Undergraduate Research Opportunity Program
(UROP) Progress 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) Progress 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

Abstract

In this literature review, we introduce the problem of computing bounds on quantum probabilies and tools that researchers use to tackle the problem. We also mention briefly what we plan to do and achieve in this project.

Keywords:
semidefinite proramming, sum of squares, weighted sum of squares, polynomial optimization, 2-prover cooperative game, non-commutative variables, positivstellensatz.

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.

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 π 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 π. 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, π).
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 project 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 project 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, π) 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 give 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

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

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

$$\omega^*\left(G\right) = \lim_{d\to\infty} \max_{\substack{|\psi\rangle\in\mathbb{C}\otimes\mathbb{C} \\ |||\psi\rangle||=1}} \max_{A_s^a, B_t^b} \sum_{a,b,s,t} \pi\left(s,t\right) V\left(a,b \mid s,t\right) \langle\psi| A_s^a \otimes B_t^b |\psi\rangle \tag{1.1}$$

where $A_s^a \in \mathbb{B}\left(\mathcal{H}_A\right)$ ($\mathbb{B}\left(\mathcal{H}_A\right)$ is the set of all bounded operators on the Hilbert space $\mathcal{H}_A$) and $B_t^b \in \mathbb{B}\left(\mathcal{H}_B\right)$ 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 $\sum_a A_s^a = \mathbb{I}_A$, $\sum_b B_t^b = \mathbb{I}_B$, $\left(A_s^a\right)^2 = A_s^a$, 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 $\omega^*\left(G\right)$, which is called the entangled value of G = G(V, π). In Chapter 3, we show that we can reduce the problem of finding bounds on $\omega^*\left(A\right)$ to finding bounds of a polynomial in non-commutative variables which are projective measurements or observables. We refer the reader to the book by Nielsen and Chuang (Nielsen & Chuang, 2010) for an excellent and detailed introduction to concepts of quantum computation and information which include quantum measurement, quantum state, projective measurement and observable.

Chapter 2

Semidefinite Programming and Finding Bounds on Polynomials in Commutative Variables

Before discussing our main problem of finding bounds on a polynomial in non-commutative variables, we go through semidenite programming, a useful tool to tackle the 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.

2.1 Semidefinite programming

Semidefinite programming is a kind of nonconvex 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 the thorough and excellent paper by Boyd and Vandenberghe (Vandenberghe & Boyd, 1996). 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:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & F(x) = F_0 + \sum_{i=1}^m x_i F_i \succeq 0 \end{array} \tag{2.1}$$

where $A \succeq 0$ means that the square matrix A of order n is positive definite, i.e. $y^T Ay \geq 0$ $\forall y \in \mathbb{R}^n$
The dual problem of the semidefinite program (2.1) is

$$\begin{array}{ll} \text{maximize} & -tr(F_0 Z) \\ \text{subject to} & tr\,(F_i Z) = c_i \; \forall i = 1, ..., m, \\ & Z = Z^T \succeq 0 \text{ and } Z \in \mathbb{R}^{n\times n} \end{array} \tag{2.2}$$

Another commonly used definition of a semidefinite program can be found in the paper of Parrilo (Parrilo, 2003). Its primal problem is more or less the dual problem (2.2) of the semidefinite program (2.1). That is,

$$\begin{array}{ll} \text{minimize} & tr(F_0 Z) \\ \text{subject to} & tr\,(F_i Z) = c_i \; \forall i = 1, ..., m, \\ & Z = Z^T \succeq 0 \text{ and } Z \in \mathbb{R}^{n\times n} \end{array} \tag{2.3}$$

The corresponding dual problem is

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & F(x) = F_0 - \sum_{i=1}^m x_i F_i \succeq 0 \end{array} \tag{2.4}$$

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.

2.2 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.

