Undergraduate Research Opportunity Program
(UROP) Project Report

# Computing bounds on quantum probabilities

By

Nguyen Truong Duy

Department of Computer Science

School of Computing

National University of Singapore

2012/13

Undergraduate Research Opportunity Program
(UROP) Project Report

# Computing bounds on quantum probabilities

By

Nguyen Truong Duy

Department of Computer Science

School of Computing

National University of Singapore

2012/13

Project No: U155020
Advisor: Prof. Stephanie Wehner
Deliverables:
Report: 1 Volume
Source code and Manual: Available online at `https://github.com/truongduy134/Quantum_Inequality`

**Abstract**

Quantum probabilities and Bell inequalities have a close relation with entanglement, which is the cause of many intriguing phenomina of quantum mechanics, and plays a central role in many applications, such as quantum cryptography and quantum teleportation. Finding the bounds on quamtum probabilities or Bell inequalities helps researchers to understand more about entanglement. Although there is a feasible approach to approximate the bounds using semidefinite programming in the literature, there is no library routine or package that helps researchers compute them. In this report, we first briefly report our finding of a new application of entanglement correlation in control theory as a motivation for the quantum bound problem. In particular, we demonstrate for the first time that entanglement can help to improve classical control problems. Then we propose an implementation in MATLAB based on semidefinite programming relaxations to solve the problem.

Subject Descriptors:

**Theory of computation** → **Quantum information theory**

*Mathematics of computing* → *Semidefinite programming*

Keywords:

quantum bound, quantum probability, semidefinite proramming, sum of squares, polynomial optimization, 2-prover cooperative game, non-commutative variables, Bell inequalities.

Implementation Software and Hardware:

- Matlab, Version 7.11.0.584 (R2010b) 64-bit
- SeDuMi 1.3
- YALMIP R20120109
- Server straylight.quantum.nus.edu.sg, Linux x86_64, Intel(R) Xeon(R) CPU 2.13GHz

**Acknowledgement**

I would like to thank my friends, families and advisors. Without them, I would not have been able to complete this project.

# List of Algorithms

# List of Figures

# List of Tables

# Table of Contents

# Chapter 1

# Introduction

In this chapter, we introduce the problem of computing bounds on quantum probabilites in the context of 2-prover cooperative games (Wehner, Toner, Liang, & Doherty, 2008). We also give motivations for the quantum bound problem by considering its relation with entanglement, one of the most important concepts in quantum mechanics. Before reading this chapter, we recommend the readers to go through Appendix A and Appendix B if they are not familiar with quantum computing.

## 1.1 Definition of a 2-prover Cooperative Game

In a 2-prover cooperative game, there are two players, namely Alice and Bob and a verifier. Let $S$ and $T$ be two finite sets of questions and $\pi$ be a probability distribution on $S \times T$. A verifier then chooses two questions $s \in S$ and $t \in T$ according to the distribution $\pi$. It assigns question $s$ to Alice and question $t$ to Bob. Let $A$ and $B$ be two finite sets. Upon receiving $s$, Alice has to reply to the verifier with an answer $a \in A$. Similarly, Bob replies with an answer $b \in B$. Let $V : S \times T \times A \times B \to \{0, 1\}$ be a binary function. We can view $V$ as one of the rules of the game. After receving answers from Alice and Bob, the verifier calculates the value of $V(a, b|s, t) = V(s, t, a, b)$. Alice and Bob wins if $V(a, b|s, t) = 1$ and loses otherwise. Another constraint of the game is that Alice and Bob can decide any kind of strategy beforehand; however, once the game begins, they cannot communicate with each other. We denote such a

game as $G = G(V, \pi)$.

The above desciption is the classical version of a cooperative game. In the quantum setting, we allow Alice and Bob to share an entangled quantum state $|\psi\rangle$ of their choice. To be precise, let $\mathcal{H}_A$ and $\mathcal{H}_B$ be the Hilbert spaces of Alice and Bob respectively. Then Alice and Bob can choose a quantum state $|\psi\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$. In addition, Alice and Bob can decide their sets of quantum projective measurements. Let $\mathcal{A} = \{A_s^a \mid a \in A, s \in S\}$ and $\mathcal{B} = \left\{B_t^b \mid b \in B, t \in S\right\}$ be the sets of projective measurements of Alice and Bob respectively, where $X_y^z$ denotes a projective measurement that results in an outcome $z$ corresponding to an input $y$ on a Hilbert space $\mathcal{H}$. Together with $|\psi\rangle$, $\mathcal{A}$ and $\mathcal{B}$ form Alice and Bob's strategy.

## 1.2 Overview of the Problem of computing bounds on quantum probabilities

Given a 2-prover cooperative game $G = G(V, \pi)$ as described in the previous section, we are interested in the maximum probability that Alice and Bob win the game.

In the quantum setting, the probability that Alice and Bob gives an answer $(a, b) \in A \times B$ is given by $\langle\psi| A_s^a \otimes B_t^b |\psi\rangle$ Therefore, given a shared quantum state $|\psi\rangle$ and two sets of measurement $\mathcal{A}$ and $\mathcal{B}$, the probability that Alice and Bob win the game is given by

$$\sum_{a,b,s,t} \pi(s,t)\, V(a,b \mid s,t)\, \langle\psi| A_s^a \otimes B_t^b |\psi\rangle \tag{1.1}$$

Hence, the maximum probability for Alice and Bob to win the game $G = G(V, \pi)$, denoted as $\omega^*(G)$, is defined as:

$$\omega^*(G) = \lim_{d\to\infty} \max_{\substack{|\psi\rangle \in \mathbb{C}\otimes\mathbb{C} \\ ||\,|\psi\rangle\,|| = 1}} \max_{A_s^a, B_t^b} \sum_{a,b,s,t} \pi(s,t)\, V(a,b \mid s,t)\, \langle\psi| A_s^a \otimes B_t^b |\psi\rangle \tag{1.2}$$

where $A_s^a \in \mathbb{B}(\mathcal{H}_A)$ ($\mathbb{B}(\mathcal{H}_A)$ is the set of all bounded operators on the Hilbert space $\mathcal{H}_A$) and $B_t^b \in \mathbb{B}(\mathcal{H}_B)$ for some Hilbert space $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$. In addition, the measurements $A_s^a$, $B_t^b$ must satisfy $\sum_a A_s^a = \mathbb{I}_A$, $\sum_b B_t^b = \mathbb{I}_B$, $(A_s^a)^2 = A_s^a$, and $\left(B_t^b\right)^2 = B_t^b$ (since we assume $A_s^a$ and $B_t^b$ are projective measurements).

Now our aim is to find bounds (particularly upperbounds) on $\omega^*(G)$, which is called the entangled value of $G = G(V, \pi)$. In fact, the set of 2-prover games we formulate above is merely a special subset of what called Bell inequalities. In Chapter 2, we show that we can reduce the problem of finding bounds on $\omega^*$ of a Bell inequality to finding bounds of a polynomial in non-commutative variables which are projective measurements or observables.

## 1.3 Motivations of the Quantum Bound Problem

One may wonder what is interesting or important about the 2-prover games and their upperbound values. The 2-prover games, or Bell inequalities in general have close relations with non-locality and quantum entanglement. Quantum entanglement plays a central role in applications of quantum mechanics, such as quantum teleportation, superdense coding, etc. In this section, we first mention briefly our finding of a new application of quantum entanglement in control theory (Nguyen Truong, McKague, & Wehner, 2013) to elaborate more on the importance and usefulness of entanglement. Then we conclude this chapter with the well-known example of mod-2 game to see the relation between entanglement and Bell inequalities.

### 1.3.1 Entanglement and Control Theory

Let us consider a generalized version of the discrete Witsenhausen problem (Papadimitriou & Tsitsiklis, 1986). Its circuit is illustrated in Figure (1.1). The idea of the circuit is that given a specific input, we want to decrease the ouput of the ciruit as close to 0 as possible with the help of two controllers $c_1$ and $c_2$. The controller $c_1$ has perfect information on the input, but it incurs a cost when using it, while the controller $c_2$ is cost free, but its input is passed from a noisy classical channel. The question we ask about this circuit is that how well we perform the task if we are allowed to use classical correlation and quantum correlation (i.e. quantum entanglement) between the controllers $c_1$ and $c_2$.

The circuit takes a random variable $x \in \mathbb{Z}$ with the distribution $\mathcal{P}_x$ as an input. The input is passed to a controller $c_1$ which produces an integer as its output. The $y = x + c_1(x)$ is passed to a noisy classical channel $\mathcal{N}$ such that $\mathcal{N}(s|y)$ is the probability of producing output $s \in \mathbb{Z}$

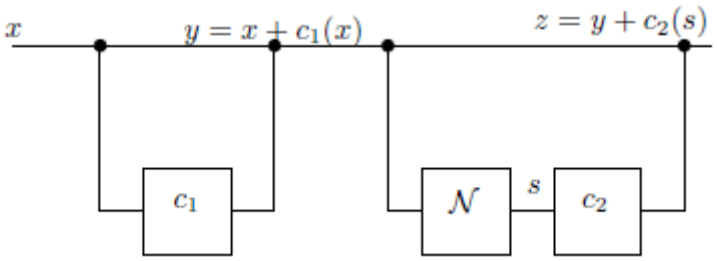


Figure 1.1: Generalized Discrete Witsenhausen Circuit

given input $y$. The ouput $s$ is then passed to the second controller, $c_2$. However, the signal is received via a noisy channel $\mathcal{N}$, where the probability of $\mathcal{N}$ outputing $s \in \mathbb{Z}$ given input $y$ is $\mathcal{N}(s|y)$. The final output of the circuit is $z = y + c_2(s)$. We define an instance of the discrete Wisenhausen problem in Figure (1.1) as $(\mathcal{N}, \mathcal{P}_x, k)$. To make the problem more interesting, we let controllers $c_1$ and $c_2$ share a resource $\rho$. We call the triple $(c_1, c_2, \rho)$ a strategy of the problem $(\mathcal{N}, \mathcal{P}_x, k)$. The strategy can be either in the quantum strategy class $SE$ or classical strategy class $SR$. For each $(c_1, c_2, \rho) \in SR$, $c_1$, $c_2$ are functions $\mathbb{Z} \times R \to \mathbb{Z}$ where $\rho \in R$ is the shared random variable. For each $(c_1, c_2, \rho) \in SE$, $\rho$ is an entangled quantum state in $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$ for some Hilbert spaces $\mathcal{H}_1$ and $\mathcal{H}_2$, $c_1$ and $c_2$ are defined by sets of projective measurements on $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively. We define the cost function of the circuit as

$$C^{\mathcal{N},P,k}(c_1, c_2, \rho) = \mathop{\mathbb{E}}_{P,\mathcal{N},\rho} \left[ k(c_1(x))^2 + (c_2(s) + x + c_1(x))^2 \right] \tag{1.3}$$

for some fixed $k \in \mathbb{R}^+$. We also define the *min cost* for the class of strategies $\Omega \in \{SR, SE\}$ to be $C_\Omega^{\mathcal{N},P,k} = \min_{(c_1,c_2,\rho)\in\Omega} \mathrm{C}^{\mathcal{N},P,k}(c_1, c_2, \rho)$. Since $SR \subset SE$, $C_{SE}^{\mathcal{N},P,k} \leq C_{SR}^{\mathcal{N},P,k}$. Can $C_{SE}^{\mathcal{N},P,k} < C_{SR}^{\mathcal{N},P,k}$, which means quantum strategy performance in the circuit is strictly better than that of classical strategies? The following theorem is an attempt to answer this question

**Theorem 1.3.1.** *For the generalized discrete Witsenhausen ciruit in Figure (1.1), a classical $\mathcal{N}_0$, a probability distribution $\mathcal{P}_0$ and a constant $k$ such that $C_{SE}^{\mathcal{N}_0,\mathcal{P}_0,k} < C_{SR}^{\mathcal{N}_0,\mathcal{P}_0,k}$*

Theorem 1.3.1 implies that for all $k \in \mathbb{R}^+$, we can find an instance (or a circuit) $(\mathcal{N}_0, \mathcal{P}_0, k)$ such that quantum entanglement performs better than randomness (or classical resource). To prove Theorem 1.3.1 we do not explicitly construct a quantum strategy that outperform any

classical strategies. Instead, we construct a family of instances $(\mathcal{N}_t, \mathcal{P}_t, k)$ such that the sequence of the minimum cost of optimal classical strategies corresponding to $t$ is unbounded. Then we give a specific example of a quantum strategy such that the sequence of its cost values corresponding to $t$ is bounded. Hence, Theorem 1.3.1 follows from the two above result. We state the two theorems we mention as follows:

**Theorem 1.3.2.** *There exists a constant $C_{SE} \in \mathbb{R}^+$ such that for all $t \geq d$, $C_{SE}^{\mathcal{N}_t,\mathcal{P}_t,k} \leq C_{SE}$*

**Theorem 1.3.3.** $\lim_{t\to\infty} C_{SR}^{\mathcal{N}_t,\mathcal{P}_t,k} = \infty$

We refer the readers to our paper, (Nguyen Truong et al., 2013), for a detailed proof of Theorem 1.3.1 and more discussion.

### 1.3.2 Mod-2 Game

We now consider a well-known instance of 2-prover games, which is called the mod-2 game.

Let $S = T = A = B = \{0, 1\}$, and $\pi(s, t) = \frac{1}{4}$ for all $s \in S, t \in T$. Let $V$ be such that $V(a, b|s, t) = 1$ if and only if $a + b \equiv st(mod 2)$. This is called the $mod - 2$ game. The optimal classical strategy is that Alice and Bob should give $a = 0$ and $b = 0$ all the time. With this strategy, they win the game with probability $\frac{3}{4}$. For a long time, it is thought that this is the best Alice and Bob can do. However, there exists a quantum strategy that beats this optimal classical winning strategy. Consider $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle|0\rangle + |1\rangle|1\rangle)$, $A_{s=0}^{a=0} = |0\rangle\langle 0|$, $A_{s=0}^{a=1} = |1\rangle\langle 1|$, $A_{s=1}^{a=0} = |+\rangle\langle +|$, $A_{s=1}^{a=1} = |-\rangle\langle -|$, $B_{t=0}^{b=0} = \frac{1}{2}(\mathbb{I} + \frac{1}{\sqrt{2}}(\sigma_x + \sigma_z))$, $B_{t=0}^{b=1} = \frac{1}{2}(\mathbb{I} - \frac{1}{\sqrt{2}}(\sigma_x + \sigma_z))$, $B_{t=1}^{b=0} = \frac{1}{2}(\mathbb{I} - \frac{1}{\sqrt{2}}(\sigma_x - \sigma_z))$, and $B_{t=1}^{b=1} = \frac{1}{2}(\mathbb{I} + \frac{1}{\sqrt{2}}(\sigma_x - \sigma_z))$. It can be verified that by applying Equation (1.1), we obtain the winning probability of this quantum strategy as $\frac{1}{2} + \frac{1}{2\sqrt{2}} > \frac{3}{4}$. Many experiments of the mod-2 games are performed, and the results are in favor of quantum mechanics over classical mechanics. In fact, it can be shown if we use a state that is not entangled, we cannot violate Bell inequalities, that is we cannot achieve better performance than that of classical mechanics. Bell inequalities can be viewed as the boundary between classical and quantum mechanics. Therefore, by knowing the upperbounds of Bell inequalties and structures of their optimizers, we can have some insights into entanglement and non-locality.

