Lecture on November 15 and 20, 2018

Prepared by *Tongyang Li, Xiaodi Wu*

Lecture notes for quantum semidefinite programming (SDP) solvers

---

1 Definitions

1.1 Linear program (LP)

Last time we introduced the HHL algorithm by Harrow, Hassidim, and Lloyd for solving linear systems. In particular, it solves sparse linear systems with complexity only poly-logarithmic in dimension n. A very natural question to ask is: beyond *solving* linear systems, could we give quantum speedup for other perspective of linear systems?

In this lecture, we consider *optimization* of linear systems, also known as *linear programs*. Mathematically, given an $m \times n$ matrix A, a constraint vector $b \in \mathbb{R}^n$ and a target vector $c \in \mathbb{R}^n$, the goal is to solve

$$\max \quad \sum_{j\in[n]} c_i x_i \tag{1}$$

$$\text{s.t.} \quad \sum_{j\in[n]} A_{ij} x_j \leq b_j \quad \forall i \in [m]; \tag{2}$$

$$x_i \geq 0 \quad \forall i \in [m]. \tag{3}$$

As an abbreviation, this is also written as

$$\max\ c^\dagger x \quad \text{s.t. } Ax \leq b,\ x \geq 0. \tag{4}$$

Linear programmings are important for two reasons. First, it is widely used in real-world problems.

For instance, assume that you have access to apples and bananas from a farmer and sell it to customers. It is known that the price of apples is \$3 per lb, and the price of bananas is \$2 per lb. The constraints are: you can only store 10 lbs food; the farmer has only 6 lbs apples and 6 lbs bananas. The maximization of your income is an LP problem:

$$\max\ 3x + 2y \quad \text{s.t. } x + y \leq 10,\ 0 \leq x, y \leq 6. \tag{5}$$

The best solution is when $x = 6$ and $y = 4$, and you get total income $3 \cdot 6 + 2 \cdot 4 = 26$. In general, many real-world optimization problems have linearity on the constraints and target, which perfectly fit into LPs.

On the other hand, LPs can be solved efficiently – there are polynomial time classical algorithms for solving LPs. Actually, there exist efficient algorithms for a larger class of problems, namely semidefinite programs (SDPs). This will be the main topic of today's lecture.

1.2 Semi-definite program (SDP)

Similar to LP, SDP is a central topic in the study of mathematical optimization, theoretical computer science, and operations research in the last decades. The power of SDP lies in their generality (that extends LP) and the fact that it admits polynomial-time solvers. Mathematically, given m real numbers $b_1, \ldots, b_m \in \mathbb{R}$ and $n \times n$ Hermitian matrices $A_1, \ldots, A_m, C$, the goal is to solve

$$\max \quad \mathrm{Tr}[CX] \tag{6}$$

$$\text{s.t.} \quad \mathrm{Tr}[A_i X] \leq b_i \quad \forall\, i \in [m]; \tag{7}$$

$$X \succeq 0. \tag{8}$$

Here, the variable matrix X is restricted to be positive semidefinite; this is the reason why it is called an SDP. Notice that this SDP contains the LP as a special case if we take $C = \mathrm{diag}(c)$ and $A_i = \mathrm{diag}(A_{i1}, \ldots, A_{in})$ for all $i \in [m]$, where c is from Eq. (1) and $A_{i1}, \ldots, A_{in}$ are from Eq. (2), respectively.

