Quantum Semidefinite Programming

Takla Nateeboon

April 22, 2024

Abstract

In this report, I provide an algorithm for solving semidefinite programming (SDP) using sampling from the oracle of a Gibbs state. The algorithm solves an SDP by solving the dual problem of the SDP[1, 2, 3]. The algorithm takes an oracle of constraint matrices as its input and outputs a probability distribution sampling from a Gibbs state associated with the constraints. The runtime of this algorithm is $O(\sqrt{nm}s^2R^{32}/\delta^{18})$ when m is the number of constraint matrices, n is the dimension of the constraint matrices. I will provide a derivation of the lower bound for the quantum algorithm. Additionally, I will also give examples of this algorithm in a low-dimensional case and a linear programming case.

Contents

Notations

In this report, $C, X \in \mathbb{R}^{n\times n}$, $\mathbf{y} \in \mathbb{R}^m$, and $\{A_j\}_{j\in[m]}$ always represent the description of a semidefinite program. Matrices are represented with uppercase Roman letters; vectors are represented with lowercase bold typeface. i, j, k, l are indices; one with subscript are marked or random indices. A number in a square bracket, e.g. $[m]$, denotes the set of m indices.

1 Semidefinite program

A semidefinite program (SDP) is an extension of the linear program. An objective is to find a matrix X, instead of a vector in linear programming, that maximizes a certain objective function of the form $\mathrm{Tr}(CX)$. Formally, the SDP is given by

$$\begin{aligned} & \max \mathrm{Tr}\left[CX\right] \\ \forall j \in [m] \quad & \mathrm{Tr}\left[A_j X\right] \leq b_j \\ & X \geq 0, \end{aligned} \tag{1}$$

where the inequality $X \geq 0$ denotes X being positive semidefinite (PSD). This form of SDP is called Primal SDP, or P-SDP. C, A_j and X are Hermitian matrix of size $n \times n$. The matrix C is called an objective matrix and A_j are constraint matrices each with corresponding constraint value b_j. Each SDP has its dual of the form

$$\begin{aligned} & \min \mathbf{b} \cdot \mathbf{y} \\ & \sum_{j=1}^{m} y_j A_j \geq C \\ & y \geq 0 \end{aligned} \tag{2}$$

where the inequality $\sum_{j=1}^{m} y_j A_j \geq C$ denotes $\sum_{j=1}^{m} y_j A_j - C \geq 0$ is a PSD matrix. This is called the dual SDP, D-SDP.

1.1 Feasibility and Duality

Definition 1.1 (Feasible solution)**.** X is a primal feasible solution if it satisfies the constraints

$$\begin{aligned} \forall j \in [m] \quad & \mathrm{tr}\left(A_j X\right) \leq b_j \\ & X \geq 0. \end{aligned}$$

Moreover, the value $\mathrm{Tr}[CX]$ is feasible for this SDP. Similarly, $\mathbf{y}$ is a dual feasible solution if it satisfies the constraints

$$\begin{aligned} & \sum_{j=1}^{m} y_j A_j \geq C \\ & y \geq 0. \end{aligned}$$

Proposition 1.2. If $X, \mathbf{y}$ are primal and dual feasible solutions respectively with objective values α_p and α_d respectively, then $\alpha_p \leq \alpha_d$.

Proof. Since $\mathbf{y}$ is dual feasible, then,

$$\sum_j y_j A_j \geq C$$
$$(X \text{ is PSD}) \rightarrow \sum_j y_j A_j X \geq CX$$
$$\text{Tr}\left[\sum_j y_j A_j X\right] \geq \text{Tr}\,[CX]. \tag{3}$$

Since X is primal feasible, then we have

$$\sum_j b_j y_j \geq \text{Tr}\left[\sum_j y_j A_j X\right] \geq \text{Tr}\,[CX]. \tag{4}$$

Equivalently, $\mathbf{b} \cdot \mathbf{y} \geq \text{Tr}[CX]$ and $\alpha_p \leq \alpha_d$. □

Definition 1.3. An SDP satisfies a strong duality if its primal and dual optimal values coincide.

Not all SDPs have a strong duality with their dual. The algorithm in this report assumes that the strong duality is satisfied. If the strong duality is satisfied, there is a way to check if X is feasible by trying to come up with a counter-example, as stated in the next proposition.