# Chapter 2

# Computing Bounds on Quantum Probabilities

In this chapter, we show that the problem of finding bounds on quantum probablities, particularly finding bounds on $\omega^*(G)$ in (1.2), can be relaxed to the problem of polynomial optimization in non-commutative variables. To do so, first we must get around the tensor product operation in $\omega^*(G)$ by introducing the field-theoretic value $\omega^f(G)$ (Wehner et al., 2008). Then we draw a connection between $\omega^*(G)$, $\omega^f(G)$, and the problem of polynomial optimization on non-commutative variables (Wehner et al., 2008). Finally, we present the semidefinite programming hierachy relaxation to solve the quantum bound problem proposed in (Pironio, Navascués, & Acín, 2010). Our implementation is based on this relaxation. Before continuing reading, we refer the readers to Appendix C for a brief introduction of semidefinite programming.

## 2.1 Field-theoretic value of $\mathrm{G} = \mathrm{G}(\pi, V)$

The field-theoretic value, denoted as $\omega^f(G)$, of a 2-prover game with classical verifier $\mathrm{G} = \mathrm{G}(\pi, V)$ is defined as

$$\omega^f(G) = \sup_{A_s'^a, B_t'^b} \left\| \sum_{a,b,s,t} \pi(s,t)\, V(a,b|s,t)\, A_s'^a B_t'^b \right\| \tag{2.1}$$

where $\|X\| = \max\left\{\lambda | \lambda \text{ is an eignvalue of } X^\dagger X\right\}$, $A_s^{'a} \in \mathbb{B}(\mathcal{H})$ and $B_t^{'b} \in \mathbb{B}(\mathcal{H})$ for the Hilbert space $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$ ($\mathcal{H}_A$ and $\mathcal{H}_B$ are Hilbert spaces of Alice and Bob respectively) satisfying $A_s^{'a}, B_t^{'b} \succeq 0$, $\sum_a A_s^{'a} = \sum_b B_t^{'b} = \mathbb{I}$ for all s, t and $\left[A_s^{'a}, B_t^{'b}\right] = 0$ ($[X, Y] = XY - YX$ is the commutator of $X$ and $Y$) for all $s \in S$, $t \in T$, $a \in A$, $b \in B$ (The sets $S$, $T$, $A$, $B$ are defined in Section 1.1).

The theoretical foundation to introduce $\omega^f(G)$ is a well-known mathematical result which is that if a Hilbert space is finite-dimensional, imposing the commutativity constraints is equivalent to demanding a tensor product structure. We will state this theorem in a rigorous form in the context of a 2-prover cooperative game as follows:

**Theorem 2.1.1.** *Let $\mathcal{H}$ be a finite-dimensional Hilbert space, and let $\left\{A_s^{'a} \in \mathbb{B}(\mathcal{H}) | s \in S\right\}$ and $\left\{B_t^{'b} \in \mathbb{B}(\mathcal{H}) | t \in T\right\}$. Then the following statements are equivalent:*

1. *For all $s \in S$, $t \in T$, $a \in A$ and $b \in B$, it holds that $\left[A_s^{'a}, B_t^{'b}\right] = 0$.*
2. *There exist Hilbert spaces $\mathcal{H}_A$, $\mathcal{H}_B$ such that $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$ and operators $A_s^a \in \mathcal{H}_A$, $B_t^b \in \mathcal{H}_B$ such that $A_s^{'a} = A_s^a \otimes \mathbb{I}_B$ and $B_t^{'b} = \mathbb{I}_A \otimes B_t^b$.*

The proof of the direction (2) $\Rightarrow$ (1) is simple because by Theorem B.2.2,

$$\begin{aligned} A_s^{'a} B_t^{'b} &= \left[A_s^a \otimes \mathbb{I}_B\right]\left[\mathbb{I}_A \otimes B_t^b\right] = \left(A_s^a \mathbb{I}_A\right) \otimes \left(\mathbb{I}_B B_t^b\right) \\ &= A_s^a \otimes B_t^b = \left(\mathbb{I}_A A_s^a\right) \otimes \left(B_t^b \mathbb{I}_B\right) = \left[\mathbb{I}_A \otimes B_t^b\right]\left[A_s^a \otimes \mathbb{I}_B\right] = B_t^{'b} A_s^{'a}. \end{aligned} \tag{2.2}$$

The proof of the other direction of Theorem 2.1.1 and a general version of Theorem 2.1.1 are presented in (Wehner et al., 2008). Also, in that paper, it is proved that

$$\omega^*(G) \leq \omega^f(G) \tag{2.3}$$

## 2.2 Finding Bounds on Quantum Probabilities and Polynomial Optimization in Non-Commutative Variables

In this section, we will draw a connection between the problem of finding bounds on quantum probabilities and the problem of polynomial optimization in non-commutative variables. In

particular, our aim is to relax the quantum bound problem to the problem of polynomial optimization.

By Equation (2.3), if we find an upperbound $\vartheta$ of $\omega^f(G)$, then $\vartheta$ is an upperbound of $\omega^*(G)$. Moreover, if there exist a quantum state and measurements in finite-dimensional Hilbert spaces such that $\vartheta = \omega^f(G)$. Then we have $\omega^*(G) = \omega^f(G) = \vartheta$ by Theorem 2.1.1. Recall that in a 2-prover cooperative game, we consider only projective measurements. Therefore, we can assume the operators $A_s^{'a}, B_t^{'b}$ in Equation (2.1) are orthogonal projectors. Hence, we have:

$$\left\| \sum_{a,b,s,t} \pi(s,t) V(a,b|s,t) A_s^{'a} B_t^{'b} \right\| = \max \left\{ \lambda | \lambda \text{ is an eigenvalue of } \sum_{a,b,s,t} \pi(s,t) V(a,b|s,t) A_s^{'a} B_t^{'b} \right\} \tag{2.4}$$

Let $\nu$ be a real number. Consider the expression:

$$q_\nu = \nu \mathbb{I} - \sum_{a,b,s,t} \pi(s,t) V(a,b|s,t) A_s^{'a} B_t^{'b} \tag{2.5}$$

If $q_\nu \succeq 0$, then clearly for all $A_s^{'a}, B_t^{'b}$, we have

$$\nu \geq \max \left\{ \lambda | \lambda \text{ is an eigenvalue of } \sum_{a,b,s,t} \pi(s,t) V(a,b|s,t) A_s^{'a} B_t^{'b} \right\} \tag{2.6}$$

Hence, $\nu \geq \omega^f(G) \geq \omega^*(G)$. Thus, $\nu$ is an upperbound of $\omega^*(G)$.

Therefore, the problem of finding a smallest upperbound (supremum) of $\omega^f(G)$ as well as $\omega^*(G)$ can be expressed as:

$$\begin{aligned}
&\text{minimize} \quad \nu \\
&\text{subject to} \quad q_\nu = \nu \mathbb{I} - \textstyle\sum_{a,b,s,t} \pi(s,t) V(a,b|s,t) A_s^{'a} B_t^{'b} \succeq 0 \\
&\qquad \textstyle\sum_{a \in A} A_s^{'a} - \mathbb{I} = 0 \text{ and } \sum_{b \in B} B_t^{'b} - \mathbb{I} = 0 \\
&\qquad \left(A_s^{'a}\right)^2 = A_s^{'a} \succeq 0 \text{ and } \left(B_t^{'b}\right)^2 = B_t^{'b} \succeq 0 \ \forall a \in A, b \in B, s \in S, t \in T \\
&\qquad \left[A_s^{'a}, B_t^{'b}\right] = 0 \ \forall a \in A, b \in B, s \in S, t \in T \\
&\qquad \left(A_s^{'a}\right)^\dagger = A_s^{'a} \text{ and } \left(B_t^{'b}\right)^\dagger = B_t^{'b} \ \forall a \in A, b \in B, s \in S, t \in T
\end{aligned} \tag{2.7}$$

Clearly, the optimization problem (2.7) is a polynomial optimization in non-commutative and Hermitian variables $A_s^{'a}, B_t^{'b}$. Hence, from now on, we turn our attention to finding the field-theoretic value $w^f(G)$, instead of $w^*(G)$. Also, our Bell inequalities are polynomials in non-commutative measurement variables, i.e. they do not contain tensor product (or Kronecker product) operation.

## 2.3 Relaxation Involving Moment Matrices

In this section, we generalize the problem of quantum probabilities problem to more than 2 parties. Then we present the semidefinite programming hierachy relaxation of Problem (2.7) involving moment matrices (Pironio et al., 2010). Our implementation in MATLAB is based on this formulation.

### 2.3.1 Moment matrices

In this section, we introduce the concept of moment matrices which is important in our solution formulation.

Let $S$ is the ordered set of monomials in 2n noncommutative variables (in particular, they are matrices) $x = (x_1, \ldots, x_n)$ and $x^\dagger = \left(x_1^\dagger, \ldots, x_n^\dagger\right)$ with coefficients in $\mathbb{C}$. Note that $x^\dagger$ is a complex conjugate tranpose of $x$. We define a linear mapping $\mathcal{L}$ as follows:

$$\begin{array}{rccc} \mathcal{L}\colon & S & \longmapsto & \mathbb{C} \\ & w & \mapsto & y_w \end{array} \tag{2.8}$$

Let us take a look at the meaning behind the introduction of the linear mapping $\mathcal{L}$. As we say in the beginning of this chapter, we use semidefinite programming to approximate solutions of quantum bound problems. Note that in the program (C.1) and (C.3), the expression we need to optimize is linear. As we will see in Subsection 2.3.2, in the quantum bound problem, we want to optimize $\mathcal{E} = \langle\psi| P |\psi\rangle = \sum p_w \langle\psi| w |\psi\rangle$ where $P = \sum p_w w$ is a polynomial, $w$ are monomials of $P$ with coefficient $p_w$. Intuitively, we will let $\langle\psi| w |\psi\rangle = y_w$ ($y_w \in \mathbb{C}$, so that $\mathcal{E}$ becomes linear in $y_w$. That is the intuition behind the linear mapping $\mathcal{L}$.

We define the moment matrix $\mathcal{M}$ as a square matrix of order $|S|$ with rows and columns indexed by elements in $S$. Also, an entry at position $(v, w)$ (where $v, w \in S$) is given by

$$\mathcal{M}(v, w) = \mathcal{L}\left(v^\dagger w\right) = y_{v^\dagger w} \tag{2.9}$$

### 2.3.2 A relaxed solution of Quantum Bound Problem

In this section, we discuss a method to relax a quantum bound problem and to transform it into a form that we can write a computer program to solve it.

First, we refomulate a generalized relaxed version of quantum bound problems. Let $\mathcal{H}$ be a Hilbert space. Let $\mathbb{B}(\mathcal{H})$ be the set of all bounded operators on $\mathcal{H}$. Let $\mathcal{S} \subset \mathbb{B}(\mathcal{H})$ be the set of Hermitian measurement operators. There are two cases in which all operators are projectors or observables. Suppose we partition $\mathcal{S}$ into $k$ disjoint subsets

$$\mathcal{S} = \bigcup_{i=1}^{k} C_i \tag{2.10}$$

such that $E_a E_b = E_b E_a$ whenever $E_a \in C_i$ and $E_b \in C_j$ and $i \neq j$. We define a function $\mathcal{P}$ as follows:

$$\begin{array}{rccc} \mathcal{P}\colon & \mathcal{S} & \longmapsto & \{1, 2, \ldots, k\} \\ & E & \mapsto & i \end{array} \tag{2.11}$$

where $i$ is the index of the partition that $E$ belongs to.

**Example 2.3.1.** *Consider the mod-2 game presented in Subsection 1.3.2.* $A_{s=0}^{a=0}$, $A_{s=0}^{a=1}$, $A_{s=1}^{a=0}$, $A_{s=1}^{a=1}$ *are Alice's measurements,* $B_{t=0}^{b=0}$, $B_{t=0}^{b=1}$, $B_{t=1}^{b=0}$, *and* $B_{t=1}^{b=1}$ *are Bob's measurement. In this example, we have two partitions corresponding to two parties (Alice and Bob) which are* $C_1 = \left\{A_{s=0}^{a=0}, A_{s=0}^{a=1}, A_{s=1}^{a=0}, A_{s=1}^{a=1}\right\}$, *and* $C_2 = \left\{B_{t=0}^{b=0}, B_{t=0}^{b=1}, B_{t=1}^{b=0}, B_{t=1}^{b=1}\right\}$. *We have* $\mathcal{P}(A_{s=1}^{a=1}) = 1$, *and* $\mathcal{P}(B_{t=1}^{b=0}) = 2$.

If all measurement operators are projectors, they take in an input and give a measurement output. Then $\mathcal{S}$ can be partitioned into $m$ disjoint subsets

$$\mathcal{S} = \bigcup_{i=1}^{m} S_i \tag{2.12}$$

where $E_u$ and $E_v \in S_i$ if they take in the same input and $\mathcal{P}\{E_u\} = \mathcal{P}\{E_v\}$. We call $S_i$ as a measurement setting with index $i$. In this case, we define another function $\mathcal{I}$ as follows:

$$\begin{array}{rccc} \mathcal{I}\colon & \mathcal{S} & \longmapsto & \{1, 2, \ldots, m\} \\ & E & \mapsto & i \end{array} \tag{2.13}$$

where $i$ is the index of the measurement setting that $E$ belongs to.

**Example 2.3.2.** *Consider the same set of measurements as specified in Example 2.3.1. We have 4 measurement inputs which are* $S_1 = \left\{A_{s=0}^{a=0}, A_{s=0}^{a=1}\right\}$, $S_2 = \left\{A_{s=1}^{a=0}, A_{s=1}^{a=1}\right\}$, $S_3 = \left\{B_{t=0}^{b=0}, B_{t=0}^{b=1}\right\}$, *and* $S_4 = \left\{B_{t=1}^{b=0}, B_{t=1}^{b=1}\right\}$. *We have* $\mathcal{I}(A_{s=1}^{a=1}) = 2$, $\mathcal{I}(B_{t=1}^{b=0}) = 4$, $\mathcal{I}(A_{s=0}^{a=0}) = 0$ *and* $\mathcal{I}(B_{t=0}^{b=1}) = 3$.