2.2.1 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:

$$f = \sum_\alpha c_\alpha v^\alpha = \sum_\alpha x_\alpha^{\alpha_1} \dots x_n^{\alpha_n} \tag{2.5}$$

where $c_\alpha \in \mathbb{R}$ and $x_\alpha^{\alpha_1} + \ldots + x_n^{\alpha_n} = \alpha$

A commutative ring of all multivariate polynomials in n commutative variables is denoted as $\mathbb{R}[x_1, x_2, \ldots, x_n]$.
The total degree of a monomial v^α is equal to $x_\alpha^{\alpha_1} + \ldots + x_n^{\alpha_n}$. 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:

$$F = \sum_i f_i^2 \tag{2.6}$$

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

Global nonnegativity and Sum of squares relaxation

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

$$F(x_1, x_2, \ldots, x_n) \geq 0 \ \forall x_1, x_2, \ldots, x_n \in \mathbb{R} \tag{2.7}$$

Note that to satisfy (2.7), 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 (2.7). However, not all nonnegative polynomial has a sum of squares decomposition. In a paper by Parrilo (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. Therefore, we want to relax the condition (2.7) 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 $F \in \mathbb{R}[x_1, x_2, \ldots, x_n]$ 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.

$$z = \left[1, x_1, x_2, \ldots, x_n, x_1x_2, \ldots, x_n^d\right]$$

Then the polynomial F can be expressed in a form:

$$F(x_1, x_2, \ldots, x_n) = z^T Q z \tag{2.8}$$

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 (2.3) 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 $p, g_1, g_2, \ldots, g_m \in \mathbb{R}[x]$.

$$K = \{x \in \mathbb{R}^n | g_1(x) \geq 0, \ldots, g_m(x) \geq 0\} \tag{2.9}$$

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

$$\begin{array}{ll} \text{maximize} & \gamma \\ \text{subject to} & p(x) - \gamma \geq 0 \ \forall x \in K \end{array} \tag{2.10}$$

In other words, the problem (2.10) finds the infimum (greatest lower bound) of the polynomial p over the semialgebraic set K. The problem of finding an upper bound of p can be formulated in a similar manner.

It can be shown that the problem (2.10) is hard. Therefore, a natural approach is to relax the problem as we do to the problem of checking global nonnegativity (2.7). In a detailed survey of the topic by Laurent (Laurent, 2000), 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:

$$M(g_1, \ldots, g_m) = \left\{ u_0 + \sum_{j=1}^{m} u_j g_j | u_0, u_j \in \Sigma \right\} \tag{2.11}$$

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 the following condition

$$\forall p \in \mathbb{R}[x], \exists K \in \mathbb{N}, N \pm p \in M(g_1, \ldots, g_m) \tag{2.12}$$

Now we can state Putinar's Positivstellensatz.
Theorem

Theorem 2.2.1. *Let K be a semialgebraic set as in (2.9) 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 (2.10),

$$\begin{aligned} \gamma &= \sup\{\rho | p(x) - \rho \geq 0 \ \forall x \in K\} \qquad (2.13) \\ &= \sup\{\rho | p(x) - \rho > 0 \ \forall x \in K\} \qquad (2.14) \end{aligned}$$

Therefore, problem (2.10) can be relaxed as

$$\begin{array}{ll} \text{maximize} & \gamma_t^{SOS} \\ \text{subject to} & p(x) - \gamma_t^{SOS} = s_0 + \sum_{j=1}^m s_j g_j \\ & s_0, s_j \in \Sigma \\ & \deg(s_0), \deg(s_j g_j) \leq 2t \end{array} \qquad (2.15)$$

where deg(f) is the degree of a polynomial f.
Equivalently, the problem (2.15) can be reformulated as

$$\begin{array}{ll} \text{maximize} & \gamma_t^{SOS} \\ \text{subject to} & p(x) - \gamma_t^{SOS} - \sum_{j=1}^m s_j g_j \in \Sigma \\ & s_j \in \Sigma \\ & \deg(s_j g_j) \leq 2t \end{array} \qquad (2.16)$$

Note that in the formulation of the problems (2.15) and (2.16), in fact we construct a hierachy of semidefinite programs in t. Clearly, $\gamma_t^{SOS} \leq \gamma$ (γ is the value we try to maximize in the problem (2.10)). According to the survey by Laurent, under certain conditions on the semialgebraic set K, there is a finite convergence of $\{\gamma_t^{SOS}\}$ to γ.

Chapter 3

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 (1.1), can be relaxed to the problem of polynomial optimization in non-commutative variables.