Proposition 1.4. Given a candidate primal solution X and a guessed α, if there is a vector $\mathbf{y} \geq 0$ such that

$$\mathbf{b} \cdot \mathbf{y} \leq \alpha \quad \text{and} \tag{5}$$
$$\text{Tr}\left[\sum_j y_j A_j X\right] \geq \text{Tr}\,[CX] \tag{6}$$

then X is either infeasible or X is feasible and the optimal value has value at most α.

The problem with this is to exhaust all possible $\mathbf{y}$. There is a workaround by assuming that there exists an oracle that can approximately exhaust these vectors. It will be discussed in the section 2.2.

Proof. X is either feasible or infeasible. From the conditions in equations (5) and (6), if X is feasible then

$$\alpha \geq \mathbf{b} \cdot \mathbf{y} \geq \sum_j \text{Tr}[A_j X] y_j \geq \text{Tr}[CX].$$

The last inequality is given by $\sum_{j=1}^m \mathbf{A}_j y_j \geq \mathbf{C}$. This shows that if X is feasible then it has value at most α. The other case is X is infeasible, which there is nothing to prove. □

Theorem 1.5 (Sufficient condition for strong duality, from [4])**.** Assume both primal and dual have feasible solutions. Then $\alpha_p \geq \alpha_d$, where α_p and α_d are the optimal values to the primal and dual respectively. Moreover, if the primal has a strictly feasible solution (a solution X such that $X > 0$ or X is positive definite) then

1. The dual optimum is attained (which is not always the case for SDPs)

2. $\alpha_p = \alpha_d$.

Similarly, if the dual is strictly feasible, then the primal optimal value is achieved and equals the dual optimal value. Hence, if both the primal and dual have strictly feasible solutions, then both α_p and α_d are attained.

Lemma 1.6 (Sufficient condition for strong duality 2, from [5])**.** Assume $(X, \mathbf{y})$ are primal and dual feasible solutions of (SDP-P) and (SDP-D), respectively, that satisfy the complementary slackness condition

$$\left(C - \sum_{i=1}^{m} A_i y_i\right) X = 0 \tag{7}$$

(and thus achieve the same cost $\mathrm{Tr}[CX] = \mathbf{b} \cdot \mathbf{y}$. Then, $(X, \mathbf{y})$ are primal and dual optimal solutions of the SDP problem.

2 Classical algorithm for semidefinite programming

The algorithm solves either SDP-P or SDP-D. To solve an SDP, the algorithm turns the SDP into a feasibility problem and uses a search algorithm to guess the optimal value α.

Before we dive into technical details. The intuitive idea of the algorithm for solving semidefinite programming is as follows

1. Guess α the optimal value.

2. Find X such that it is feasible and give its value

   (a) If can find, then guess a higher value for α.

   (b) If cannot find it, then guess a lower value for α.

To check if there is such X, aka feasibility testing, the algorithm for this is as follows

1. Initially guess $X \propto I$

2. Find $\mathbf{y}$ such that it satisfies the equation (6).

   (a) This step uses an oracle in the Definition 2.6

3. If cannot be found, then X is a feasible solution and the optimal value is higher.

4. If can find such $\mathbf{y}$, then X may not be feasible and we use the amount, $\sum_{j=1}^{m} y_j(t) A_j - C$ to reduce X accordingly.

5. Repeat with the new X.

Firstly, let us consider how one can use an algorithm called matrix weight multiplication to reduce X and adjust X according to $\sum_{j=1}^{m} y_j(t) A_j - C$.

2.1 Matrix multiplicative weights algorithm

The matrix multiplicative weight algorithm (MMW) is an important subroutine in the classical algorithm for SDP. The goal of the algorithm is as follows.

Input: Loss matrices $\{M(t)\}_{t\in[T]}$ feeding to the algorithm one at a time.

Output: A density operator such that $\mathrm{Tr}[\sum_t M(t)P]$ is close to the lowest eigenvalue of $M(t)$ by a function of $\eta,\ T$ and N.

Algorithm 1 Multilicative weights algorithm

Require: Parameter $\eta \leq 1$, number of rounds T.