Let $P$ be a fixed polynomial such that

$$P = \sum_{i=1}^{t} p_w w \tag{2.14}$$

where $p_w \in \mathbb{R}$ and each $w$ is a monomial whose variables are operators in $\mathcal{S}$.

Then the quantum bound problem is to find

$$w^f = \sup_{\psi \in \mathcal{H},\, \mathcal{S} \subset \mathbb{B}(\mathcal{H})} \langle\psi| P |\psi\rangle \tag{2.15}$$

In addition, in the case where the equality can achieved, we want to extract the quantum state $\psi_0$ and the set of operators $\mathcal{S}_0$ that yields the optimal value. By (Pironio et al., 2010), we know that $w^f$ can be found by solving a hierachy of semidefinite programs (SDP). Now we will state the optimization problem and show how it can be transformed into a semidefinite program in two cases in which $\mathcal{S}$ contains projectors and observables respectively. Before that, we need to introduce some notations. Let $W_d$ be the set of all monomials with degree less than or equal to $d$, whose variables are operators in $\mathcal{S}$. Let $\mathcal{M}_{W_d}$ be the moment matrix with rows and columns indexed by $W_d$. Let $y_{u^\dagger v} = \mathcal{L}\left(u^\dagger v\right) = \mathcal{M}_{W_d}(u, v)$ (the entry in $\mathcal{M}_{W_d}$ at row labelled $u$ and column labelled $v$ where $u, v \in W_d$). If $t = u^\dagger v \in \mathcal{M}_{W_d}$, we denote $y_{u^\dagger v} = y_t$. Also, $\mathbb{I}$ is the identity monomial. If $\mathcal{S}$ contains observables, then consider the following optimization problem at level $d$ where $d$ is such that $2d \geq \deg(P)$:

$$\begin{aligned} \omega_d = \quad & \min && \textstyle\sum_w p_w y_w \\ & \text{subject to} && y_{\mathbb{I}} = 1 \\ & && \mathcal{M}_{W_d} \succeq 0 \\ & && (E_a)^2 = \mathbb{I} \text{ (Observable constraint)} \\ & && E_a E_b = E_b E_a \text{ if } \mathcal{P}(E_a) \neq \mathcal{P}(E_b) \text{ (Commutativity constraint)} \end{aligned} \tag{2.16}$$

Note that because $2d \geq \deg(P)$, $w \in W_{2d}$ and there exist some $u, v \in W_d$ such that $w = u^\dagger v$; hence, $y_w$ is well-defined. Also note that $P = \sum_{i=1}^{t} p_w w$ (2.14).

We now show that the optimization problem (2.16) is a semidefinite program. It is noted that the entry $y_{u^\dagger v}$ in $\mathcal{M}_{W_d}$ in (2.18) or (2.16) corresponds to $\langle\phi| u^\dagger v |\phi\rangle$ where $\phi \in \mathcal{H}$. Therefore, each constraint in (2.16) corresponds to an equality involving entries in the moment matrix. Let us illustrate this with the below example.

**Example 2.3.3.** *Consider the set of observable operators* $\{A_1, A_2, B_1, B_2\}$ *which is such that whenever* $A_i \in \{A_1, A_2\}$ *and* $B_j \in \{B_1, B_2\}$, $A_i B_j = B_j A_i$. *Consider the optimization problem (2.16) at level* $d = 2$. *We have* $A_1, B_1 \in W_2$ *and* $A_1^\dagger B_1 = B_1^\dagger A_1 = A_1 B_1 = B_1 A_1$ *(by Commutativity constraint). Hence, in the moment matrix* $\mathcal{M}_{W_d}$, *we have an inequality* $\mathcal{M}_{W_2}(A_1, B_1) = \mathcal{M}_{W_2}(B_1, A_1)$. *Or by Observable constraint, we have* $A_1^\dagger (A_1 B_2) = (A_1 A_1) B_2 = \mathbb{I} B_2 = \mathbb{I}^\dagger B_2 = B_2$. *Hence,* $\mathcal{M}_{W_2}(A_1, A_1 B_2) = \mathcal{M}_{W_2}(\mathbb{I}, A_2)$.

Thus, all the constraints of different types listed in (2.16) and their effect gives us a set of linear equalities involving entries in the moment matrix. Each linear equality itself can be expressed in the form $tr(Q\mathcal{M}_{W_d}) = 0$ where $Q$ is some matrix used to extract corresponding entries in $\mathcal{M}_{W_d}$ with suitable coefficients. Therefore, the optimization problem (2.16) can be transformed into

$$\begin{aligned} \omega_d = \quad & \min && \textstyle\sum_w p_w y_w = tr(P\mathcal{M}_{W_d}) \\ & \text{subject to} && y_{\mathbb{I}} = tr(Q\mathcal{M}_{W_d}) = 1 \\ & && \mathcal{M}_{W_d} \succeq 0 \\ & && tr(Q_i \mathcal{M}_{W_d}) = 0 \end{aligned} \tag{2.17}$$

which is a semidefinite program (see (Vandenberghe & Boyd, 1996)).

If $\mathcal{S}$ contains projectors, then consider the following optimization problem at level $d$ where $d$ is such that $2d \geq \deg(P)$:

$$\begin{aligned} \omega_d = \quad & \min && \textstyle\sum_w p_w y_w \\ & \text{subject to} && y_{\mathbb{I}} = 1 \\ & && \mathcal{M}_{W_d} \succeq 0 \\ & && E_a E_b = 0 \text{ if } \mathcal{I}(E_a) = \mathcal{I}(E_b) \text{ and } (E_a)^2 = E_a \text{ otherwise (Orthogonality constraint)} \\ & && \textstyle\sum_{E_a \in S_i} E_a = \mathbb{I} \text{ (Completeness or Identity constraint)} \\ & && E_a E_b = E_b E_a \text{ if } \mathcal{P}(E_a) \neq \mathcal{P}(E_b) \text{ (Commutativity constraint)} \end{aligned} \tag{2.18}$$

where $w$ is a monomial of $P$. Similarly to the observable case, the program (2.18) is a semidefinite program.

It can be proved (Wehner et al., 2008) that the sequence $\{\omega_d\}$ is non-increasing and it eventually converges to $w^f$, that is, $\lim_{d\to\infty} \omega_d = w^f$.

# Chapter 3

# Implementation

In this chapter, we propose a computer implementation to solve the quantum bound problems (2.18) and (2.16). In our implementation, we use SeDuMi and YALMIP which are third-party library routines and packages. SeDuMi (Sturm, ) is an excellent and efficient MATLAB solver routine for semidefinite programming and other optimization problems. YALMIP (Löfberg, 2004) is an expressive interface to SeDuMi. Therefore, in our implementation, our main task is to provide another upper-level interface that transforms the problem of quantum probabilities into a form of a semidefinite program that YALMIP accepts.

## 3.1 Data Structure for Monomials and Polynomials

Clearly, to solve the optimization problem (2.18) or (2.16), we need to deal with monomials in non-commutative variables and their operations frequently. More specifically, the monomials we deal with have a special structure, that is its set of variables can be partitioned into subsets such that any two variables from different subsets are commutative. Therefore, one of the compulsory requirements in the implementation is to find an efficient representation of such monomials which exploit such a special structure. In this section, we mention the data structure used to represent a monomial in non-commutative variables and the implementation of some important monomial operations.

### 3.1.1 Monomial Object

In our implementation, a monomial is an object with three main attributes:

- *varOrdering*: This variable represents the variable ordering of a monomial. A good representation is a list. The $ith$ element of this list represent the ordering of variables which belong to the $ith$ party (or partition). The variable ordering of each partition is itself a list of integers. Note that we map variables to integers so that they can be represented easily. Hence, varOrdering is a list of list in general. In this data structure representation, the relative ordering of variables in different partitions does not matter, which enforces the fact that variables in different paritions commute. Particularly, in our MATLAB implementation, varOrdering is a cell [1] to indicate the ordering of variables. The cell has k elements which correspond to k partitions (described in Subsection 2.3.2). Each element in the cell is an array of integers that represents the ordering of variables in each partition.

- *degree*: an integer to indicate the degree of a monomial.

- $coeff$: a real number to indicate the coefficient of a monomial.

**Example 3.1.1.** *A monomial $A_1A_2A_1B_2B_1C_2C_1C_3$, in three partitions of operators $\{A_1, A_2\}$, $\{B_1, B_2\}$ and $\{C_1, C_2, C_3\}$, has the variable ordering cell as {[1 2 1] [4 3] [6 5 7]} (Note that we map $A_1$ to 1, $A_2$ to 2, $B_1$ to 3, $B_2$ to 4, $C_1$ to 5, $C_2$ to 6, $C_3$ to 7).*

It is worth pointing out that our monomial representation specifically takes into account the commutativity constraint in the SDP (2.18) or (2.16). However, such a monomial representation can still be applied in the general case when we have only one partition. For consistency, the zero monomial has coefficient as 0, degree as -2147483648[2], and its variable ordering is an empty cell, that is, {}. The value of the degree of zero monomial is chosen to follow the mathematical convention that the zero monomial has degree which equals negative infinity. Also, the monomial

[1] A cell is a distinguished data structure in MATLAB. Roughly speaking, it is a collection that can contain variables or class instances of different types

[2] -2147483648 is the smallest 32-bit integer that MATLAB can represent

of degree 0 has variable ordering as a cell that contains k empty arrays where k is the number of partitions. For example, suppose the set of operators has 3 partitions corresponding to 3 parties, then the identity monomial $\mathbb{I}$ has the variable ordering as {[] [] []}.

### 3.1.2 Arithmetics with monomials

Monomial addition is only applicable to two monomials that have the same variable ordering. In this case, the monomial addition is simply the addition of coefficients.
Monomial multiplication plays a more important role in the package. With the mentioned data structure for monomial representation, monomial multiplication simply involves a coefficient multiplication and a concatenation of arrays of corresponding paritions. Note that the degree can be recalculated by counting the total number of integers (which are variable identifiers) in the list (or variable ordering).

**Example 3.1.2.** *Consider three partitions of observables* $\{A_1, A_2\}$, $\{B_1, B_2\}$ *and* $\{C_1, C_2\}$. *Consider two monomials* $3A_1A_2B_2B_1C_1$ *and* $4A_1B_2$ *whose representations of variable ordering are {[1 2] [4 3] [5]} and {[1] [4] []} respectively. The multiplication of the two monomials yields a new monomial* $12A_1A_2A_1B_2B_1B_2C_1$ *whose ordering is {[1 2 1] [4 3 4] [5]}.*

In the implementation of monomial multiplication, we also take into account the orthogonality or observable constraints of the variables. For instance, let us consider the same set of observables in Example 3.1.2. Consider the variable ordering {[1]} corresponds to the monomial $A_1$. Then the multiplication of {[1]} with itself should give {[]} as a result, which is a variable representation corresponding to $\mathbb{I}$. It is because $A_1 \times A_1 = \mathbb{I}$. The work of taking these kinds of contrains into account can be done by simply removing consecutive duplicates in each array in the cell of variable ordering. In case the variables are projectors and if there are two consecutive different variables belonging to the same measurement setting, that monomial becomes zero.

### 3.1.3 Adjoint of a monomial

The monomial representation described in Subsection 3.1.1 also simplifies the process of computing the conjugate transpose (adjoint) of a monomial provided the variables are Hermitian.

We simply reverse the list representing the variable ordering in each partition.

**Example 3.1.3.** *A monomial $A_1A_2B_2B_1$ in two partitions $\{A_1, A_2\}$ and $\{B_1, B_2\}$ has the variable ordering cell as {[1 2] [4 3]} (Note that we map $A_1$ to 1, $A_2$ to 2, $B_1$ to 3, $B_2$ to 4). Its conjugate transpose is $(A_1A_2B_2B_1)^\dagger = B_1^\dagger B_2^\dagger A_2^\dagger A_1^\dagger = B_1B_2A_2A_1 = A_2A_1B_1B_2$ which has the variable ordering cell as {[2 1] [3 4]}.*

The following is the pseudocode to find the adjoint of a monomial.

**Algorithm 1:** FindAdjointMonomial($M$)

**input** : $M$: a monomial object. It is assumed that $M$ is a monomial of Hermitian variables.

**output**: *AdjointM*: the adjoint monomial of $M$.

```
Let AdjointM be a new monomial object
AdjointM.degree ← M.degree
AdjointM.coeff ← M.coeff
Let N be the number of partitions.
for index ← 1 to N do
    AdjointM.varOrdering(index) ← ReverseList(M.varOrdering(index))
return AdjointM
```

### 3.1.4 Polynomial Object

A polynomial in non-commutative variables is in fact a sum of monomials in non-commutative variables. Therefore, a natural representation of this type of polynomials is a list of monomials in non-commutative variables. In our implementation, a polynomial is an object with the following attributes:

- *listMonomial*: list of monomials in non-commutative variables.
- *degree*: the degree of the polynomial.
- *varType*: 0 (if variables are projectors), 1 (if variables are observables)

- $\mathcal{P}$, $\mathcal{I}$: functions described in Subsection 2.3.2. They are used to tell which partition or which measurement setting each variable belongs to.

- *varProperties*: a list of $N$ partitions. If all variables are observables, each partition is simply a list of variable identifiers belonging to that partition. If variables are projectors, each parition is a list of measurement settings; each measurement setting is a list of variable identifiers.

Polynomial addition and multiplication can be implemented using straightforward algorithms.

## 3.2 High-level Description of Algorithm

In this section, we give a pseudocode to present a high-level description of the algorithm we use in implementing the package. See Algorithm 2 on page 18. Details of important steps in the algorithm will be explained in subsequent sections.

## 3.3 Generating a list of monomials

Based on the optimization problem (2.18) or (2.16), we see that we need to generate the list of monomials $W_d$ in order to start the SDP at level $d$. For example, again consider the set of observable operators $\{A_1, A_2, B_1, B_2\}$ which is such that whenever $A_i \in \{A_1, A_2\}$ and $B_j \in \{B_1, B_2\}$, $A_i B_j = B_j A_i$. Suppose we want to start the SDP at level 2. Then we need to generate $W_2 = \{\mathbb{I}, A_1, A_2, B_1, B_2, A_1A_2, A_2A_1, B_1B_2, B_2B_1, A_1B_1, A_1B_2, A_2B_1, A_2B_2\}$. Note that again we take into account the orthogonality or observable, and commutativity constraint of the variables. That is, in the former example, the monomial $A_1^2$ should not appear in $W_2$ because $A_1^2$ can be furthur simplified to $\mathbb{I}$.