3.1 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

$$\omega^f (G) = \sup_{A_s^{'a}, B_t^{'b}} \left\| \sum_{a,b,s,t} \pi (s,t) V(a,b|s,t) A_s^{'a} B_t^{'b} \right\| \tag{3.1}$$

where $\|X\| = \max \left\{ \lambda | \lambda \text{ is an eignvalue of } X^\dagger X \right\}$ ($X^\dagger$ is the Hermitian conjugate of the operator X), $A_s^{'a} \in \mathbb{B}(\mathcal{H})$ 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 3.1.1. *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 a property of tensor product $A_s^{'a} B_t^{'b} = \left[A_s^a \otimes \mathbb{I}_B\right] \left[\mathbb{I}_A \otimes B_t^b\right] = \left(A_s^a \mathbb{I}_A\right) \otimes \left(\mathbb{I}_B B_t^b\right) = A_s^a \otimes B_t^b = B_t^{'b} A_s^{'a}$. The proof of the other direction of Theorem 3.1.1 and a general version of Theorem 3.1.1 are presented in the paper by Wehner et al. (Wehner, Toner, Liang, & Doherty, 2008). Also, in that paper, it is proved that

$$\omega^*(G) \le \omega^f(G) \tag{3.2}$$

3.2 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.

By Equation (3.2), if we find an upperbound ϑ of $\omega^f(G)$, then ϑ is an upperbound of $\omega^*(G)$. 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 (3.1) are orthogonal projectors. Note that an operator X is orthogonal projectors if and only if $X^\dagger = X$ and $X^2 = X$. Hence, we have:

$$\left\| \sum_{a,b,s,t} \pi(s,t)\, V(a,b|s,t)\, A_s^{'a} B_t^{'b} \right\| = \max\left\{ \lambda | \lambda \text{ is an eigenvalue of } \sum_{a,b,s,t} \pi(s,t)\, V(a,b|s,t)\, A_s^{'a} B_t^{'b} \right\} \tag{3.3}$$

Let ν be a real number. Consider the expression:

$$q_\nu = \nu\mathbb{I} - \sum_{a,b,s,t} \pi(s,t)\, V(a,b|s,t)\, A_s^{'a} B_t^{'b} \tag{3.4}$$

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

$$\nu \ge \max\left\{ \lambda | \lambda \text{ is an eigenvalue of } \sum_{a,b,s,t} \pi(s,t)\, V(a,b|s,t)\, A_s^{'a} B_t^{'b} \right\} \tag{3.5}$$

Hence, $\nu \ge \omega^f(G) \ge \omega^*(G)$. Thus, ν 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:

$$\begin{array}{ll} \text{minimize} & \nu \\ \text{subject to} & q_\nu = \nu\mathbb{I} - \sum_{a,b,s,t} \pi(s,t)\, V(a,b|s,t)\, A_s^{'a} B_t^{'b} \succeq 0 \\ & \sum_{a\in A} A_s^{'a} - \mathbb{I} = 0 \text{ and } \sum_{b\in B} B_t^{'b} - \mathbb{I} = 0 \\ & \left(A_s^{'a}\right)^2 = A_s^{'a} \succeq 0 \text{ and } \left(B_t^{'b}\right)^2 = B_t^{'b} \succeq 0 \;\forall a \in A, b \in B, s \in S, t \in T \\ & \left[A_s^{'a}, B_t^{'b}\right] = 0 \;\forall a \in A, b \in B, s \in S, t \in T \\ & \left(A_s^{'a}\right)^\dagger = A_s^{'a} \text{ and } \left(B_t^{'b}\right)^\dagger = B_t^{'b} \;\forall a \in A, b \in B, s \in S, t \in T \end{array} \tag{3.6}$$

Clearly, the optimization problem (3.6) is a polynomial optimization in non-commutable and Hermitian variables $A_s^{'a}, B_t^{'b}$ (An operator X is Hermitian if $X^\dagger = X$).

3.3 Helton and McCullough's Positivstellensatz and Semidefinite Programming Hierachy

In this section, we introduce the Helton and McCullough's Positivstellensatz, which is used to relax Problem (3.6). We also discuss a hierachy of semidefinite program which is used to solve a relaxation of Problem (3.6).