We consider an example of SDP that is similar to (5). Consider

$$\max \quad \mathrm{Tr}\left[\begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix} X\right] \tag{9}$$

$$\text{s.t.} \quad \mathrm{Tr}\left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} X\right] \le 10,\ \mathrm{Tr}\left[\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} X\right] \le 6,\ \mathrm{Tr}\left[\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} X\right] \le 6;\ X \succeq 0. \tag{10}$$

If we denote $X = \begin{pmatrix} x & z \\ z & y \end{pmatrix}$, then the SDP is equivalent to

$$\max\ 3x + 2y + 2z \quad \text{s.t. } x + y \le 10,\ 0 \le x, y \le 6,\ xy - z^2 \ge 0. \tag{11}$$

The maximum must take place when $x \in [0, 6]$, $y = 10 - x$ and $z = \sqrt{xy} = \sqrt{x(10-x)}$; in this case,

$$\max_{x \in [0,6]} 3x + 2y + 2z = \max_{x \in [0,6]} 20 + x + 2\sqrt{x(10-x)}. \tag{12}$$

Its optimal solution is $x = 6, y = 4, z = \sqrt{24}$, where the maximum is $26 + 2\sqrt{24}$.

You might feel that SDP is more complicated than LP because the semi-definite constraint $X \succeq 0$ seems to be a bit involved to keep with. However, in the rest of my lecture I will introduce a framework called the *matrix multiplicative weight* method, which can solve SDPs efficiently.

2 Matrix multiplicative weights (MMW)

2.1 Multiplicative weight (MW) method

To begin with, I will introduce how people come up with the multiplicative weight (MW) idea. We start with a related problem: *learning with experts*. This problem has T rounds and n "experts". At the beginning of round $t \in [T]$, there is an enclosed value $y_t \in \{0, 1\}$ and all experts make a guess for y_t. The learner is then asked to make her own prediction for y_t, denoted $\hat{y}_t$. At the end of round t, the true value of y_t is revealed and the learner makes a mistake if $\hat{y}_t \neq y_t$.

This learning with expert problem is very natural as it is a basic case for many *online* problems. For instance, you can put stock prediction into this problem: the experts are from the Wall Street, making predictions about whether a certain stock is going up or down; you hear the opinions from these experts and make your own predictions. At the end of the day, you know whether it is indeed going up or down, and could then make corresponding changes for the next day. Intuitively, a good strategy at the end of each round is to penalize those experts who make a wrong prediction for the round. How should we do this mathematically?

One possible solution is to assign *weights* to the experts: at each round you make prediction according to the weights of all experts, and penalize wrong experts by making their weight exponentially smaller. An algorithm along this line looks like this:

Algorithm 1: Multiplicative weights method for learning with experts.

Initialization: Fix a $\delta \le 1/2$. Initialize the weight vector $w^{(1)} = \mathbf{1}_n$;
for $t = 1, 2, \ldots, T$ **do**
  For all $i \in [m]$, receive the prediction $\xi_i \in \{0, 1\}$ from expert i. Take $q_0 = \sum_{i:\ \xi_i = 0} w^{(t)}$ and $q_1 = \sum_{i:\ \xi_i = 1} w^{(t)}$;
  The learner predicts $\hat{y} \in \{0, 1\}$ such that $\hat{y} = 1$ with probability $q_1/(q_0 + q_1)$;
  The learner observes the true value $y \in \{0, 1\}$;
  For each i such that $\xi_i \neq y$, $w_i^{(t+1)} \leftarrow \delta w_i^{(t)}$; otherwise $w_i^{(t+1)} \leftarrow w_i^{(t)}$.

For this problem, the learner cannot be absolutely good – it suffices that she is relatively good compared to the experts, namely the expert making fewest mistakes (the "best" expert). In fact, we could prove:

Theorem 2.1. *Denote L_A as the number of mistakes made by Algorithm 1 and L as the number of mistakes made by the best expert. Then*

$$\mathbb{E}[L_A] \leq \frac{\ln(1/\delta)}{1-\delta} L + \frac{1}{1-\delta} \ln n. \tag{13}$$

Proof. At round t, we denote $W_t = \sum_{i=1}^n w_i^{(t)}$ and denote l_t as the probability that Algorithm 1 makes a mistake: $l = \sum_{i:\, \xi_i \neq y_t} w_i^{(t)} / W_t$. Then the weight in this round is changed as

$$W_{t+1} = \sum_{i:\, \xi_i \neq y_t} \delta w_i^{(t)} + \sum_{i:\, \xi_i = y_t} w_i^{(t)} = \delta l_t W_t + (1 - l_t) W_t = W_t\big(1 - l_t(1-\delta)\big). \tag{14}$$