In our data structure representation of monomials, variable ordering of a parition is indepdent of the ordering of another partition. Therefore, to generate a monomial of degree $d$, we generate the variable ordering of each partition with suitable length, and then concatenate them to form a monomial. The elaborated pseudocode for line 2 is provided in Algorithm 3 on page 19. The part of generating variable ordering in each partition, and the part of concatenating

**Algorithm 2:** FindQuantumBound(PolyData, $d$, SolverSetting)

**input** :

- PolyData: an object that represents a polynomial.
- $d$: the level of the SDP. It is assumed that $2 * d \geq PolyData.degree$
- SolverSetting: a data structure to indicate information about the SDP solver to use.

**output**:

- OptimalVal: the optimal upperbound of the polymonial given.
- RunLog: information about the execution returned by underlying SDP solver
- $\mathcal{M}_{W_d}$: the resulted moment matrix of this SDP.
- mapLocation: A data structure that maps a monomial to location in momentMatrix
- $W_d$: list of monomials that are used to index columns and rows of $\mathcal{M}_{W_d}$

`// STEP 1:`

*Generate a list of monomials of degree less than or equal to d, i.e.* $W_d$

`// ADDITION STEP: Declaration of essential variables:`

$lenMonoList \leftarrow$ `LengthList(`$W_d$`)`

`// 1) Moment matrix (We use the object of type sdpvar provided by YALMIP)`

`// The instruction declares a complex hermitian square matrix of order` $lenMonoList$ `whose entries are variables`

$\mathcal{M}_{W_d} \leftarrow$ `sdpvar(`*lenMonoList, lenMonoList, 'hermitian', 'complex'*`)`

`// 2) A vector of constraints we put on the moment matrix and its entries. Initially, we need the moment matrix to be positive semidefinite and the entry` $\mathcal{M}_{W_d}(1,1)$ `(at row 1, column 1) to be 1`

$constraintSet \leftarrow [\mathcal{M}_{W_d} \succeq 0, \mathcal{M}_{W_d}(1,1) == 1]$

`// 3) Initialize the data structure that map monomials to locations`

$mapLocation \leftarrow EMPTY$

`// STEP 2: It involves three tasks described below`

*Establishing relations between entries in the moment matrix based on orthogonality constraints, commutativity constraints, projector / observable constraints). Add these constraints to constraintSet. Update mapLocation*

`// STEP 3:`

*Deal with identity contraints (only applicable to the case variables are projectors). Add additional constraints (if any) to constraintSet*

`// STEP 4: Describe the variable we want to maximize`

$OptimalVal = 0$

**foreach** *monomial* $m$ *in* $PolyData.ListMono$ **do**

  $OptimalVal = OptimalVal + \mathcal{M}_{W_d}(mapLocation.getData(m))$

`// Call underlying SDP solver indirectly through YALMIP function solvesdp`

$RunLog \leftarrow$ `solvesdp(`*constraintSet, -OptimalVal, SolverSetting*`)`

**return** $[OptimalVal \quad RunLog \quad \mathcal{M}_{W_d} \quad mapLocation \quad W_d]$

**Algorithm 3:** Elaborate Line 1 (Step 1) in Algorithm 2

**Summary of some variables used in Algorithm 2**:

- PolyData: an object that represents a polynomial that we want to find its upperbound. One of its attributes is:
  - *varProperties*: a list of $N$ partitions. If all variables are observables, each partition is simply a list of variable identifiers belonging to that partition. If variables are projectors, each parition is a list of measurement settings; each measurement setting is a list of variable identifiers.
- $d$: The level of the SDP, and also the maximum degree of monomials in $W_d$. It is assumed that $2 * d \geq PolyData.degree$
- $W_d$: The list of monomials of degree less than or equal to $d$.

1 *Add the identity monomial* $\mathbb{I}$ *to* $W_d$
2 $N =$ `length(`*PolyData.varProperties*`)`
3 **for** *degree* $\leftarrow 1$ **to** $d$ **do**
4  *Partition of degree into* $N$ *non-negative parts. Let* $L$ *be such a list of partitions.*
5  **foreach** *partition* $P$ *in* $L$ **do**
6  **for** $i \leftarrow 1$ **to** $N$ **do**
7  $LV(i) \leftarrow$ `GetListOfVarOrListOfMeasurementSettingsInPartition(`*PolyData.varProperties, i*`)`
 `// Generate a list of variable ordering in the` $i$`-th parition of length` $P(i)$
8  $LM(i) \leftarrow$ `GenerateVarOrdering(`*LV(i), P(i)*`)`
 `// Concatenate each list in` $LM(1), \ldots, LM(N)$ `to form a variable ordering`
9  $VO \leftarrow$ `ConcatenateVarOrdering(`*LM(1), ..., LM(N)*`)`
10  **foreach** *ordering* $o$ *in* $VO$ **do**
 `// Create a monomial with the specified ordering and degree, with coefficient is 1`
11  $m \leftarrow$ `MonomialConstructor(`*o, degree, 1*`)`
12  *Add* $m$ *to* $W_d$

each set $\{o_1, \ldots, o_N\}$ (where $o_i$ is a variable ordering of partition $i$) can be done recursively. To optimize the running time, we use an iterative algorithm. We refer the readers to (Knuth, 2005) for more discussion on algorithms to generate permutations.

## 3.4 Establishing relations between entries in the moment matrix

As we mentioned in Subsection 2.3.2, to solve the quantum bound problem, we must transform the optimization problem (2.18) or (2.16) into a semidefinite program of the form (2.17) by establishing all possible equaltities between entries in the moment matrix. In this section, we discuss how we deal with this issue in our software package.

### 3.4.1 Hashing

An essential step to establish relations (or equalities particularly) between entries in the moment matrix is that we must retrieve the location of the entry $y_\omega$ corresponding to a monomial $\omega$ fast. This can be done using hashing. In general, we need a hash table that maps a monomial $\omega$ to the location of the entry $y_\omega$ in the moment matrix. Now then, we discuss how we apply the general approach mentioned to our implementation in MATLAB. The only data structure that supports mapping in MATLAB is the class containers.Map. There is a constraint on the ability of containers.Map. The key we pass to the data structure can only be integers, doubles or an array of characters. As a result, we have a function to convert the variable ordering of a monomial into an array of characters, which we call a string of variable ordering.

**Example 3.4.1.** *A variable ordering {[1 2 1] [5 4] [6] []} is converted into a string "1*1*1/5*4/6//".*

Therefore, our containers.Map object takes a string representing the variable ordering as a key, and a pair of indices (to indicate the position of an entry in the moment matrix) as the associated data. Note that we map monomials in the list $W_d$ to the set of integers $\{1, 2, ..., |W_d|\}$ for easy matrix index referencing.

### 3.4.2 Dealing with orthogonality, observable and commutativity constraints

We present the pseudocode of the algorithm to establish orthogonality, observable and commutativity constraint. See Algorithm 4 on page 22. Note that in the pseudocode, for general purposes, mapLocation is an abstract data structure that maps a monomial $\omega$ to the location of the entry $y_\omega$ in the moment matrix. We do not enforce any explicit implementation on mapLocation. A detailed explanation of the pseudocode is given below.

It is noted that we take into account the orthogonality, observable and commutativity constraints in the monomial multiplication. Therefore, we only need to go through the moment matrix once to establish all equalities regarding orthogonality, observable and commutativity constraints. We now refer the containers.Map as a hash table for general purposes. Intially, let the hash table be empty. Let L be the list of monomials we generate. Then for each entry $(row, col)$ in the moment matrix, we then compute a monomial $\omega$ which is the product of the conjugate transpose of L[row] and L[col], i.e. $\omega = \big(L[row]\big)^\dagger \big(L[col]\big)$. Let s be the string of variable ordering of $\omega$. If the hash table contains s, then we take the associated data $(hashRow, hashCol)$ of s from the the hash table. Then we have an equality $\mathcal{M}(row, col) = \mathcal{M}(hashRow, hashCol)$. Otherwise, we add s and its associated data $(row, col)$ to the hash table.

### 3.4.3 Dealing with identity / completeness constraints

Identity (or completeness) constraints are not applicable to observable variables. For the projector case, they are the most tricky constraints to deal with. Before discussing how to deal with this type of constraints, let us consider a concrete example of identity contraints.

**Example 3.4.2.** *Consider 8 projectors in two partitions* $\{A_0^0, A_0^1, A_1^0, A_1^1\}$ *and* $\{B_0^0, B_0^1, B_1^0, B_1^1\}$. *Also,* $\{A_0^0, A_0^1\}$, $\{A_1^0, A_1^1\}$, $\{B_0^0, B_0^1\}$ *and* $\{B_1^0, B_1^1\}$ *form 4 measurement settings. Consider a monomial* $\omega = A_0^0 A_1^1 A_0^0 B_1^0$. *In this example, we find all identity constraints regarding* $A_0^0$ *and* $\omega$. *Clearly, each instance of* $A_0^0$ *in* $\omega$ *gives us an identity constraint. To establish an identity constraint, we replace an instance of* $A_0^0$ *by other projectors in the same measurement setting. In this example, we replace* $A_0^0$ *by* $A_0^1$. *Then the identity constraint is an equality where the left-hand side is the sum of* $\omega$ *and other monomials obtained by replacing* $A_0^0$ *by* $A_0^1$, *and the*

**Algorithm 4:** Elaborate Line 6 (Step 2) in Algorithm 2

**Summary of variables used in Algorithm 2**:

- PolyData: an object that represents a polynomial that we want to find its upperbound.
- $d$: The level of the SDP. It is assumed that $2 * d \geq PolyData.degree$
- $W_d$: The list of monomials of degree less than or equal to $d$.
- $\mathcal{M}_{W_d}$: the moment matrix
- mapLocation: A data structure that maps a monomial to its location in the moment matrix. It has the following functions:
  - getData(K): returns data associated with the key K
  - containKey(K): returns TRUE if the mapLocation contains the key K and its data. It returns FALSE otherwise.
  - add(K, D): adds the new key K and its associated D to mapLocation.
- constraintSet: A vector of constraints we put on the moment matrix and its entries.

```
1  for row ← 1 to lenMonoList do
2      for col ← 1 to lenMonoList do
           // FindAdjointMonomial(A) returns the adjoint of a matrix A
3          adjointRow ← FindAdjointMonomial(W_d(row))
           // MultiplyMono(A, B, P, I) returns the result of A * B (A times
              B). It needs the arguments P and I described in Subsection 2.3.2
              to simplify the result
4          mulResult ←
           MultiplyMono(adjointRow, W_d(col), PolyData.varType, PolyData.P, PolyData.I)

5          if IsZeroMonomial(mulResult) == TRUE then
               // The instruction below adds a new constraint to the set of
                  existing constraints
6              constrainSet = [constrainSet, M_{W_d}(row, col) == 0]
7          else
8              if mapLocation.containKey(mulResult) == TRUE then
9                  (rowData, colData) = mapLocation.getData(mulResult)
10                 constrainSet =
                   [constrainSet, M_{W_d}(row, col) == M_{W_d}(rowData, colData)]
11             else
12                 mapLocation.add(mulResult, (row, col))
```

*right-hand side is a monomial obtained by removing $A_0^0$ (due to identity effect). Hence, in this example, we have two identity constraints:*

$$\begin{aligned} A_0^0A_1^1A_0^0B_1^0 + A_0^1A_1^1A_0^0B_1^0 &= (A_0^0 + A_0^1)A_1^1A_0^0B_1^0 \\ &= \mathbb{I}A_1^1A_0^0B_1^0 \\ &= A_1^1A_0^0B_1^0 \end{aligned} \tag{3.1}$$

*and*

$$\begin{aligned} A_0^0A_1^1A_0^0B_1^0 + A_0^0A_1^1A_0^1B_1^0 &= A_0^0A_1^1(A_0^0 + A_0^1)B_1^0 \\ &= A_0^0A_1^1\mathbb{I}B_1^0 \\ &= A_0^0A_1^1B_1^0 \end{aligned} \tag{3.2}$$

Following the method mentioned in the example 3.4.2, one may come up with a brute-force algorithm to enumerate all identity constraints. However, this process is computationally expensive. In fact, there is a simple yet clever method to get around generating all identity constraints. Clearly, we see that one variable in each full measurement setting can be express as a polynomial expression of the others in the same setting. Intuitively, substituting these expressions does not change the value we want to optimize. Moreover, when missing one measurement in a setting, an identity constraint cannot be established. We next formulate this idea more rigorously.

Let us write the polynomial we optimize in the SDP (2.18) as $\sum_w p_w y_w = P(S_1, \ldots, S_m)$, which is a polynomial in $\sum_{i=1}^m |S_i|$ non-commutative projector variables. Each $S_i$ is a measurement setting such that $\sum_{E_a \in S_i} E_a = \mathbb{I}$. Because $|S_i| \geq 2$, let us pick from each $S_i$ a projector $E_{a_0}$. Let $S_i' = S_i \setminus \{E_{a_0}\}$. Then $E_{a_0} = \mathbb{I} - \sum_{S_i'} E_a$. Substituting all the $E_{a_0}$ from $m$ measurement

settings, we obtain a polynomial $H(S_1^{'}, \ldots, S_m^{'})$. The SDP (2.18) is now equivalent to

$$\begin{aligned}
\omega_d = \quad & \min \quad && H(S_1^{'}, \ldots, S_m^{'}) \\
& \text{subject to} \quad && y_{\mathbb{I}} = 1 \\
& && \mathcal{M}_{W_d} \succeq 0 \\
& && E_a E_b = 0 \text{ if } \mathcal{I}(E_a) = \mathcal{I}(E_b) \text{ and } (E_a)^2 = E_a \text{ otherwise (Orthogonality constraint)} \\
& && E_a E_b = E_b E_a \text{ if } \mathcal{P}(E_a) \neq \mathcal{P}(E_b) \text{ (Commutativity constraint)} \\
& && \textstyle\sum_{E_a \in S_i^{'}} E_a \preceq \mathbb{I}
\end{aligned} \tag{3.3}$$

with incomplete measurement settings $S_1^{'}, \ldots, S_m^{'}$.