1: $W(1) \leftarrow I$

2: **for** $t = 1, \ldots, T$ **do**

3: Set a density operator $P^{(t)} = W^{(t)} / \mathrm{Tr}\left[W^{(t)}\right]$

4: Recieve $M^{(t)}$

5: Update the weight matrix W with

$$W^{(t+1)} = \exp\left(-\eta \sum_{\tau=1}^{t} M^{(\tau)}\right).$$

6: **end for**

Theorem 2.1. For any sequence of loss matrix $M^{(1)}, M^{(2)}, \ldots, M^{(T)}$, the MMW generates density matrices $P^{(1)}, \ldots, P^{(T)}$ such that

$$\sum_{t=1}^{T} \mathrm{Tr}\left[M^{(t)}P^{(t)}\right] - \lambda_n\left(\sum_{t=1}^{T} M^{(t)}\right) \leq \eta \sum_{t=1}^{T} \mathrm{Tr}\left[(M^{(t)})^2 P^{(t)}\right] + \frac{\ln n}{\eta} \tag{8}$$

Corollary 2.2. For any sequence of loss matrix $M^{(1)}, M^{(2)}, \ldots, M^{(T)}$, the MMW generates density matrices $P^{(1)}, \ldots, P^{(T)}$ such that

$$\sum_{t=1}^{T} \mathrm{Tr}\left[M^{(t)}P^{(t)}\right] - \lambda_n\left(\sum_{t=1}^{T} M^{(t)}\right) \leq \eta T + \frac{\ln n}{\eta} \tag{9}$$

2.2 Primal-dual algorithm

The algorithms, both classical and quantum, assume the following

- $\alpha \geq 1$
- $b_j \geq 1$ for all j
- $\|A_j\|, \|C\| \leq 1$.

Also, these additional bounds are assumed.

Definition 2.3 (Trace bound)**.** For any of the SDPs considered after this, we set $A_1 = I$ and $b_1 = R$. This gives the first condition of feasibility, which is

$$\mathrm{Tr}[X] = R$$

This R is called a trace bound.

Definition 2.4 (Dual bound)**.** For any of SDP-D considered after this, we set a constraint that $\|\mathbf{y}\|_1 \leq r$.

In [3], they proved that one can rescale the problem to satisfy these bounds, solve using their algorithm, and pullback to be the answer for the original SDP using the following theorem.

Theorem 2.5 (Lemma 2 in [3])**.** One can sample from a δ-optimal solution of the SDP given by Eq. (2) (with dimension n, m variables, size parameter R and upper bound r on optimal solution vector) given the ability to sample from a (δ/r)-optimal solution of the SDP given by *Eq.* (2) (with dimension $n+1, m+1$ variables and size parameter $2R+1$) in which $b_i \geq 1$ for all $i \in [m]$.

The algorithm assumes that there exists an oracle that outputs $\mathbf{y}$ such that it flags where X may violate the constraint. If this oracle fails to output $\mathbf{y}$, then X is feasible (see the proposition 1.4).

Definition 2.6. Oracle is an algorithm that, given a primal candidate solution $X \succeq 0$, either outputs $\mathbf{y}$ from a convex polytope

$$\begin{aligned}\mathcal{P}(X) := \{\mathbf{y} \in \mathbb{R}^m : &\mathbf{b} \cdot \mathbf{y} \leq \alpha \\ &\operatorname{Tr}\left[\sum_{j=1}^{m} y_j A_j X\right] \geq \operatorname{Tr}[CX] \\ &\mathbf{y} \geq 0\},\end{aligned}$$

or outputs fail.

Definition 2.7 (Oracle width-bound)**.** The width of oracle is the smallest non-negative w such that every primal candidate X, the vector returned by the Oracle satisfies $\left\|\sum_j A_j y_j - C\right\| \leq w$

Proposition 2.8. $w \leq r+1$

Proof.

$$\begin{aligned}w = \sup_{\mathbf{y}\in\mathcal{P}(X)} \left\|\sum_j y_j A_j - C\right\| &\leq \left\|\sum_j y_j A_j\right\| + \|C\| \\ &\leq \sum_j y_j \|A_j\| + \|C\| \\ &\leq \sum_j y_j + 1 = r+1\end{aligned}$$

□

The following algorithm is firstly shown in [1] and rephrased in [6]

Theorem 2.9 (from [1])**.** Suppose that the primal-dual SDP algorithm is run with the parameter settings $\eta = \frac{\varepsilon}{2\rho R}$ and $T = \left\lceil \frac{4\rho^2 R^2 \ln(n)}{\varepsilon^2} \right\rceil$, and assume that ORACLE never fails in any of the T iterations. Then the dual solution output, $\overline{\mathbf{y}}$, is feasible with objective value at most $\alpha + \varepsilon$

2.3 Complexity

The table 1 is from [1] showing the complexity of the algorithm 2. Notice that all complexity is larger than $O(\sqrt{nm})$.