As a result,

$$W_T = n \prod_{t=1}^T \big(1 - l_t(1-\delta)\big) \leq n \prod_{t=1}^T \exp(-l_t(1-\delta)) = n \cdot \exp\Big[-(1-\delta)\sum_{t=1}^T l_t\Big]. \tag{15}$$

Denote L_i as the number of mistakes made by expert i. Because for all $i \in [n]$, $\delta^{L_i} = w_i^T \leq W_T$, we have

$$\delta^L \leq W_T \leq n \cdot \exp\Big[-(1-\delta)\sum_{t=1}^T l_t\Big]. \tag{16}$$

Notice that $L_A = \sum_{t=1}^T l_t$ is the expected number of mistakes made by Algorithm 1. As a result, Eq. (13) directly follows from Eq. (16). □

2.2 Matrix version

For solving SDPs, we are going to follow this multiplicative weight idea. However, the first technical issue is: the variables in Algorithm 1 are scalar (even boolean), but SDPs play with PSD matrices. Fortunately, there is a matrix version for the MW method, namely the matrix multiplicative weight (MMW) method.

Algorithm 2: Matrix multiplicative weights method.

Initialization: Fix a $\delta \leq 1/2$. Initialize the weight matrix $W^{(1)} = I_n$;
for $t = 1, 2, \ldots, T$ **do**
  Set the density matrix $\rho^{(t)} = \frac{W^{(t)}}{\mathrm{Tr}[W^{(t)}]}$;
  Observe a gain matrix $M^{(t)} \succeq 0$;
  Define the new weight matrix: $W^{(t+1)} = \exp[-\delta \sum_{\tau=1}^t M^{(\tau)}]$;

Theorem 2.2. *Algorithm 2 guarantees that after T rounds, for any density matrix ρ, we have*

$$(1+\delta)\sum_{t=1}^T \mathrm{Tr}[M^{(t)}\rho^{(t)}] \geq \sum_{t=1}^T \mathrm{Tr}[M^{(t)}\rho] - \frac{\ln n}{\delta}. \tag{17}$$

The proof of Theorem 2.2 is basically the same as that of Theorem 2.1, under the help of the the Golden-Thompson inequality. Also notice that Theorem 2.1 is written as a *loss* result, but here for the convenience of solving SDPs we write Theorem 2.2 as a *gain* result, i.e., the sum of traces is not much smaller than the maximal possible trace sum.

3 Solving SDP by matrix multiplicative weights

3.1 Feasibility of SDPs

To solve SDPs by the matrix multiplicative weight method, we first introduce the feasibility problem of SDPs, defined as follows:

Definition 3.1 (Feasibility)**.** Given an $\epsilon > 0$, m real numbers $a_1, \ldots, a_m \in \mathbb{R}$, and Hermitian $n \times n$ matrices $A_1, \ldots, A_m$ where $-I \preceq A_i \preceq I, \forall\, j \in [m]$, define $\mathcal{S}_\epsilon$ as all X such that

$$\mathrm{Tr}[A_i X] \leq a_i + \epsilon \quad \forall\, i \in [m]; \tag{18}$$

$$X \succeq 0; \tag{19}$$

$$\mathrm{Tr}[X] = 1. \tag{20}$$

Notice that this is basically the same as the SDP optimization problem, with the only difference that we only care about whether the SDP is feasible, i.e., whether $\mathcal{S}_\epsilon$ is an non-empty set or not. It is a standard fact that one can use binary search to reduce any optimization problem to a feasibility one. For the normalized case $-I \preceq A \preceq I$, we first guess a candidate value $c_1 = 0$ for the objective function, and add that as a constraint to the optimization problem:

$$\text{s.t. } \mathrm{Tr}[CX] \geq c_1. \tag{21}$$

If this problem is feasible, that means the optima is larger than c_1 and we accordingly take $c_2 = c_1 + \frac{1}{2}$; if this problem is infeasible, that means the optima is smaller than c_1 and we accordingly take $c_2 = c_1 - \frac{1}{2}$; we proceed with $c_3 = c_2 + \frac{1}{2^2}$ or $c_3 = c_2 - \frac{1}{2^2}$ depending on whether the second problem is feasible or not, and the same for all c_i. By using this binary search idea, we could solve the optimization problem with precision ϵ using $\lceil \log_2 \frac{1}{\epsilon} \rceil$ calls to the feasibility problem.

It remains to solve the feasibility problem. Formally, for ϵ-approximate feasibility testing, we require that:

- If $\mathcal{S}_0 = \varnothing$, output "infeasible";
- If $\mathcal{S}_\epsilon \neq \varnothing$, output an $X \in \mathcal{S}_\epsilon$.

Zero-sum game approach for SDPs. Next, we adopt the zero-sum game approach to solve SDPs. This relies on an oracle that searches for violated constraints:

> Input a density matrix X, output an $i \in [m]$ such that Eq. (18) is violated. If no such i exists, output "feasible".

This oracle helps establish a game view to solve any SDP feasibility problem. Imagine Player 1 who wants to provide a feasible $X \in \mathcal{S}_\epsilon$. Player 2, on the other side, wants to find any violation of any proposed X. (This is exactly the function of the oracle.) If the original problem is feasible, there exists a feasible point X_0 (provided by Player 1) such that there is no violation of X_0 that can be found by Player 2. This actually refers to an *equilibrium* point of the zero-sum game, which can be approximated by the matrix multiplicative weight update method.

This game view of solving the SDP feasibility problem has appeared in the classical literatures and has already been used in solving semidefinite programs in the context of quantum complexity theory, in particular $\mathsf{QIP} = \mathsf{PSPACE}$ as introduced in previous lectures. The main difference, however, lies in the way one make use of the matrix multiplicative weight update method, which is a meta algorithm behind both the primal-dual approach and the game view approach.

3.2 Main result

We use Algorithm 2 and Theorem 2.2 to test the feasibility of SDPs.

Algorithm 3: Matrix multiplicative weights algorithm for testing the feasibility of SDPs.

Initialize the weight matrix $W^{(1)} = I_n$, and $T = \lceil \frac{16 \ln n}{\epsilon^2} \rceil$;
for $t = 1, 2, \ldots, T$ **do**
  Prepare the density matrix $\rho^{(t)} = \frac{W^{(t)}}{\mathrm{Tr}[W^{(t)}]}$;
  Find a $j^{(t)} \in \{1, 2, \ldots, m\}$ such that $\mathrm{Tr}(A_{j^{(t)}} \rho^{(t)}) > a_{j^{(t)}} + \epsilon$ by the oracle. Take $M^{(t)} = \frac{1}{2}(I_n - A_{j^{(t)}})$ if such $j^{(t)}$ can be found; otherwise, claim that $\mathcal{S}_\epsilon \neq \varnothing$, output $\rho^{(t)}$ as a feasible solution, and terminate the algorithm;
  Define the new weight matrix: $W^{(t+1)} = \exp[-\frac{\epsilon}{4} \sum_{\tau=1}^{t} M^{(\tau)}]$;
Claim that $\mathcal{S}_0 = \varnothing$ and terminate the algorithm;

Theorem 3.2. *For any $\epsilon > 0$, feasibility of the SDP in (18), (19), and (20) can be tested by Algorithm 3 with at most $\lceil \frac{16 \ln n}{\epsilon^2} \rceil$ queries to the search-for-violation oracle.*