Note that in the SDP (3.3), there is no identity constraints. It is because projectors in each $S_i^{'}$ do not establish identity equality. Instead, for each measurement setting, we have a constraint that $\sum_{E_a \in S_i^{'}} E_a \preceq \mathbb{I}$. It is because $\mathbb{I} - \sum_{E_a \in S_i^{'}} E_a = E_{a_0}$ which is a projector itself; hence it is positive semidefinite. Therefore, $\mathbb{I} - \sum_{E_a \in S_i^{'}} E_a \succeq 0$, which means $\sum_{E_a \in S_i^{'}} E_a \preceq \mathbb{I}$.

Transforming the original polynomial into the form $H(S_1^{'}, \ldots, S_m^{'})$ is much faster than enumerating all identity constraints because it involves only polynomial addition and multiplication, and the number of terms of a polynomial in practice is relatively small. Furthermore, this trick reduces the number of variables; hence, it also reduces the number of monomials. It also adds less contraints by ignoring identity contraints. Therefore, the size of the SDP is reduced dramatically in general, and the running time is better.

In our package, the part of transforming the original polynomial is done in the input preprocessing section, before the main function *FindQuantumBound* is called.

# Chapter 4

# Stopping criteria and Extraction of quantum state and measurements

Algorithm 2 produces the result of the semidefinite program at level $d$, $\omega_d$. However, we do not know if this result $\omega_d$ is actually $w^f$ (the supremum or optimal quantum bound). Therefore, it is desirable to know if the result of a semidefinite program at a specific level is optimal, so that we can stop at that level. Also, in case we obtain the optimal quantum bound at a specific level, we want to extract the optimal state and measurements to examine their structures. In this chapter, we present stopping criteria and extraction of optimal state and measurements for the bipartite case, i.e. when there are only two parties (or partitions) based on the results in (Wehner, 2008) and (Pironio, Navascués, & Acín, 2008). We then discuss how these algorithms are implemented in our package, and point out some numerical round-off errors one may encounter when implementing these algorithms, especially in the case variables are projectors.

## 4.1 Observable case

In this section, we consider the case in which measurements are observables. This turns out to be a simple case because $\omega_1$ is optimal, i.e. we obtain the optimal bound at the very first level of the semidefinite program hierachy. The construction of optimal state and measurement can be done based on Tsirelson's work.

### 4.1.1 Tsirelson's Theorem

In this subsection, we give a definition of Clifford algebra, state and prove a part of Tsirelson's Theorem. These provide a theoretical foundation for our algorithm of constructing optimal state and observable measurements later.

The generators of Clifford algebra $\Gamma_1$, $\Gamma_2$, ..., $\Gamma_N$ where $N = 2n$ are obtained by the following procedure:

$$\begin{aligned} \Gamma_{2j-1} &= \sigma_y^{\otimes(j-1)} \otimes \sigma_x \otimes \mathbb{I}^{\otimes(n-j)} \\ \Gamma_{2j} &= \sigma_y^{\otimes(j-1)} \otimes \sigma_z \otimes \mathbb{I}^{\otimes(n-j)} \end{aligned} \tag{4.1}$$

where $j = 1, 2, \ldots, n$ and $A^{\otimes(n)}$ is the $nth$ Kronecker power of a matrix A. In case $N = 2n+1$, $\Gamma_{2n+1} = i\Gamma_1\Gamma_2 \ldots \Gamma_{2n}$. The generators of Clifford algebra have the following properties: (Wehner, 2008)

$$\begin{aligned} \Gamma_i^2 &= \mathbb{I} \quad \text{for } i = 1, \ldots, N \\ \Gamma_i\Gamma_j &= -\Gamma_j\Gamma_i \end{aligned} \tag{4.2}$$

Now we are ready to state and give a proof of Tsirelson's Theorem. Note that in the scope of this paper, we only state one direction of the original Tsirelson's Theorem. We refer the readers to (Wehner, 2008) for its full version.

**Theorem 4.1.1** (Tsirelson's Theorem)**.** *Let $|x_s\rangle$ and $|y_t\rangle \in \mathbb{R}^N$ be unit vectors. Let $|\psi\rangle \in \mathcal{H}^a \otimes \mathcal{H}^b$ be a maximally entangled state where $\dim(\mathcal{H}^A) = \dim(\mathcal{H}^B) = 2^{\lfloor N/2 \rfloor}$. Then for $s = 1, \ldots, n$ and $t = 1, \ldots, m$, we can find observables $X_s$ on $\mathcal{H}^A$ and $Y_t$ on $\mathcal{H}^B$ such that*

$$\langle x_s|y_t\rangle = \langle\psi| X_s \otimes Y_t |\psi\rangle \tag{4.3}$$

*Proof.* Let $x_s = (x_s^1, \ldots, x_s^N)$ and $y_t = (y_t^1, \ldots, y_t^m)$ where $x_s^i, y_t^j \in \mathbb{R}$. Let $\Gamma_1, \Gamma_2, \ldots, \Gamma_N$ be generators of Clifford algebra of order N. Define

$$X_s = \sum_{i=1}^{N} x_s^i \Gamma_i^T \qquad\qquad Y_t = \sum_{i=1}^{N} y_t^i \Gamma_i \tag{4.4}$$

Let $|\psi\rangle = \frac{1}{\sqrt{d}} \sum_{k=0}^{d-1} |k\rangle |k\rangle$ be a maximally entangled state with $d = 2^{\lfloor N/2 \rfloor}$. Then by

$$\begin{aligned} \langle\psi| X_s \otimes Y_t |\psi\rangle &= \tfrac{1}{d} Tr(X_s^T Y_t) && \text{(by Theorem B.2.3)} \\ &= \tfrac{1}{d} \sum_{i=1}^{N} \sum_{j=1}^{N} x_s^i y_t^j Tr(\Gamma_i \Gamma_j) \\ &= \tfrac{1}{d} \sum_{i=1}^{N} x_s^i y_t^i Tr(\mathbb{I}) && \text{(by Equations 4.2)} \\ &= x_s y_t \end{aligned} \tag{4.5}$$

Also, we have

$$X_s^2 = \sum_{i=1}^{N} \sum_{j=1}^{N} x_s^i x_s^j \Gamma_i^T \Gamma_j^T = \sum_i x_s^i x_i^2 \mathbb{I} = \mathbb{I} \tag{4.6}$$

since $x_s$ is a unit vector. Similarly, we have $Y_t^2 = \mathbb{I}$. Hence $X_s$ and $Y_t$ are observables satisfying the properties. □

### 4.1.2 Extraction of Optimal State and Observable Measurements

In this subsection, we apply Theorem 4.1.1 to construct the optimal state and observable measurements. The pseudocode of the algorithm will be provided.

Stopping criteria are not needed for the observable case. It can be shown that the semidefinite hierachy converges to the optimal value at the very first level, i.e. $\omega^f = \omega_d$. Also, the resulted moment matrix is a real matrix.

Let $A_1$, $A_2$, ..., $A_n$ and $B_1$, $B_2$, ..., $B_m$ be observables of two parties respectively. Then the moment matrix at level 1 is $\mathcal{M}_{W_1}$ where $W_1 = \{\mathbb{I}, A_1, \ldots, A_n, B_1, \ldots, B_m\}$. Because $\mathcal{M}_{W_1}$ is Hermitian and positive semidefinite, we can find its Cholesky decomposition (See Section B.3 for a discussion about how to find such a decomposition). That is, we have an upper triangular matrix $M$ such that

$$M = \begin{pmatrix} |\xi_{\mathbb{I}}\rangle & |\xi_{A_1}\rangle & \ldots & |\xi_{A_n}\rangle & |\xi_{B_1}\rangle & \ldots & |\xi_{B_m}\rangle \end{pmatrix} \tag{4.7}$$

and $\mathcal{M}_{W_1} = M^\dagger M$. Because $\mathcal{M}_{W_1}$ is real, we can construct $M$ such that it is a real matrix. Therefore, $\xi_P$ where $p \in W_1$ are real vectors. Also, because $\forall P \in W_1$, $P^2 = \mathbb{I}$ and $P^\dagger = P$, we have $\mathcal{M}_{W_1}(P, P) = \mathcal{M}_{W_1}(\mathbb{I}, \mathbb{I}) = 1$ (by Algorithm 2). Since $\mathcal{M}_{W_1}(P, P) = \langle \xi_P | \xi_P \rangle$, $\langle \xi_P | \xi_P \rangle = 1$; hence, $|\xi_P\rangle$ is a unit vector $\forall P \in W_1$. Applying Tsirelson's theorem and the procedure presented

in its proof to the vectors $\xi_{A_1}, \ldots, \xi_{A_n}, \xi_{B_1}, \ldots, \xi_{B_m}$, we get a maximal entangled state $|\psi\rangle$ as an optimal state and observables $A_1, \ldots, A_n, B_1, \ldots, B_m$ such that

$$\langle\psi| A_i \otimes B_j |\psi\rangle = \left\langle \xi_{A_i} | \xi_{B_j} \right\rangle = \mathcal{M}_{W_1}(A_i, B_j) \tag{4.8}$$

for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$. Let $A_i^{'} = A_i \otimes \mathbb{I}_A$ and $B_i^{'} = \mathbb{I}_B \otimes B_i$. Then $\langle\psi| A_i^{'} B_j^{'} |\psi\rangle = \mathcal{M}_{W_1}(A_i, B_j)$. Hence, we find a quantum state and sets of observables such that $\omega_d = \omega^f$. The whole procedure we have presented is summarized in Algorithm 5 on page 29.

The diffcult part in implementing Algorithm 5 is finding a Cholesky decomposition of $\mathcal{M}_{W_d}$. In MATLAB, *chol* is a built-in function to find Cholesky decomposition. Unfortunately, *chol* requires the input matrix to be positive definite while our moment matrix is positive semidefinite. Also, the moment matrix computed by some solvers (SeDuMi for instance) is not even positive semidefinite in the mathematical sense. The moment matrix $\mathcal{M}_{W_d}$ just needs to satisfy that $\min\{|\lambda| : \lambda \text{ is an eigenvalue of } M\} \leq \epsilon$ for some very small $\epsilon > 0$. Therefore, one may need to implement his own function to find a Cholesky decomposition of $\mathcal{M}_{W_d}$. A pseudocode to do that is provided in Algorithm 8 on page B-3 in Appendix B.

## 4.2 Projector case

In (Pironio et al., 2010), for the two-partition inequality, the rank loop method is introduced to detect whether the sequence $\{\omega_d\}$ converges to $w^f$ at the finite level $d$. Also, in their paper, they show how to extract the quantum state and measurements that yield the optimal bound when a rank loop occurs. We will summarize these results and provide pseudocodes of our implementation of these features.

### 4.2.1 Rank loop

In this subsection, we define a rank loop, state an important result regarding it and provide a pseudocode to check if rank loop occurs.

Let $S_1^{'}, S_2^{'}, \ldots, S_m^{'}$ be incomplete measurement settings[1] of the first party, and $T_1^{'}, \ldots, T_n^{'}$

[1] As presented in Subsection 3.4.3, in a complete measurement setting, the sum of its projectors equals the identity. If we remove any projector from the setting, we obtain an incomplete measurement setting.

**Algorithm 5:** ExtractOptimalStateAndObservable($\mathcal{M}_{W_d}, listVar, mapLocation$)

**input** :

- $\mathcal{M}_{W_d}$
- listVar: A list of variables (or singleton monomials)
- mapLocation: A data structure that maps a monomial to its location in the moment matrix. This variable is returned by FindQuantumBound

**output** :

- optimalState: Optimal state
- listOptimalObservable: a list of matrix representations of optimal observables.

**assumption**:

- There are only 2 parties.

```
// Note that M_{W_d} is a square matrix
[numRow  numCol] ← size(M_{W_d})
N ← numRow
choleskyDecomposition ← getCholeskyDecomposition(M_{W_d})
[Γ_1  ...  Γ_N] ← getCliffordGenerators(N)
[numRow  numCol] ← size(Γ_1)
order ← numRow
Initiate listOptimalObservable to be a list of size length(listVar)
for varIndex ← 1 to length(listVar) do
    // Following Algorithm 4, we expect rowPosition = 1 and colPosition is
       the position of listVar(varIndex) in the ordered list of monomials
       indexing columns and rows of M_{W_d}
    [rowPosition  colPosition] ← mapLocation.getData(listVar(varIndex))
    listOptimalObservable(i) = getZeroMatrixOfSize(size(Γ_1))
    for i ← 1 to N do
        if belongToFirstParty(listVar(i)) == TRUE then
            listOptimalObservable(i) ← listOptimalObservable(i) +
            choleskyDecomposition(i, colPosition) * getTranspose(Γ_i)
        else
            listOptimalObservable(i) ←
            listOptimalObservable(i) + choleskyDecomposition(i, colPosition) * Γ_i
    // I_n is the identity matrix of order n
    if belongToFirstParty(listVar(i)) == TRUE then
        listOptimalObservable(i) = listOptimalObservable(i)⊗I_order
    else
        listOptimalObservable(i) =I_order ⊗ listOptimalObservable(i)
// optimalState is a maximally entangled state
optimalState ← getZeroMatrixOfSize([order  1])
for j ← 1 to order do
    optimalState ← optimalState+I_order(:, j)⊗I_order(:, j)
return [optimalState  listOptimalObservable]
```

be incomplete measurement settings of the second party. Denote $S^{'} = \bigcup_{i=1}^{m} S_i^{'}$ and $T^{'} = \bigcup_{i=1}^{n} T_i^{'}$. Let $\mathcal{M}_{W_d}$ be a moment matrix of the SDP at level $d \geq 2$. Let $E_a \in S^{'}$ and $E_b \in T^{'}$ be arbitrary. Let $SM_{E_a, E_b}$ be the set of monomials of length $d-1$, and monomials of length at most $d$ in which $E_a$, $E_b$ are the leading variables in the variable orderings of two parties respectively. Clearly, $SM_{E_a, E_b} \subset W_d$. Define $\mathcal{M}_{W_d}^{E_a, E_b}$ to be the submatrix of $\mathcal{M}_{W_d}$ indexed by $SM_{E_a, E_b}$. If $rank(\mathcal{M}_{W_d}^{E_a, E_b}) = rank(\mathcal{M}_{W_d})$ for all $E_a \in S^{'}$ and $E_b \in T^{'}$, then we say $\mathcal{M}_{W_d}$ has a rank loop. Note that the concept of rank loop does not apply to the SDP at level 1. Below is an important theorem of rank loop.