Algorithm 2 Primal-dual algorithm for feasibility SDP

Require: Trace bound R, the dual bound r, parameteres η and T for MMW. Constraint matrices and an objective matrix $\{A_j\}_{j\in[m]}, C$.

Take $\rho(1) := I/n$
Set $w = r + 1$
for t=1,2,...,T **do**
    $X(t) = R\rho(t)$
    $y(t) \leftarrow \mathsf{Oracle}(X(t))$. If Oracle fails, abort. ▷ Check if there is a dual feasible solution with this X.
    Set $M(t) = \frac{1}{w}\left(\sum_{j=1}^{m} A_j y_j(t) - C\right)$ ▷ If there is such $\mathbf{y}$, X may not be primal feasible.
    Set $H(t) = \sum_{\tau=1}^{t} M(t)$, and $W(t) = \exp\left(-\eta H(t)\right)$
    Set $\rho(t+1) \leftarrow W(t)/\operatorname{Tr}[W(t)]$ ▷ (the t-th element of MMW)
end for
If Oracle never fails in any of T rounds, output the dual solution $\bar{\mathbf{y}} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{y}(t) + \frac{\varepsilon}{R}\hat{\mathbf{e}}_1$

Table 1: Complexity of algorithm in each specific type of SPD. The table is taken directly from [1]

Problem	Previous Best: $O(\sqrt{\log n})$ apx	Algorithm 2: $O(\sqrt{\log n})$ apx	Algorithm 2: $O(\log n)$ apx
Undir. SPARSEST CUT	$\tilde{O}\left(n^2\right)$	$\tilde{O}\left(n^2\right)$	$\tilde{O}\left(m + n^{1+o}(1)\right)$
Undir. BALANCED SEPARATOR	$\tilde{O}\left(n^2\right)$	$\tilde{O}\left(n^2\right)$	$\tilde{O}\left(m + n^{1+o(1)}\right)$
Dir. SPARSEST CUT	$\tilde{O}\left(n^{9.5}\right)$	$\tilde{O}\left(m^{1.5} + n^{2+o(1)}\right)$	$\tilde{O}\left(m^{1.5}\right)$
Dir. BALANCED SEPARATOR	$\tilde{O}\left(n^{9.5}\right)$	$\tilde{O}\left(m^{1.5} + n^{2+o(1)}\right)$	$\tilde{O}\left(m^{1.5}\right)$
MIN UNCUT	$\tilde{O}\left(n^{9.5}\right)$	$\tilde{O}\left(n^4\right)$	-
MIN 2CNF DELETION	$\tilde{O}\left(n^{9.5}\right)$	$\tilde{O}\left(nm^{1.5} + n^4\right)$	-

3 Quantum algorithm

The quantum algorithm approaches SDP similarly to the Primal-dual algorithm. It also uses the same assumption. The quantum algorithm replaces matrix exponentiation with quantum Gibbs state preparation and the Oracle with the estimation using samples from the distribution of the Gibbs state.

3.1 Quantization ideas

In this subsection, we will discuss the following points.

1. One can prepare a thermal state (the Gibbs state) instead of doing matrix exponentiation in the multiplicative weight method.
2. One can create an approximate Oracle quantumly using Gibbs sampling method.

3.1.1 Gibb States

A Gibbs state, ρ, is a quantum state of the form

$$\rho = \frac{\exp\left(\beta H\right)}{\operatorname{Tr}[\exp\left(\beta H\right)]} \tag{10}$$

where H is a Hermitian matrix and β is a positive parameter. Notice that this is similar to the state we required in MMW and the primal-dual algorithm. Preparing Gibbs state can be done with an oracle that

gives access to the matrix elements of the matrix H. For this algorithm, the oracle is working on a sparse matrix and the oracle is defined as follows.

Definition 3.1 (Oracle for matrix elements)**.** Given a list of matrix $\{A_1, \ldots, A_m, A_{m+1} = C\}$. An oracle for matrix elements of these matrices A_j is a quantum oracle that maps

$$|j,k,l,z\rangle \mapsto \left|j,k,l,z \oplus (A_j)_{k f_{jk}(l)}\right\rangle \tag{11}$$

where A_j is an s-spares matrix, f_{jk} is a function mapping $l \in [s]$ to an index of the l-th nonzero index of row k of the matrix A_j.