Proof of Theorem 3.2. For all $j \in [m]$, denote $M_j = \frac{1}{2}(I_n - A_j)$; note that $0 \preceq M_j \preceq I \ \forall j \in [m]$. In round t, after computing the density matrix $\rho^{(t)}$, equivalently speaking, the oracle checks whether there exists a $j \in [m]$ such that $\mathrm{Tr}(M_j \rho^{(t)}) < \frac{1}{2} - \frac{a_j + \epsilon}{2}$. If not, then $\mathrm{Tr}(M_j \rho^{(t)}) \geq \frac{1}{2} - \frac{a_j + \epsilon}{2} \ \forall j \in [m]$, $\mathrm{Tr}(A_j \rho^{(t)}) \leq a_j + \epsilon \ \forall j \in [m]$, and hence $\rho^{(t)} \in \mathcal{S}_\epsilon$.

Otherwise, the oracle outputs an $M_{j^{(t)}} \in \{M_j\}_{j=1}^m$ such that $\mathrm{Tr}(M_{j^{(t)}} \rho^{(t)}) < \frac{1}{2} - \frac{a_{j^{(t)}} + \epsilon}{2}$. After $T = \lceil \frac{16 \ln n}{\epsilon^2} \rceil$ iterations, by Theorem 2.2 (taking $\delta = \epsilon/4$ therein), this matrix multiplicative weights algorithm promises that for any density matrix ρ, we have

$$\Big(1 + \frac{\epsilon}{4}\Big) \sum_{t=1}^{T} \mathrm{Tr}(M_{j^{(t)}} \rho^{(t)}) \geq \sum_{t=1}^{T} \mathrm{Tr}(M_{j^{(t)}} \rho) - \frac{4 \ln n}{\epsilon}. \tag{22}$$

If $\mathcal{S}_0 \neq \varnothing$, there exists a $\rho^* \in \mathcal{S}_0$ such that $\mathrm{Tr}(M_{j^{(t)}} \rho^*) \geq \frac{1}{2} - \frac{a_{j^{(t)}}}{2}$ for all $t \in [T]$. On the other hand, $\mathrm{Tr}(M_{j^{(t)}} \rho^{(t)}) < \frac{1}{2} - \frac{a_{j^{(t)}} + \epsilon}{2}$ for all $t \in [T]$. Plugging these two inequalities into (22), we have

$$\Big(1 + \frac{\epsilon}{4}\Big) \sum_{t=1}^{T} \Big(\frac{1}{2} - \frac{a_{j^{(t)}} + \epsilon}{2}\Big) > \sum_{t=1}^{T} \Big(\frac{1}{2} - \frac{a_{j^{(t)}}}{2}\Big) - \frac{4 \ln n}{\epsilon}, \tag{23}$$

which is equivalent to

$$\frac{\epsilon}{4} \sum_{t=1}^{T} \Big(\frac{1}{2} - \frac{a_{j^{(t)}}}{2}\Big) - \Big(1 + \frac{\epsilon}{4}\Big) \frac{T\epsilon}{2} + \frac{4 \ln n}{\epsilon} > 0 \tag{24}$$

and

$$\frac{16 \ln n}{\epsilon^2} > \frac{3 + \epsilon}{2} T + \frac{1}{2} \sum_{t=1}^{T} a_{j^{(t)}}. \tag{25}$$

Furthermore, since $\frac{1}{2} - \frac{a_{j^{(t)}}}{2} \leq \mathrm{Tr}(M_{j^{(t)}} \rho^*) \leq 1$, we have $a_{j^{(t)}} \geq -1$ for all $t \in [T]$. Plugging this into (25), we have $\frac{16 \ln n}{\epsilon^2} > (1 + \frac{\epsilon}{2})T$, and hence

$$T < \frac{16 \ln n}{\epsilon^2 (1 + \epsilon/2)} < \frac{16 \ln n}{\epsilon^2}, \tag{26}$$

contradiction! Therefore, if $\mathrm{Tr}(M_{j^{(t)}} \rho^{(t)}) < \frac{1}{2} - \frac{a_{j^{(t)}} + \epsilon}{2}$ happens for at least $\lceil \frac{16 \ln n}{\epsilon^2} \rceil$ times, it must be the case that $\mathcal{S}_0 = \varnothing$. □