**Theorem 4.2.1.** *Let $\omega_d$ and $\mathcal{M}_{W_d}$ be the result and moment matrix of the SDP (2.16). Then $\omega_d = \omega^f$ if and only if $\mathcal{M}_{W_d}$ has a rank loop and $rank(\mathcal{M}_{W_d}) \leq d$.*

We refer the readers to (Pironio et al., 2008) for the proof of Theorem 4.2.1. Algorithm 6 on page 31 is a pseudocode to detect rank loop. One must be careful when computing rank of the moment matrix using MATLAB built-in function *rank* because of round-off issues. For example, consider a matrix $A = \begin{pmatrix} 1.5 & 2 \\ 1.4999999999 & 2.000000001 - 0.000000001i \end{pmatrix}$. If we call $rank(A)$ in MATLAB, the result is 2 while we prefer the rank of $A$ to be 1. In practice, the moment matrix $\mathcal{M}_{W_d}$ is not 'numerically nice', just as the matrix $A$ above. Therefore, to determine the rank more accurately, we use $rank(A, \epsilon)$ which returns the number of singular values of $A$ larger than $\epsilon$ (See Section B.4 for the relation between rank of a matrix and its singular values). One should choose the value $\epsilon$ wisely so that the rank is determined accurately. If $\epsilon$ is too small, then $rank(A, \epsilon)$ likely tells that $A$ is full rank. If $\epsilon$ is too large, then clearly the resulted rank is not accurate either. After some experiments, we find that choosing $10^{-6} \leq \epsilon \leq 10^{-8}$ yields satisfactory accuracy.

### 4.2.2 Extraction of Optimal State and Projector Measurements

From the proof of Theorem 4.2.1, we also know how to extract the optimal state and projectors.

Let $\mathcal{M}_{W_d}$ be a moment matrix of the SDP at level $d$. Assume $\mathcal{M}_{W_d}$ has a rank loop. Because $\mathcal{M}_{W_d}$ is positive semidefinite, we can find its Cholesky decomposition, just as we do

**Algorithm 6:** HasRankLoop($\mathcal{M}_{W_d}, d, W_d, varProperties$)

**input** :

- $\mathcal{M}_{W_d}$
- d
- $W_d$
- *varProperties*: this variable is one of the attributes of the polynomial object described in Subsection 3.1.4

**output** :

- flagRankLoop: a binary variable which equals 1 if rank loop occurs, 0 otherwise.

**assumption**:

- There are only 2 parties.

```
1  varPartitionOne ← getAllVariablesInPartition(varProperties, 1)
2  varPartitionTwo ← getAllVariablesInPartition(varProperties, 2)
3  rankMoment ← computeRank(M_{W_d})
4  for i ← 1 to length(varPartitonOne) do
5      for j ← 1 to length(varPartitionTwo) do
6          E_a ← varPartitionOne(i)
7          E_b ← varPartitionTwo(j)
8          SM_{E_a,E_b} ← chooseListOfMonomialsForRankLoop(W_d, E_a, E_b, d)
9          M_{W_d}^{E_a,E_b} ← getSubMatrix(M_{W_d}, SM_{E_a,E_b})
10         if computeRank(M_{W_d}^{E_a,E_b}) ≠ rankMoment then
11             flagRankLoop ← 0
12             return flagRankLoop
13 flagRankLoop ← 1
14 return flagRankLoop
```

in Subsection 4.1.2. Hence, we have $\mathcal{M}_{W_d}(E_a, E_b) = \langle \xi_{E_a} | \xi_{E_b} \rangle$ for some finite set of vectors $\{|\xi_{E_a}\rangle : E_a \in W_d\}$. For each measurement $E_a \in S' \cup T'$ ($S'$, $T'$ are defined in Subsection 4.2.1), let $ST_{E_a}$ be the set of monomials in $W_d$ in which $E_a$ is the leading term in the variable ordering of the partition $E_a$ belongs to. We can get the value (or matrix representation) of the variable $E_a$ by setting

$$E_a = proj(span\{|\xi_E\rangle : E \in ST_{E_a}\}) \tag{4.9}$$

where $span(A)$ is the span of column vectors in the matrix $A$, and $proj(()V)$ is an orthogonal projection onto the vector space $V$. See Appendix B.5 for more details on how to find such a projection. It is shown in (Pironio et al., 2008) that the state $|\xi_{\mathbb{I}}\rangle$ and the set of matrices $\left\{E_a : E_a \in S' \cup T'\right\}$ found above are optimal state and projector measurements.

**Algorithm 7:** ExtractOptimalStateAndProjector($\mathcal{M}_{W_d}, W_d, listVar$)

**input** :

- $\mathcal{M}_{W_d}$
- $W_d$
- listVar: A list of variables (or singleton monomials)

**output** :

- optimalState: Optimal state
- listOptimalProjector: a list of matrix representations of optimal projectors.

**assumption**:

- There are only 2 parties, and $\mathcal{M}_{W_d}$ has rank loop.

```
choleskyDecomposition ← getCholeskyDecomposition(M_{W_d})
Initiate listOptimalProjector to be a list of size length(listVar)
for varIndex ← 1 to length(listVar) do
    E_a ← listVar(varIndex)
    ST_{E_a} ← filterMonomialsByLeadTerm(W_d, E_a)
    listOptimalProjector(i) ← FindProjection(choleskyDecomposition(:, ST_{E_a}))
optimalState ← choleskyDecomposition(:, 1)
return [optimalState  listOptimalProjector]
```

# Chapter 5

# Improving Performance and User-friendliness

## 5.1 Intermediate Levels

After preliminary experiments, we find out that the bottleneck of the program is the part in which SeDuMi solves the SDP. Therefore, improving the speed of transforming a quantum bound problem into an SDP has no significant importance. However, by reducing the size of the list of monomials, we can reduce the size of the SDP significantly. As a result, we can speed up the part of solving the SDP using SeDuMi.

Currently, we process each level $d$, then moving up to $d+1$. At the new level $d+1$, we have to generate all monomials of degree less than or equal to $2(d+1)$ which is costly because the number of monomials increases exponentially with the input degree. However, we can introduce the concept of 'intermediate' level. That is, after processing level $d$, we add some specific subset $S$ of $W_{d+1} \setminus W_d$ instead of the whole set itself. Then we can solve the SDP with the new list of monomials. The result may be a good approximation to $\omega_{d+1}$, and can be computed faster. In fact, we have $\omega_d \leq \omega_{S \cup W_d} \leq \omega_{d+1}$ [1] where $\omega_{S \cup W_d}$ is the result of the semidefinite program corresponding to the set of monomials $S \cup W_d$ (Pironio et al., 2008).

[1] This observation can be generalized that for the sets of monomials $S_1, S_2, S_3$ such that $S_1 \subset S_2 \subset S_3$, we have $\omega_{S_1} \leq \omega_{S_2} \leq \omega_{S_3}$.

In our package, we allow the users to specify how he wants the set of monomials of a cetain degree is generated. For instance, he can specify

- the number of partitions in which there must be at least one variable belonging to that partition that appears in the variable ordering of a monomial.
- whether he wants to take the full set of monomials generated, or randomly chooses a subset of the whole set generated; and in the latter case, how many monomials should be chosen randomly.

## 5.2 Input format

To make the software package user friendly, we must take into consideration the format users have to follow to enter their inputs, particularly the Bell inequalities they want to find their bounds. It would be a very bad package if users have to manually map their variables to integer values, then expand the polynomial expression they want to optimize to the sum of monomials, and then declare monomials and polynomials before calling a function to solve the quantum bound problem!

In our implementation, a user can enter his or her Bell inequality as a string of characters. Then the users provide information regarding the variables, such as their names, and types (projectors or observables). The package has a routine that takes all those parameters as inputs, does the mapping and parsing internally, and finally returns to the user the polynomial representation of his or her Bell inequality.

# Chapter 6

# Testing and Performance

We have tested the software package with the following Bell inequalities:

- $CHSH$ Inequality: $B_{CHSH} = A_1B_1 + A_1B_2 + A_2B_1 - A_2B_2$ where $A_1$, $A_2$ are observables of the first partition, and $B_1$, $B_2$ are observables of the second partition.
- $I_{3322}$ Inequality: $B_{I_{3322}} = A_1^a(B_1^b+B_2^b+B_3^b)+A_2^a(B_1^b+B_2^b-B_3^b)+A_3^a(B_1^b-B_2^b)-A_1^a-2B_1^b-B_2^b$ where $\{A_1^a\}$, $\{A_2^a\}$, $\{A_3^a\}$ are 3 incomplete measurement settings of the first party, and $\{B_1^b\}$, $\{B_2^b\}$, $\{B_3^b\}$ are 3 incomplete measurement settings of the second party.
- $Yao$ Inequality: $B_{Yao} = A_1B_2C_3 + A_2B_3C_1 + A_3B_1C_2 - A_1B_3C_2 - A_2B_1C_3 - A_3B_2C_1$ where $\{A_1, A_2, A_3\}$, $\{B_1, B_2, B_3\}$, $\{C_1, C_2, C_3\}$ are 3 partitions of observables.
- $Mod-2$ $Game$ Inequality: $B_{Mod-2} = \frac{1}{4}(A_0^1B_0^1 + A_0^0B_0^0 + A_0^0B_1^0 + A_0^1B_1^1 + A_1^0B_0^0 + A_1^0B_1^1 + A_1^1B_1^0 + A_1^1B_0^1)$ where $\{A_0^1, A_0^0\}$, $\{A_1^0, A_1^1\}$ are 2 complete measurement settings of the first party, $\{B_0^1, B_0^0\}$, $\{B_1^0, B_1^1\}$ are 2 complete measurement settings of the second party.
- $Mod-3$ $Game$ Inequality: $B_{Mod-3} = \frac{1}{9}(A_0^0(B_0^0+B_1^0+B_2^0)+A_0^1(B_0^2+B_1^2+B_2^2)+A_0^2(B_0^1+B_1^1+B_2^1)+A_1^0(B_0^0+B_1^1+B_2^2)+A_1^1(B_0^2+B_1^0+B_2^1)+A_1^2(B_0^1+B_1^2+B_2^0)+A_2^0*(B_0^0+B_1^2+B_2^1)+A_2^1(B_0^2+B_1^1+B_2^0)+A_2^2(B_0^1+B_1^0+B_2^2))$ (variables are projectors)
- $GHZ$ Inequality: $B_{GHZ} = \frac{1}{4}\sum A_s^aB_t^bC_u^c$ where variables are projectors and $(a+b+c) \bmod 2 = \begin{cases} 0 & \text{if } stu = 000 \\ 1 & \text{if } stu \in \{011, 101, 110\} \end{cases}$.

$GHZ$ Inequality is related to GHZ Paradox, another exciting thought experiment concerning with quantum entanglement (Greenberg, Horne, & Zeilinger, 1989). We have run many test cases with these inequalities in different levels. Results are shown in Table 6.1:

Table 6.1: Testing Bell Inequalities Table

| Inequality | Level | Upperbound | Does rank loop occurs? |
|---|---|---|---|
| $B_{CHSH}$ | 1 | $2.8284271247 \approx 2\sqrt{2}$ | N.A. |
| $B_{I_{3322}}$ | 1 | $0.3750000005 \approx \frac{3}{8}$ | N.A. |
| $B_{I_{3322}}$ | 3 | 0.2508755632 | No |
| $B_{I_{3322}}$ | 3 and choosing randomly half of monomials of order 4 in which variables in each partition appears at least once | 0.2508754027 | N.A. |
| $B_{Yao}$ | 2 | $5.1961524226 \approx 3\sqrt{3}$ | N.A. |
| $B_{Mod-2}$ | 2 | $0.8535533905 \approx \frac{1}{2} + \frac{1}{2\sqrt{2}}$ | Yes |
| $B_{Mod-3}$ | 2 and choosing randomly 50 monomials of order 3 in which variables in each partition appears at least once | 0.7123925377 | No |
| $B_{GHZ}$ | 2 | $0.9999999999 \approx 1$ | N.A. |

All the upper bounds of $B_{CHSH}$, $B_{I_{3322}}$, $B_{Yao}$ are the same as the results in (Wehner et al., 2008). The upper bound we obtain for $B_{Mod-3}$ is more precise than the one presented in (Liang, Lim, & Deng, 2009). In that paper, the result only has 4 digits after the decimal. The results obtained for $B_{Mod-2}$ and $B_{GHZ}$ also coincides with the known optimal upperbound.

Through experiments, the part of transforming the quantum bound problem into the SDP form that SeDuMi accepts is fast and accounts for only 10 percent of the running time. The bottleneck of the package is the part of solving the SDP itself. The part of detecting rank loop or extracting optimal measurements and states also has reasonable running time, at most a minute in all the test cases. As a rule of thumb, solving the SDP has a reasonable running time when the number of monomials is less than 120.

# Chapter 7

# Conclusion

## 7.1 Contributions

In this report, we provides an approach to implement an SDP-based algorithm to solve the quantum bound problem suggested in (Wehner et al., 2008), (Pironio et al., 2010) and (Pironio et al., 2008). The approach is illustrated by our MATLAB package. Though the discussion in this report is quite specific to the MATLAB implementation which uses SeDuMi and YALMIP as solvers of semidefinite programs, the high level ideas can be used to reimplement the algorithm in another language. Our implemented MATLAB package is distributed as open-source software. The scientific community can freely use it for finding bounds of Bell inequalities, or extracting optimal states and mesurements and examining their structures, or modifying the package to suit their needs.

## 7.2 Future Work

The bottleneck of the package is solving the SDP. Therefore, future work may involve implementing a parallel SDP algorithm, or reducing the size of the SDP by formulating our problem into the SDP form that SeDuMi accepts in a more clever way. A feature to detect rank loop for intermediate levels will be useful. We may implement a similar routine in C with the help of LAPACK to utilize the computation power of supercomputers to improve the running time.

# References


Friedberg, S. H., Insel, A. J., & Spence, L. E. (2002). *Linear algebra*. Pearson Education International, 4th edition.

Golub, G. H., & Van Loan, C. F. (1996). *Matrix computations*. The Johns Hopkins University Press, 3rd edition.

Greenberg, D. M., Horne, M. A., & Zeilinger, A. (1989). Going beyond bell's theorem. *Bell's Theorem, Quantum Theory, and Conceptions of the Universe*, , 1989.

Knuth, D. E. (2005). *The art of computer programming, volume 4, fascicle 2: Generating all tuples and permutations*. Addison-Wesley Professional, 1st edition.

Laurent, M. (2000). Sums of squares, moment matrices and optimization over polynomials. In M. Putinar, & S. Sullivant (Eds.), *Emerging applications of algebraic geometry*, Vol. 149 of *The IMA Volumes in Mathematics and its Applications* (pp. 157–270). Springer.

Liang, Y. C., Lim, C. W., & Deng, D. L. (2009). Reexamination of a multisetting bell inequality for qudits. *Physical Review A*, *80*, 2009.