Definition 3.2 (GibbsSampler)**.** Given H a Hamiltonian and $O[H]$ an oracle for its entries. GibbsSampler$(O[H], \varepsilon)$ is a quantum operation that outputs a state ρ such that $\|\rho - e^H / \operatorname{Tr} e^H\|_1 \leq \varepsilon$.

The following proposition confirms that we can use queries bounded by $O(\sqrt{\dim H})$ to prepare for a Gibbs state. It appears in the proof of corollary 17 of [3].

Proposition 3.3. One can prepare an ε-approximation th the Gibbs state of H, $e^{\beta H} / \operatorname{Tr}\left[e^{\beta H}\right]$, quantumly using an quantum oracle inquiring elements of H. The combined query and two-qubit gates complexity is of the order

$$O\left(\frac{1}{\varepsilon}\sqrt{\dim(H)}\beta s'\right), \tag{12}$$

when H is s'-spares and $\|H\| \leq 1$. (each column or row has at most s' nonzero elements)

Remark. Given any $\|H\| \geq 1$ one can rescale H to $\beta H'$ with $\|H'\| \leq 1$.

Definition 3.4. Given a quantum state ρ, define $h(\rho, \lambda, \mu)$

$$h(\rho,\lambda,\mu) = \sum_{i=1}^{m} r_i \left|i\right\rangle\!\left\langle i\right|,$$
$$\text{when} \quad r_i = \lambda \operatorname{Tr}\left[A_i \rho\right] + \mu b_i.$$

Corresponding to h, define a truncated version as

$$\bar{h}(\rho,\lambda,\mu) = \sum_{i=1}^{m} \bar{r_i} \left|i\right\rangle\!\left\langle i\right|, \tag{13}$$

where $\bar{r}_i$ is a rounding of r_i to the precision of $\delta/(56R^2)$.

Remark. In the algorithm, the Hamiltonian is noted by $\bar{h}(\rho, k)$ which is defined as

$$\bar{h}(\rho,k) := \bar{h}\left(\rho, -\frac{\varepsilon}{8R^2}k, -\frac{\varepsilon}{8R^2}(\gamma - k)\right)$$

where $\gamma := \left\lceil \frac{8}{\varepsilon^2}\log(m)R^2 \right\rceil \geq: k$ and for an $\varepsilon > 0$ defined in the algorithm.

3.1.2 A quantum counterpart of the Oracle

There are two tasks in the Oracle

1. come up with a suitable **q** that may be in $\mathcal{P}$

2. evaluate if $\mathbf{q}$ is in this polytope

First, in the algorithm, the candidate $\mathbf{q}$ is sampled from the $\mathsf{GibbsSampler}\big(\bar{h}(\rho,k)\big)$. Then, we sample $i_1, i_2, \ldots, i_m$ independently from $\mathbf{q}$, the algorithm approximately computes

$$\frac{1}{M}\sum_{j\in[M]} \mathrm{Tr}\big[A_{i_j}\rho\big]. \tag{14}$$

To understand what happens here, consider the limit $M \to \infty$,

$$\lim_{M\to\infty}\frac{1}{M}\sum_{j\in[M]} \mathrm{Tr}\big[A_{i_j}\rho\big] = \sum_{j\in[m]} q_j \,\mathrm{Tr}[A_j\rho]$$

this is corresponding to the sum over $y_j \,\mathrm{Tr}[A_j\rho]$ in the Oracle. Another term calculated from the sample is

$$\frac{1}{M}\sum_{j\in[M]} b_{i_j}. \tag{15}$$

Similar to the previous calculation,

$$\lim_{M\to\infty}\frac{1}{M}\sum_{j\in[M]} b_{i_j} = \sum_{j\in[m]} q_j b_j = \mathbf{q}\cdot\mathbf{b}.$$

As seen in the quantum algorithm, these two terms will be compared to another two terms in inequalities. A lemma in [3] shows that if the Oracle does not fail then comparing terms in equations 14 and 15 will output a vector $\mathbf{q}$ approximating the output of Oracle.

3.2 The algorithm

The algorithm 3 is given in [3] and [2].