4 Quantum SDP solver

We consider a quantum version of Algorithm 3. To be more specific, inside each iteration of the algorithm, the cost mainly comes from two steps:

1. Apply the oracle for searching a violated constraint;
2. Prepare the density matrix

$$\rho^{(t)} = \frac{\exp[-\frac{\epsilon}{4}\sum_{\tau=1}^{t-1} M^{(\tau)}]}{\mathrm{Tr}[\exp[-\frac{\epsilon}{4}\sum_{\tau=1}^{t-1} M^{(\tau)}]]}. \tag{27}$$

The first step is nontrivial, but you could imagine that quantumly it can be done with cost $\tilde{O}(\sqrt{m})$ by Grover search; this is formally solved by "fast quantum OR lemma" in arXiv:1710.02581v2. We very briefly introduce how to achieve the second step, with the full details presented in the same paper; because (27) is essentially a *thermal* state or a *Gibbs* state, this is known as Gibbs state preparation in quantum algorithm literatures.

Recall that in phase estimation, we are given a unitary U and a state $|\psi\rangle$ such that $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, $\mathsf{PhaseEst}(U)|\psi\rangle = |\psi\rangle|\theta\rangle$. We denote the eigen-decomposition of ρ as $\rho = \sum_l \lambda_l|\psi_l\rangle\langle\psi_l|$. Then $e^{2\pi\rho i} = \sum_l e^{2\pi\lambda_l i}|\psi_l\rangle\langle\psi_l|$. As a result,

$$\mathsf{PhaseEst}(e^{2\pi\rho i})\cdot\rho\cdot\mathsf{PhaseEst}(e^{2\pi\rho i})^\dagger = \sum_l \lambda_l|\psi_l\rangle\langle\psi_l|\otimes|\lambda_l\rangle\langle\lambda_l|. \tag{28}$$

Measuring the second register, with probability λ_l we get λ_l, and the first system collapses to $|\psi_l\rangle\langle\psi_l|$.

The quantum algorithm for Gibbs state preparation works as follows (denote $\rho = \frac{\epsilon}{4}\sum_{\tau=1}^{t-1} M^{(\tau)}$):

1. Implement the phase estimation of the unitary operator $e^{2\pi\rho i}$ on ρ (by the quantum PCA technique by Lloyd et al.); the output state is

$$\bar{\rho} := \sum_l \lambda_l|\psi_l\rangle\langle\psi_l|\otimes|\lambda_l\rangle\langle\lambda_l|. \tag{29}$$

2. We first estimate $\mathrm{Tr}[e^{-\rho}]$: Measure the second system of $\bar{\rho}$. If the output is λ, return $\lambda^{-1}e^{-\lambda}$. The expectation of this step is

$$\sum_l \lambda_l\cdot\lambda_l^{-1}e^{-\lambda_l} = \mathrm{Tr}[e^{-\rho}]. \tag{30}$$

3. Run Step 1 again. Measure the second system of $\bar{\rho}$. If the output is λ, accept the first system with probability $\frac{\delta\lambda_l^{-1}e^{-\lambda_l}}{\mathrm{Tr}[e^{-\rho}]}$ (δ is a chosen parameter). If rejected, run this step again until accepted. The expectation of this step is

$$\sum_l \lambda_l\cdot\frac{\delta\lambda_l^{-1}e^{-\lambda_l}}{\mathrm{Tr}[e^{-\rho}]}\cdot|\psi_l\rangle\langle\psi_l| = \frac{\delta e^{-\rho}}{\mathrm{Tr}[e^{-\rho}]}, \tag{31}$$

which is proportional to the Gibbs state.