Löfberg, J. (2004). Yalmip : A toolbox for modeling and optimization in MATLAB. *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

Nguyen Truong, D., McKague, M., & Wehner, S. (2013). Entanglement assistance in a classical control circuit. `http://www.locc.la/duy/controldraft.pdf`, , 2013. In preparation.

Nielsen, M. A., & Chuang, I. L. (2010). *Quantum computation and quantum information*. Cambridge University Press, 10th anniversary edition.

Papadimitriou, C. H., & Tsitsiklis, J. (1986). Intractable problems in control theory. *SIAM J. Control Optim.*, *24*(4), July, 1986, 639–654.

Parrilo, P. A. (2003). Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming B*, *96*, 2003, 293–320.

Pironio, S., Navascués, M., & Acín, A. (2008). A convergent hierachy of semidefinite programs characterizing the set of quantum correlations. *NEW J.PHYS.*, *10*, 2008, 073013.

Pironio, S., Navascués, M., & Acín, A. (2010). Convergent relaxations of polynomial optimization problems with non-commuting variables. *SIAM Journal on Optimization*, *20*(5), April, 2010.

Steeb, W. H., & Hardy, Y. (2011). *Matrix calculus and kronecker product: A practical approach to linear and multilinear algebra*. World Scientific Publishing Company, 2nd edition.

Sturm, J. Sedumi, a matlab toolbox for optimization over symmetric cones. `http://sedumi.ie.lehigh.edu`.

Vandenberghe, L., & Boyd, S. (1996). Semidefinite programming. *SIAM Review*, *38*(1), March, 1996, 49–95.

Wehner, S. (2008). *Cryptography in a quantum world*. PhD thesis, Centrum Wiskunde & Informatica, Amsterdam.

Wehner, S., Toner, B., Liang, Y. C., & Doherty, A. C. (2008). The quantum moment problem and bounds on entangled multi-prover games. *quant-ph/0803.4373*, , March, 2008.

# Appendix A

# Preliminaries of Quantum Computing

In this section, we briefly mention some basic concepts in quantum mechanics and computing which are used in our report. We refer the readers to some introductory textbooks of the fields, such as (Nielsen & Chuang, 2010), for more details.

## A.1 State Space and Quantum Bit

First, we state the postulate of quantum mechanics about the state space (Nielsen & Chuang, 2010).

**Postulate A.1.1.** *Any isolated physical system can be associated to a complex vector space equipped with an inner product operation (i.e. a Hilbert space). This vector space is called the state space of the system. The system is completely described by a unit vector $|\psi\rangle$ in the state space, which is called state vector.*

A quantum bit or a qubit is a unit of quantum information. It is the simplest quantum mechanical system. Its state space is $\mathbb{C}^2$. An orthornormal basis of this state space is $\left\{|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}, |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}\right\}$. Then any state vector $|\psi\rangle$ in the state space is equal to $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ for some $\alpha, \beta \in \mathbb{C}$ such that $\alpha^2 + \beta^2 = 1$. As you can see, unlike a classical bit which can only exist in either state 0 or 1, a qubit can be in a so-called superposition state, $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. This is one of the key properties that makes quantum computation much more efficient than classical computation.

## A.2 Composite Systems and Entanglement

We can use tensor product (See Section B.2) to put together two Hilbert spaces to form a larger one, which results in a larger quantum system. A qudit is a unit of information in a level $d$ quantum system. Its state space is $\mathbb{C}^d$ with an orthonormal basis $\{|i\rangle : i = 0, \ldots, d-1\}$ where $|i\rangle$ is a unit vector in $\mathbb{C}^d$ such that its $(i+1)$-th entry is 1.

The operation to use tensor product to obtain larger quantum systems, which are composite systems, leads to one of the most intriguing phenomemon in quantum mechanics, which is entanglement. We next define entanglement mathematically. Let $\mathcal{H}_A$ and $\mathcal{H}_B$ be Hilbert spaces.

A state vector $|\psi\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$ is entangled if there is no state vectors $|\psi_A\rangle \in \mathcal{H}_A$ and $|\psi_B\rangle \in \mathcal{H}_B$ such that $|\psi\rangle = |\psi_A\rangle |\psi_B\rangle$[1]. If $\psi$ is not entangled, we say it is separable.

**Example A.2.1.** *Consider a two qubit state* $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ *belonging to the state space* $\mathbb{C}^4 = \mathbb{C}^2 \otimes \mathbb{C}^2$. *It can be shown easily that there is no state* $|\zeta_1\rangle$, $|\zeta_2\rangle \in \mathbb{C}^2$ *such that* $\psi = |\zeta_1\rangle |\zeta_2\rangle$. *Hence,* $|\psi\rangle$ *is entangled.*

A state $|\psi\rangle$ in a level $d$ quantum system is maximally entangled if

$$|\psi\rangle = \frac{1}{\sqrt{d}} \sum_{i=0}^{d-1} |i\rangle |i\rangle \tag{A.1}$$

## A.3 Measurement

In this section, we define projectors (or projective measurements) and observables.

**Postulate A.3.1.** *Quantum projective measurements are a set of operators* $\{P_m\}$ *acting on the state space of the system being measured. The index* $m$ *is a measurement outcome that may occur in the measurement experiment. These operators* $P_m$ *must satisfy:*

- *They are Hermitian, i.e.* $P_m^\dagger = P_m$.
- $P_{m_1} P_{m_2} = 0$ *if* $m_1 \neq m_2$, *and* $P_m^2 = P_m$.
- $\sum_m P_m^\dagger P_m = \sum_m P_m = \mathbb{I}$

*If the state of the quantum system immediately before the measurement is* $|\psi\rangle$, *then outcome* $m$ *occurs with probability*

$$p(m) = \langle\psi| P_m |\psi\rangle \tag{A.2}$$

*and the state of the system after the measurement is:*

$$\left|\psi'\right\rangle = \frac{P_m |\psi\rangle}{\sqrt{p(m)}} \tag{A.3}$$

Let $M = \sum_m m P_m$. Then $M$ is an observable of the system being observed. Clearly, $M \succeq 0$ and $M^2 = \mathbb{I}$. Moreover, eigenvalues of $M$ are either 1 or -1.

## A.4 Other Special States and Operators

Pauli matrices (operators) are extremely useful in quantum mechanics. Their matrix representations are

$$\begin{aligned} \mathbb{I} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} & \sigma_x = X &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ \sigma_y = Y &= \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} & \sigma_z = Z &= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \end{aligned} \tag{A.4}$$

Note that the state space of qubit has another commonly used orthonormal basis which comprises two states

$$|+\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad |-\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{A.5}$$

[1] $|\psi_A\rangle |\psi_B\rangle = |\psi_A\rangle \otimes |\psi_B\rangle = |\psi_A \psi_B\rangle$

# Appendix B

# Linear Algebra

## B.1 Definitions of special matrices

In this section, we give definitions of special types of complex matrices commonly used in quantum computing. Throughout this entire report, we use Dirac standard notations of vectors and matrices[1].

A complex square matrix (or operator) $A$ is Hermitian or self-adjoint if $A^\dagger = A$. $A$ is normal if $AA^\dagger = A^\dagger A$. $A$ is unitary if $A^\dagger A = \mathbb{I}$.

An Hermitian operator $A \in \mathbb{C}^{n\times n}$ is positive semidefinite if for any vector $|\psi\rangle \in \mathbb{C}^n$, we have $\langle\psi| A |\psi\rangle$ is a real non-negative number. If for any nonzero vector $|\psi\rangle \in \mathbb{C}^n$, $\langle\psi| A |\psi\rangle$ is a real positive number, then we say $A$ is a positive definite matrix.

## B.2 Kronecker product

In linear algebra, tensor product is an operation of putting together vector spaces to form a larger one. Specifically, if $V$ and $W$ are vector spaces of dimension m and n respectively, then the tensor product of $V$ and $W$, denoted by $V \otimes W$, is an $m \times n$ dimensional vector space. Elements of $V \otimes W$ are linear combinations of tensor products $|v\rangle \otimes |w\rangle$ (or $|v\rangle |w\rangle$) where $|v\rangle \in V$ and $|w\rangle \in W$.

Kronecker product is a kind of tensor product applied when $V$ and $W$ are matrix spaces, and is used in quantum mechanics. For a matrix $A = (a_{ij})_{m\times n}$ and a matrix $B = (b_{ij})_{p\times q}$, the Kronecker product of $A$ and $B$ is defined as folows:

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix} \tag{B.1}$$

[1]Note that Dirac notations may differ from notations used in other linear algebra textbooks. For example, $A^*$ in Dirac notation denotes the complex conjugate of a matrix $A$ while $A^*$ denotes the conjugate transpose of $A$ in many textbooks. Another example is that $A^\dagger$ denotes the conjugate transpose of $A$, while $A^\dagger$ denotes the pseudo-inverse in other textbooks.

**Example B.2.1.** *Let* $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ *and* $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$. *Then*

$$A \otimes B = \begin{pmatrix} 1\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} & 2\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \\ 3\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} & 4\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{pmatrix} \tag{B.2}$$

The *nth* Kronecker power of a matrix $A$, denoted by $A^{\otimes(n)}$, is defined as:

$$\begin{aligned} A^{\otimes(0)} &= (1) \\ A^{\otimes(n)} &= A \otimes A^{\otimes(n-1)} \quad \text{for } n \geq 1 \end{aligned} \tag{B.3}$$

Below are some useful properties of Kronecker product that we use in the report. We refer the readers to (Steeb & Hardy, 2011) for a more thorough discussion about Kronecker product.

**Theorem B.2.2** (Basic Properties of Kronecker Product)**.** *Let A, B, C, D be matrices and k be a constant. Then*

- $A \otimes (B + C) = A \otimes B + A \otimes C$
- $(A + B) \otimes C = A \otimes C + B \otimes C$
- $(kA) \otimes B = A \otimes (kB) = k(A \otimes B)$
- $(A \otimes B)(C \otimes D) = AC \otimes BD$
- $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$
- $(A \otimes B)^T = A^T \otimes B^T$
- $(A \otimes B)^\dagger = A^\dagger \otimes B^\dagger$

The following are another useful result involving Kronecker product and maximally entangled state, and its proof.

**Theorem B.2.3.** *Let* $A, B \in \mathbb{C}^{d \times d}$ *and* $|\psi\rangle \in \mathbb{C}^d$ *be a maximally entangled state. Then*

$$\langle\psi| A \otimes B |\psi\rangle = \frac{1}{d} Tr(A^T B) \tag{B.4}$$

*Proof.* We have $|\psi\rangle = \sum_{i=0}^{d-1} |i\rangle \otimes |i\rangle$ where $|i\rangle$ is a unit vector whose $i+1$-th entry is 1. Then

$$\begin{aligned} \langle\psi| A \otimes B |\psi\rangle &= (\textstyle\sum_{i=0}^{d-1} \frac{1}{\sqrt{d}} \langle i| \otimes \langle i|) A \otimes B (\sum_{i=0}^{d-1} \frac{1}{\sqrt{d}} |i\rangle \otimes |i\rangle) \\ &= (\tfrac{1}{\sqrt{d}} \textstyle\sum_{i=0}^{d-1} \langle i| A \otimes \langle i| B)(\frac{1}{\sqrt{d}} \sum_{i=0}^{d-1} |i\rangle \otimes |i\rangle) \\ &= \tfrac{1}{d} \textstyle\sum_{i=0}^{d-1} \sum_{j=0}^{d-1} \langle i| A |j\rangle \otimes \langle i| B |j\rangle \\ &= \tfrac{1}{d} \textstyle\sum_{i=1}^{d} \sum_{j=1}^{d} A(i,j) \otimes B(i,j) = \frac{1}{d} \sum_{i=1}^{d} \sum_{j=1}^{d} A(i,j) B(i,j) \end{aligned} \tag{B.5}$$

(because $|i\rangle$ is a unit vector whose $i+1$-th entry is 1, $\langle i| A |j\rangle = A(i,j)$)

Also, $Tr(A^T B) = \sum_{i=1}^{d} (A^T B)(i,i) = \sum_{i=1}^{d} \sum_{j=1}^{d} A^T(i,j) B(j,i) = \sum_{i=1}^{d} \sum_{j=1}^{d} A(j,i) B(j,i) = \sum_{i=1}^{d} \sum_{j=1}^{d} A(i,j) B(i,j)$.

Therefore, $\langle\psi| A \otimes B |\psi\rangle = \frac{1}{d} Tr(A^T B)$. □

## B.3 Cholesky Decomposition of Positive Semidefinite Matrix

**Theorem B.3.1** (Existence of Cholesky Decomposition)**.** $A \in \mathbb{C}^{n \times n}$ *is positive semidefinite if and only if there exists a lower triangular matrix* $L \in \mathbb{C}^{n \times n}$ *with non-negative diagonal entries such that* $A = LL^\dagger$. *The decomposition of* $A = LL^\dagger$ *is called a Cholesky decomposition of the matrix A.*

If $A$ is positive definite, $A$ has a unique Cholesky decomposition. However, in case $A$ is positive semidefinite, a Cholesky decomposition is not unique.

Algorithm 8 on page B-3 is a pseudocode to find a Cholesky decomposition of a positive semidefinite matrix. The implementation performs satisfactorily in our testing. However, there is a more numerical stable version using the pivoting technique. We refer the readers to (Golub & Van Loan, 1996) for a more in-depth discussion of implementation versions of Cholesky decomposition.

**Algorithm 8:** GetCholeskyDecomposition($A$)

**input** :

- $A$: a complex positive-semidefinite square matrix

**output**:

- $U$: an upper triangular matrix such that $U^\dagger U = A$. Clearly $A = U^\dagger U$ is a Cholesky decomposition.

```
1  [numRow  numCol] ← size(A)
2  N ← numRow
   // Initiate U to be a zero matrix
3  U ← getZeroMatrixOfSize([N  N])
4  for i ← 1 to N do
5      U(i, i) ← A(i, i) − getConjugateTranspose(U(1 : i − 1, i)) ∗ U(1 : i − 1, i)
6      if isZero(|U(i, i)|) ≠ TRUE then
7          U(i, i) ← √U(i, i)
8          for j ← i + 1 to N do
9              U(i, j) ← 1/U(i, i) ∗ (A(i, j) − U(1 :
               i − 1, j) ∗ getConjugateTranspose(U(1 : i - 1, i))
10 return U
```

## B.4 Singular Value Decomposition

In this section, we define singular values of a matrix and state the theorem of singular value decomposition without proof. Then we introduce some applications of singular value decomposition in computing rank and finding orthonormal bases of a matrix.

