & Remarks

1. The proof for the lower bound requires access to primal and dual solutions as output. Do we have a different lower bound when we simply have to compute the optimal value of the problem

2. The current algorithms only provide a Quantum Advantage over classical algorithms and that too not in all cases. Can we achieve Quantum supremacy in very specific cases? What would their applications look like?

3. How does noise in the input matrices affect the stability of our solution? How does the stability of classical algorithms compare to the stability of the quantum approach?

4.2 Lower Bounds: Generic Case

Now that we have discussed the lower bounds for a special case where the solutions sizes are bounded, let us now discuss the general case where the solutions are unbounded and we are looking for arbitrary precision in the optimal value. Apeldoorn and Gilyen [10] have proved lower bounds for the general case and the results are not promising. Here, we provide the lower bound and a brief outline for the proof while emphasizing on the parts relevant to the class.

Theorem 4.3. *It takes at least $\Omega(\sqrt{max\{n,m\}^{\frac{1}{2}}\,min\{n,m\}^{\frac{3}{2}}})$ blackbox quantum queries to compute the optimal value of an LP problem in the hard case.*

$$\begin{aligned}
\mathrm{OPT} = \max \quad & \sum_{k=1}^{c} w_k \\
\text{s.t.} \quad & \begin{bmatrix} I & -Z^1 & \cdots & -Z^a \\ 0 & \mathbf{1}^T & & \\ & & & \\ 0 & & \ddots & \\ 0 & & & \mathbf{1}^T \end{bmatrix} \begin{bmatrix} w \\ v^{(1)} \\ \vdots \\ v^{(a)} \end{bmatrix} \leq \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \\
& v^1, \ldots, v^a \in \mathbb{R}^b_+, w \in \mathbb{R}^c_+.
\end{aligned}$$

Figure 2: Hard LP problem

The high level approach here is to construct an LP problem whose optimal value reduces to a simpler composite bit function under a promise. This equivalency allows us to easily obtain a lower bound on the generic LP problem as a whole. In fact the optimal value for the LP problem shown in the figure 2 has the optimal value of

$$\sum_{i=0}^{a} \max_{j\in[b]} \sum_{l=1}^{c} Z_{ijl} \tag{4}$$

where $Z \in \{0,1\}^{a\times b\times c}$. The size of the problem depends on a, b, c. We skip the proof since it is not relevant to the course.

$$\begin{aligned}
&MAJ_a(\\
&\qquad OR_b(MAJ_c(Z_{111},\ldots,Z_{11c}),\ldots,MAJ_c(Z_{1b1},\ldots,Z_{1bc})),\\
&\qquad \ldots,\\
&\qquad OR_b(MAJ_c(Z_{a11},\ldots,Z_{a1c}),\ldots,MAJ_c(Z_{ab1},\ldots,Z_{abc}))\\
&\quad)
\end{aligned}$$

Figure 3: The Majority-Or-Majority Problem

The next step is to prove that the value given by the expression 4 is equivalent to the composite function in the figure 3 when the number of 0s and 1s are roughly equal in each of the majority functions. Proving this is relatively straightforward and hence we skip the proof here.

The interesting part for the course is figuring out the lower bound for the said composition of MAJ-OR-MAJ function. The lower bound of the composition function is the product of the lower bound of its individual functions as demonstrated by [11]. We are already aware that the OR function has a lower bound of $O(\sqrt{N})$. Now, we want the lower bound on the majority function under the promise that there are roughly equal number of 0s and 1s.

Lemma 4.4. *Under the promise that there are roughly equal number of* 0*s and* 1*s, the MAJORITY function (mode) takes at least* $O(N)$ *quantum queries to the blackbox.*

Proof. We attempt to prove this using the General Adversary Method discussed in class. Assume that N is odd, so that there is no ambiguity in the definition of a MAJORITY. The inverse of 0 is S_0, the set of all the bit strings having $\frac{N}{2}$ ones and the inverse of 1 is S_1 the set of all bit strings having $\frac{N}{2}+1$ ones. We build an edge between elements of S_0 and S_1 iff they differ in exactly one bit. There are roughly $\frac{N}{2}$ edges from each element in one set to another. This means that both the left and right degrees are $O(N)$. The general adversary method also requires the change in hamming weight which by construction is 1. Hence, using the formula $O(\sqrt{\frac{d_{left}d_{right}}{c_{left}c_{right}}}) = O(\sqrt{\frac{N*N}{1*1}}) = O(N)$.

□

Combining this fact with multiplicative nature of the composition functions, the lower bound for the composite function is $O(a\sqrt{b}c)$. By expressing a, b, c in terms of the size of the problem, we get the claimed lower bound.