Theorem 3.5. The algorithm runs in time

$$\tilde{O}\left(\frac{R^{21}}{\delta^{11}}G_{\bar{h}}G_M\right) + O\left(\frac{R^{13}}{\delta^5}T_{\mathrm{Meas}}\right)$$

when $G_{\bar{h}}$ is a maximum number of queries that $\mathsf{GibbsSampler}\big(O(\bar{h}(\rho(t),k)),\varepsilon/4\big)$ use over any $t \le T$, G_M is the number of queries for $\mathsf{GibbsSampler}\Big(O\Big[-\varepsilon'\sum_{j\in[t]M(j)}\Big],\varepsilon/4\Big)$, and T_{Meas} is an estimating time for each of $\mathrm{Tr}[A_i\rho(t)]$ and $\mathrm{Tr}[C\rho(t)]$.
The algorithm fails with probability at most

$$O\left(\left(\frac{R}{\delta}\right)^{18}(nm)^{10}\exp\Big(-\log^{\xi}(nm)\Big)\right) \tag{16}$$

The Gibbs sampler is considered here as a black box that uses the number of queries to prepare a Gibbs state with ε-close. The number of queries is bounded by

$$O\Big(\sqrt{\dim(H)}\beta s'/\varepsilon\Big)$$

as in proposition 3.3. There are two inputs for $\mathsf{GibbsSampler}$, one is the Hamiltonian $\bar{h}(\rho)$, another one is

Algorithm 3 Quantum Primal-Dual Algorithm for SDPs

Require: Trace bound R, guessed optimal value α, multiplicative error δ, a parameter $\xi > 0$, description of the problem $\{A_1, \ldots, A_m, C\}$ such that $\|A_j\|, \|C\| \leq 1$ and $\{b_1, \ldots, b_m\}$ such that $b_i \geq 1$.

Output: $\mathbf{y}$ a dual feasible with objective value less than $(1+\delta)\alpha$, or the label Larger indicating the optimal objective value is larger than $(1-\delta)\alpha$.

Set $\rho(1) = I/n$.

Let $\varepsilon = \frac{\delta}{28R^2}, \varepsilon' = -\ln(1-\varepsilon)$, ▷ Start: set many complicated parameters

$M = 80\log^{1+\xi}\left(8R^2nm/\varepsilon\right)/\varepsilon^2$,

$L = 80\log^{1+\xi}(nm)/\varepsilon^{\varepsilon}$ and

$Q = 10^6 R^6 \ln^{2+\xi}(\mathrm{nm})/\delta^4$. ▷ End: set many complicated parameters

for t=1,2,..., $T = \frac{500R^3\ln(n)}{\delta^2}$. **do**

  Start: analog to the Oracle in classical algorithm.

  Set $\mathbf{y}(t) = \begin{pmatrix}0 & \ldots & 0\end{pmatrix}^T$.

  for $k = 1, \ldots, \gamma$ **do**

    for $N = 1, \ldots, \lceil \alpha/\varepsilon \rceil$ **do**

      Create M copies of $\mathbf{q} \leftarrow \mathsf{GibbsSampler}\left(O[\bar{h}(\rho(t), k)], \varepsilon/4\right)$

      Sample $i_1, \ldots, i_M$ independently from $\mathbf{q}$.

      Compute estimates $e_{i_j} \approx \mathrm{Tr}\left[A_{i_j}\rho(t)\right]$ and $f \approx \mathrm{Tr}[C\rho]$ with accuracy $\varepsilon/2$.

      Check if $\mathbf{q}$ is in the polytope

      if $\frac{1}{M}\sum_{j\in[M]} e_{i_j} \geq \frac{f}{\varepsilon N} - \varepsilon$ and $1/M\sum_{j\in[M]} b_{i_j} \leq \alpha/(\varepsilon n) + R\varepsilon$, **then**.

        Admit $k_t \leftarrow k$, and $N_t \leftarrow N$, $\mathbf{q}(t) = \mathbf{q}$ and $\mathbf{y}(t) = \varepsilon N \mathbf{q}(t)$.

      end if

    end for

  end for

  End: Analog to the Oracle.

  If $\mathbf{y} = 0$, abort and label Larger.

  Start: analog to MMW

  Create $Q+1$ copies of $\mathbf{q}(t) \leftarrow \mathsf{GibbsSampler}\left(O\left[\bar{h}(\rho(t), k(t))\right], \varepsilon/4\right)$

  Sample $i_1, \ldots, i_Q$ independently from $\mathbf{q}(t)$.

  Set $M(t) = \left(\varepsilon N(t) Q^{-1}\sum_{j=1}^{Q} A_{i_j} - C + 2\alpha I\right)/4\alpha$.