Let $A \in \mathbb{C}^{m\times n}$ (the sets of $m \times n$ complex matrices). It follows from definition that $A^\dagger A$ is a square positive semidefinite matrix. Therefore, eigenvalues $\lambda$ of $A^\dagger A$ are real and non-negative. Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_r$ be positive eigenvalues of $A^\dagger A$. Set $k = \min(m, n)$. Let $\lambda_{r+1} = \ldots = \lambda_k = 0$.

Then singular values of $A$ are positive real numbers $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k$ where $\sigma_i = \sqrt{\lambda_i}$.

The theorem of Singular Value Decomposition is stated as follows:

**Theorem B.4.1** (Existence of Singular Value Decomposition of Matrices)**.** *Let $A$ be an $m \times n$ matrix with $k = \min(m, n)$ non-negative singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k$. Let $\Sigma$ be a $m \times n$ diagonal rectangular matrix such that $\Sigma(i, i) = \sigma_i$ for $i = 1, \ldots, k$. Then there exist unitary matrices $U$ and $V$ of order $m$ and $n$ respectively such that*

$$A = U\Sigma V^\dagger \tag{B.6}$$

A factorization of a matrix $A = U\Sigma V^\dagger$ in Theorem B.4.1 is called a singular value decomposition of $A$.

We refer the readers to (Friedberg, Insel, & Spence, 2002) for the proof of Theorem B.4.1. Following from the proof, we have a useful corollary:

**Corollary B.4.2.** *Let $A = U\Sigma V^\dagger$ be a singular value decomposition of the $m \times n$ matrix $A$. Let $S = \{i \,|\, \Sigma(i, i) > 0, i = 1, \ldots, \min(m, n)\}$. The set of $i$-th column vectors of $U$ where $i \in S$ forms an orthonormal basis of the column space of $A$.*

Another useful corollary is:

**Corollary B.4.3.** *Let $A$ be an $m \times n$ matrix. Then the rank of $A$ equals the number of its nonzero singular values.*

**Example B.4.4.** *Consider $A = \begin{pmatrix} 1 & 1 & -1 \\ 1 & 1 & -1 \end{pmatrix}$. Its singular value decomposition is $A = U\Sigma V^\dagger$ where $U = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$, $\Sigma = \begin{pmatrix} \sqrt{6} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, and $V = \begin{pmatrix} \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \\ \frac{-1}{\sqrt{3}} & 0 & \frac{2}{\sqrt{6}} \end{pmatrix}$. Because $A$ has only $\sqrt{6}$ as its nonzero singular value, $\texttt{computeRank}(A) = 1$. Also, because $\Sigma(1, 1) = \sqrt{6}$, the singleton set $\left\{ \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \right\}$ (the first column of $U$) forms an orthonormal basis of the column space of $A$.*

Corollary (B.4.2) provides a more numerically stable algorithm than the Gram-Schmidt process to find an orthonormal basis of the column space of a matrix. The pseudocode is presented in Algorithm 9 on page B-4.

**Algorithm 9:** GetOrthonormalBasis($A$)

**input** :

- $A$: a complex matrix

**output**:

- $O$: a matrix whose columns form an orthnormal basis of the column space of $A$.

```
1 [numRow  numCol] ← size(A)
  // Get the Singular Value Decomposition of A. In MATLAB, svd is a function
     to find such a decomposition
2 [U  Σ  V] ← svd(A)
  // Set O to be an empty matrix
3 O ← EMPTY
4 for i ← 1 to min(numRow, numCol) do
5     if isZero(Σ(i, i)) ≠ TRUE then
          // Add the corresponding column in U to M
          // We denote U(:, i) to be the i-th column of U
6         addColumn(O, U(:, i))
7 return O
```

Similarly, Corollary (B.4.3) provides a method which is more reliable and simpler to implement (provided we already have a built-in function to find singular value decompositions) to compute a rank of a matrix.

## B.5 Projection

In this section, we show how to find a projector onto a subspace spanned by a set of vector.

Let $W \subset V$ be a subspace of a vector space $V$. Let $\{|\psi_1\rangle, \ldots, |\psi_m\rangle\}$ be the spanning set of $W$. We know that we can find an orthonormal basis $\{|\beta_1\rangle, \ldots, |\beta_k\rangle\}\, of W$. Then

$$P = \sum_{i=1}^{k} |\beta_i\rangle \langle\beta_i| \tag{B.7}$$

is a projector onto the subspace $W$.

Algorithm 10 on page B-5 is a pseudocode to compute a projector onto the column space of a matrix A. Note that this code is a helper routine used in extracting optimal projector measurements presented in Subsection 4.2.2.

**Algorithm 10:** FindProjection($A$)

**input** :

- $A$: a complex matrix

**output**:

- $P$: a projector onto the column space of $A$

```
// OB is a matrix whose columns form an orthonormal basis of the column
   space of A
// GetOrthonormalBasis is a procedure defined in Algorithm 9
OB ← GetOrthonormalBasis(A)
[numRow  numCol] ← size(OB)
// Set P to be a zero matrix
P ← getZeroMatrixOfSize([numRow  numCol])
for i ← 1 to numCol do
    // We denote OB(:, i) to be the i-th column of OB
    P ← P + getConjugateTranspose(OB(:, i)) * OB(:, i)
return P
```

# Appendix C

# Semidefinite Programming and Finding Bounds on Polynomials in Commutative Variables

In this appendix, we briefly introduce semidenite programming, a useful tool to tackle the quantum bound problem. Then we discuss the problem of finding bounds on a polynomial in commutative variables first to have a clear picture of the concepts and techniques used. All these topics are discussed more in-depth in (Vandenberghe & Boyd, 1996), (Parrilo, 2003), and (Laurent, 2000).

## C.1 Semidefinite programming

Semidefinite programming is a kind of convex programming with many applications. It can be seen as a generalization of linear programming. Here we present two equivalent definitions of a semidefinite program that are mostly used and their corresponding dual problems.

The first definition is in (Vandenberghe & Boyd, 1996), a thorough and excellent survey of semidefinite programming. Let x $\in \mathbb{R}^m$ be a variable. Let c $\in \mathbb{R}^m$ be a fixed vector. Let $F_0, F_1, ..., F_m$ be m + 1 symmetric matrices in $\mathbb{R}^{n\times n}$. Then a semidefinite program is an optimization problem:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & F(x) = F_0 + \sum_{i=1}^{m} x_i F_i \succeq 0 \end{array} \tag{C.1}$$

where $A \succeq 0$ means that the square matrix A of order n is positive definite, i.e. $y^T A y \geq 0$ $\forall y \in \mathbb{R}^n$. The dual problem of the semidefinite program (C.1) is

$$\begin{array}{ll} \text{maximize} & -tr(F_0 Z) \\ \text{subject to} & tr\,(F_i Z) = c_i \ \forall i = 1, ..., m, \\ & Z = Z^T \succeq 0 \text{ and } Z \in \mathbb{R}^{n\times n} \end{array} \tag{C.2}$$

Another commonly used definition of a semidefinite program can be found in (Parrilo, 2003). Its primal problem is more or less the dual problem (C.2) of the semidefinite program (C.1). That is,

$$\begin{array}{ll} \text{minimize} & tr(F_0 Z) \\ \text{subject to} & tr\,(F_i Z) = c_i \ \forall i = 1, ..., m, \\ & Z = Z^T \succeq 0 \text{ and } Z \in \mathbb{R}^{n\times n} \end{array} \tag{C.3}$$

The corresponding dual problem is

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & F(x) = F_0 - \sum_{i=1}^{m} x_i F_i \succeq 0 \end{array} \tag{C.4}$$

There are many efficient polynomial-time algorithms to solve a semidefinite program, such as primal-dual potential reduction methods (see (Vandenberghe & Boyd, 1996)). SeDuMi (Sturm, ) is a popular solver for semidefinite programming.

## C.2 Finding bounds on a polynomial of commutative variables

In this section, we discuss an application of semidefinite programming in computing bounds on a polynomial of commutative variables, which is quite related to our non-commutative case.

### C.2.1 Global nonnegativity of multivariate polynomials

#### Polynomials of commutative variables

A polynomial $f$ in $n$ commutative variables $x_1$, $x_2$, ..., $x_n$ is a finite linear combination of monomials. It has a form $f = \sum_\alpha c_\alpha v^\alpha = \sum_\alpha x_\alpha^{\alpha_1} \dots x_n^{\alpha_n}$ where $c_\alpha \in \mathbb{R}$ and $x_\alpha^{\alpha_1} + \dots + x_n^{\alpha_n} = \alpha$. A commutative ring of all multivariate polynomials in $n$ commutative variables is denoted as $\mathbb{R}[x_1, x_2, \dots, x_n]$. The total degree of a monomial $v^\alpha$ is equal to $x_\alpha^{\alpha_1} + \dots + x_n^{\alpha_n}$. The total degree of a polynomial $f$ is the maximum total degree of all its monomials. A homogenous polynomial (or a form) is a polymial where all its monomials have the same total degree. A polynomial $F$ is called to have a sum of squares decomposition if $F$ can be expressed as:

$$F = \sum_i f_i^2 \tag{C.5}$$

where $f_i \in \mathbb{R}[x_1, x_2, \dots, x_n]$

#### Global nonnegativity and Sum of squares relaxation

Given a polynomial $F \in \mathbb{R}[x_1, x_2, \dots, x_n]$, the global nonnegativity problem is to check if

$$F(x_1, x_2, \dots, x_n) \geq 0 \ \forall x_1, x_2, \dots, x_n \in \mathbb{R} \tag{C.6}$$

Note that to satisfy ($C$.6), it is necessary that the degree of $F$ is even. Clearly, if a polynomial $F$ has a sum of squares decomposition, then $F$ satisfies the condition ($C$.6). However, not all nonnegative polynomial has a sum of squares decomposition. In (Parrilo, 2003), there is an example of such a polynomial, which is $M(x, y, z) = x^4y^2 + x^2y^4 + z^6 - 3x^2y^2z^2$. It is known that the global nonnegativity problem of a polymial is NP-hard (Parrilo, 2003). Therefore, we want to relax the condition (C.6) so that it can be solved more efficiently. A possible relaxation is to check if a polynomial $F$ has a sum of square decomposition.

#### Sum of Squares and Semidefinite Programming

Consider a polynomial $F \in \mathbb{R}[x_1, x_2, \dots, x_n]$ of degree $2d$. Let $z$ be a row vector of all the monomials of $F$ that have the total degree less than or equal to $d$, i.e. $z = \left[1, x_1, x_2, \dots, x_n, x_1x_2, \dots, x_n^d\right]$. Then the polynomial $F$ can be expressed in a form $F(x_1, x_2, \dots, x_n) = z^T Q z$ where $Q$ is some constant matrix. It can be shown that $F$ has a sum of squares decomposition if and only if $Q$ is positive semidefinite, i.e. $Q \succeq 0$. Therefore, the problem of checking if $F$ has a sum of squares decomposition can be cast as a semidefinite program of the form (C.3) where the expression that needs minimizing is a constant. Hence, it can be determined in polynomial time in the size of the matrix $Q$ if a polynomial $F$ has a sum of squares decomposition.

**Polynomial Optimization and Hierachy of Semidefinite Programs**

Now we consider the problem of polynomial optimization in commutative variables. Let $p, g_1, g_2, \ldots, g_m \in \mathbb{R}[x]$ be such that

$$K = \{x \in \mathbb{R}^n | g_1(x) \geq 0, \ldots, g_m(x) \geq 0\} \tag{C.7}$$

be a semialgebraic set. The problem of polynomial optimization can be formulated as follows:

$$\begin{array}{ll} \text{maximize} & \gamma \\ \text{subject to} & p(x) - \gamma \geq 0 \ \forall x \in K \end{array} \tag{C.8}$$

In other words, the problem (C.8) finds the infimum (greatest lower bound) of the polynomial $p$ over the semialgebraic set $K$. The problem of finding an upperbound of $p$ can be formulated in a similar manner. It can be shown that the problem (C.8) is hard. Therefore, a natural approach is to relax the problem as we do to the problem of checking global nonnegativity (C.6). In (Laurent, 2000), a detailed survey of the topic, there are many efficient relaxations which depend on additional properties of $K$. Here we present a relaxtion based on Putinar's Positivestellensatz.

The quadratic module generated by polynomials $g_1, \ldots, g_m$ is defined as $M(g_1, \ldots, g_m) = \left\{u_0 + \sum_{j=1}^m u_j g_j | u_0, u_j \in \Sigma\right\}$ where $\Sigma \subset \mathbb{R}[x]$ is the set of all polynomials that have a sum of squares decomposition. A quadratic module $M(g_1, \ldots, g_m)$ is Archimedean if it satisfies $\forall p \in \mathbb{R}[x], \exists K \in \mathbb{N}, N \pm p \in M(g_1, \ldots, g_m)$. Now we can state Putinar's Positivstellensatz.

**Theorem C.2.1.** *Let $K$ be a semialgebraic set as in (C.7) and suppose the quadratic module $M(g_1, \ldots, g_m)$ is Archimedean. For a polynomial $F \in \mathbb{R}[x]$, if $F > 0$ on $K$, then $F \in M(g_1, \ldots, g_m)$.*

Note that in the problem (C.8),

$$\gamma = \sup\{\rho | p(x) - \rho \geq 0 \ \forall x \in K\} = \sup\{\rho | p(x) - \rho > 0 \ \forall x \in K\} \tag{C.9}$$

Therefore, problem (C.8) can be relaxed as

$$\begin{array}{ll} \text{maximize} & \gamma_t^{SOS} \\ \text{subject to} & p(x) - \gamma_t^{SOS} = s_0 + \sum_{j=1}^m s_j g_j \\ & s_0, s_j \in \Sigma \\ & \deg(s_0), \deg(s_j g_j) \leq 2t \end{array} \tag{C.10}$$

where $\deg(f)$ is the degree of a polynomial $f$.

Equivalently, the problem (C.10) can be reformulated as

$$\begin{array}{ll} \text{maximize} & \gamma_t^{SOS} \\ \text{subject to} & p(x) - \gamma_t^{SOS} - \sum_{j=1}^m s_j g_j \in \Sigma \\ & s_j \in \Sigma \\ & \deg(s_j g_j) \leq 2t \end{array} \tag{C.11}$$

Note that in the formulation of the problems (C.10) and (C.11), we can construct a hierachy of semidefinite programs in $t$. Clearly, $\gamma_t^{SOS} \leq \gamma$ ($\gamma$ is the value we try to maximize in the problem (C.8)). According to (Laurent, 2000), under certain conditions on the semialgebraic set $K$, there is a finite convergence of $\left\{\gamma_t^{SOS}\right\}$ to $\gamma$.