4.2.1 Other Interesting Questions & Remarks

1. Unlike the quantum search case, the MAJORITY function is invariant under permutations of the input. Henceforth, I wouldn't be surprised if we can prove this bound using the polynomial method as well.
2. The lower bound we get in this case is much worse than the special case. As a consequence, we would ideally like to find applications for the special case. Most of the classical applications for the SDP problem do not have sparse solutions and hence finding good applications is a research problem in its own right.
3. It would be interesting to have an explicit lower bound on the size of the solution in the generic case.
4. The proof for the lower bound requires access to primal and dual solutions as output. Do we have a different lower bound when we simply have to compute the optimal value of the problem
5. The proof uses the fact that we are solving LP problems. Is there an even better lower bound for the SDP problem?

5 Improvements

While the Quantum Arora-Kale algorithm has shown promising speed-ups over its classical counterpart [5], several potential improvements could further enhance its performance and usability. Firstly, a more efficient construction of the Hamiltonian oracle could increase the overall speed of the algorithm. Current implementations are complex and require numerous quantum gates, leading to increased potential for error [4].

Another promising avenue for improvement lies in the fusion of quantum computing techniques with interior-point methods (IPMs) [12], a robust and popular approach for solving SDPs in the classical realm [13]. In classical computation, IPMs have been shown to solve SDPs efficiently in polynomial time. However, these methods are inherently sequential, making them a challenging candidate for direct quantum speedup. The central path-following nature of IPMs means that each step depends on the previous one, hence reducing the potential for parallelization, a feature quantum algorithms often exploit for speedup [14].

Nevertheless, there may exist quantum algorithmic techniques that could be harnessed to accelerate specific parts of IPMs. For instance, linear system solvers or matrix inversion subroutines, which are computationally expensive parts of IPMs, can potentially be sped up using quantum linear system algorithms like HHL (Harrow-Hassidim-Lloyd) [14]. Such hybrid methods could provide an interesting direction for future research, potentially outperforming the current Quantum Arora-Kale algorithm for certain types of SDPs.

The algorithm could benefit from improvements in quantum hardware. As of now, the quantum computing devices capable of running this algorithm are limited by noise and relatively small qubit counts [15]. With advancements in quantum error correction and coherence time, the practical implementation of the Quantum Arora-Kale algorithm can become more feasible [4].

Lastly, the algorithm could be extended to handle a wider variety of problems. Currently, the algorithm is optimized for semidefinite programs where the matrices A_j and C are sparse and have low rank, which is not always the case in real-world problems [5]. For example, expanding the algorithm to handle SDPs with dense matrices, high-rank matrices, or those derived from large-scale control systems and robust optimization could significantly increase its applicability [**?**]. Similarly, integrating the algorithm into a hybrid quantum-classical framework could allow it to tackle mixed-integer semidefinite programs (MISDPs), which are notoriously difficult to solve and occur frequently in areas such as scheduling and logistics [16].

References

[1] L. Vandenberghe and S. Boyd, “Semidefinite programming,” *SIAM review*, vol. 38, no. 1, pp. 49–95, 1996.

[2] S. Boyd and L. Vandenberghe, “Convex optimization.” Cambridge University Press, 2004.

[3] ——, *Convex Optimization.* Cambridge university press, 2004.

[4] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information.* Cambridge university press, 2000.

[5] F. Brandao and K. Svore, “Quantum speed-ups for solving semidefinite programs,” 10 2017, pp. 415–426.

[6] S. Arora, E. Hazan, and S. Kale, “Fast algorithms for approximate semidefinite programming using the multiplicative weights update method,” in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS’05)*, 2005, pp. 339–348.

[7] E. T. Jaynes, “Information theory and statistical mechanics. ii,” *Physical Review*, Oct 1957.

[8] S. Golden, “Lower bounds for the helmholtz function,” *Physical Review*, Feb 1965.

[9] C. J. Thompson, “Inequality with applications in statistical mechanics,” *Journal of Mathematical Physics*, vol. 6, pp. 1812–1813, 1965.

[10] J. van Apeldoorn, A. Gilyén, S. Gribling, and R. de Wolf, “Quantum SDP-solvers: Better upper and lower bounds,” *Quantum*, vol. 4, p. 230, feb 2020. [Online]. Available: https://doi.org/10.22331%2Fq-2020-02-14-230

[11] S. Kimmel, *Chicago Journal of Theoretical Computer Science*, vol. 19, no. 1, pp. 1–14, 2013. [Online]. Available: https://doi.org/10.4086%2Fcjtcs.2013.004

[12] I. Kerenidis and A. Prakash, “A quantum interior point method for lps and sdps,” *ACM Transactions on Quantum Computing*, vol. 1, pp. 1 – 32, 2018.

[13] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming.* Society for Industrial and Applied Mathematics, 1994. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611970791

[14] A. W. Harrow, A. Hassidim, and S. Lloyd, “Quantum algorithm for linear systems of equations,” *Phys. Rev. Lett.*, vol. 103, p. 150502, Oct 2009. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.103.150502

[15] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018. [Online]. Available: https://doi.org/10.22331/q-2018-08-06-79

[16] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: A survey," *Surveys in Operations Research and Management Science*, vol. 17, pp. 97–106, 2012.