  Set $C(t) := \frac{10\log(m)}{\varepsilon^2}\left(\frac{\gamma\alpha}{\varepsilon}M + Q\right) G_{\bar{h}}\left(\rho^{(t)}\right) + \frac{2\gamma\alpha}{\varepsilon}ML$

  Set create $C(t)$ copies of $\rho(t+1) \leftarrow \mathsf{GibbsSampler}\left(-\varepsilon'\left(\sum_{\tau\in[t]} M(\tau), \epsilon/4\right)\right)$

  End: analog to MMW

end for

Output: $\|\bar{y}\|_1$ and a sample from $\bar{y}/\|\bar{y}\|_1$ with $\bar{y} = \frac{\delta\alpha}{2R}e_1 + \frac{1}{T}\sum_{t=1}^{T} y^{(t)}$

the constraint violation $M(t)$. The first $\bar{h}$ has a dimension equal to the number of constraints m and the second has a dimension equal to the n. We will state the query complexity here, the details derivation can be found in lemma 17 of [3],

$$\text{The query complexity is } \tilde{O}(\sqrt{nm}s^2R^{32}/\delta^{18}).$$

3.3 Lower bound

Consider the following two SDPs

$$\begin{aligned} &R = 1 && A_1 = I \\ &b_j = 1 \quad \forall j \in [m] \\ &\exists! i_0 \in [n] \quad C_{ii} = 1 && C_{kl} = 0 \quad \text{elsewhere} \end{aligned}$$

and either

1. $\exists! j_0 \in [m]$ such that $(A_{j_0})_{i_0 i_0} = 2$ and elements of A_{j_0} equal to zero elsewhere. All other matrices are zero matrices.
2. All the matrices A_j are zero matrices $\forall j > 1$.

Solving these two problems leads to two different results. The first case has the optimal solution $X = |i_0\rangle\langle i_0|/2$ and $\mathbf{y} = |j_0\rangle\langle j_0|$ with the objective value 1/2. While the second case $X = |i_0\rangle\langle i_0|$ and $\mathbf{y} = |1\rangle\langle 1|$ with the objective value 1. Given an oracle that gives out the optimal solution, one can query the oracle with complexity n to find what i_0 and j_0. This is solving a search problem that quantumly requires $O(\sqrt{n}+\sqrt{m})$ queries to complete with bounded error. The algorithm is able to come up with an approximate solution that can distinguish these two cases with bounded errors. Therefore, the algorithm requires at least $\Omega(\sqrt{n}+\sqrt{m})$ queries to the oracles to complete.

4 Conclusion

Semidefinite programming can be solved quantumly using the quantum version of the Primal-dual approach to solve semidefinite programming. The algorithm relies on access to Gibb sampling and the oracles inquiring elements of a matrix. The query complexity of this algorithm is $O(\sqrt{nm})$ when n is a dimension of the matrix and m is the number of constraints. There is room for improvement as the lower bound is $O(\sqrt{n}+\sqrt{m})$.

References

[1] Sanjeev Arora and Satyen Kale. "A Combinatorial, Primal-Dual Approach to Semidefinite Programs". In: *J. ACM* 63.2 (Apr. 2016). ISSN: 0004-5411. DOI: 10.1145/2837020. URL: https://doi.org/10.1145/2837020.

[2] F. L. Brandao and K. M. Svore. "Quantum Speed-Ups for Solving Semidefinite Programs". In: *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2017, pp. 415–426. DOI: 10.1109/FOCS.2017.45. URL: https://doi.ieeecomputersociety.org/10.1109/FOCS.2017.45.

[3] Fernando G. S. L. Brandao and Krysta Svore. *Quantum Speed-ups for Semidefinite Programming*. 2017. arXiv: 1609.05537 [quant-ph].

[4] Alex Beutel Anupam Gupta. *Semidefinite Duality*. 2021. URL: https://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f11/www/.

[5] G. Blekherman, P. A. Parrilo, and R. R. Thomas. "Semidefinite optimization and convex algebraic geometry". In: (2012). DOI: 10.1137/1.9781611972290.

[6] Joran van Apeldoorn et al. "Quantum SDP-Solvers: Better upper and lower bounds". In: *Quantum* 4 (Feb. 2020), p. 230. ISSN: 2521-327X. DOI: 10.22331/q-2020-02-14-230. URL: https://doi.org/10.22331/q-2020-02-14-230.