Let $\mathbb{C}\left[x, x^{\dagger}\right]$ be the set of polynomials in the 2n noncommutative variables $x = (x_1, \ldots, x_n)$ and $x^{\dagger} = \left(x_1^{\dagger}, \ldots, x_n^{\dagger}\right)$ with coefficients in $\mathbb{C}$. Let $\mathcal{P} \subset \mathbb{C}\left[x, x^{\dagger}\right]$ be a collection of Hermitian polynomials. The convex cone $\mathcal{C}_{\mathcal{P}}$ generated by $\mathcal{P}$ consists of polynomials of the form

$$q = \sum_{i=1}^{M} r_i^{\dagger} r_i + \sum_{j=1}^{N} \sum_{k=1}^{L} s_{jk}^{\dagger} p_j s_{jk} \tag{3.7}$$

where $p_i \in \mathcal{P}$, M, N, L are finite, and $r_i, s_{jk} \in \mathbb{C}\left[x, x^{\dagger}\right]$.
Equation (3.7) is called a weighted sum of squares (WSOS) representation of a polynomial q.

Consider G = G(π, V). Denote $\mathcal{A}' = \left\{A_s'^a \mid a \in A, s \in S\right\}$ and $\mathcal{B}' = \left\{B_t'^b \mid b \in B, t \in T\right\}$ where $A_s'^a \in \mathbb{B}(\mathcal{H})$ 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). Let us choose

$$\mathcal{P} = \left\{i\left[A_s'^a, B_t'^b\right]\right\} \cup \left\{\sum_{a\in A} A_s'^a - \mathbb{I}\right\} \cup \left\{\sum_{b\in B} B_t'^b - \mathbb{I}\right\} \cup \left\{\left(A_s'^a\right)^2 - A_s'^a\right\} \cup \left\{\left(B_t'^b\right)^2 - B_t'^b\right\} \tag{3.8}$$

Now we will state the Helton and McCullough's Positivstellensatz in the context of a 2-prover game.

Theorem 3.3.1. *Let $G = G(\pi, V)$ be a 2-prover cooperative game, and let $\mathcal{C}_{\mathcal{P}}$ be the cone generated by the set $\mathcal{P}$ defined in (3.8). Set*

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

If $q_\nu \succ 0$, then $q_\nu \in \mathcal{C}_{\mathcal{P}}$.

We refer the reader to the paper by Wehner et al. (Wehner et al., 2008) for a more general version of Theorem 3.3.1 and its proof.
By Theorem 3.3.1, we have a relaxation version of Problem (3.6) as follows:

$$\begin{array}{ll} \text{minimize} & \nu \\ \text{subject to} & \nu\mathbb{I} - \sum_{a,b,s,t} \pi(s,t) V(a,b|s,t) A_s'^a B_t'^b \succeq 0 = \sum_{i=1}^{M} r_i^{\dagger} r_i + \sum_{j=1}^{N} \sum_{k=1}^{L} s_{jk}^{\dagger} p_j s_{jk} \\ & \max\left(\deg(r_i), \deg\left(s_{jk}^{\dagger} p_j s_{jk}\right)\right) \le 2t \end{array} \tag{3.9}$$

where $p_j \in \mathcal{P}$, and r_i, s_{jk} are some arbitrary polynomial in variables $A_s'^a, B_t'^b$.
Note that for each fixed value of t, Problem (3.9) is a semidefinite program. Hence, we can form a hierachy of semidefinite programs based on the value of t. Let $\omega_t^{sdp}(G)$ denote the solution to the semidefinite program (3.9) corresponding to t. Then Wehner et al. proves that $\lim_{n\to\infty} \omega_t^{sdp}(G) = \omega^f(G)$.
Another relaxtion of Problem (3.6) is discussed in the paper by Pironio et al. (Pironio, Navascués, & Acín, 2010) which involves moment matrices and localizing matrices. We discuss this relaxtion in Chapter **??**. This relaxation version turns out to be the primal program of (3.9).

Chapter 4

Formulation Involving Moment Matrices

In this chapter, we generalize the problem of quantum probabilities problem to more than 2 parties. Also, we discuss the primal program of (3.9) which involes moment matrices. Our implementation in MATLAB is based on this formulation.

4.1 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 $x = (x_1, \ldots, x_n)$ 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:

$$\begin{array}{rccc} \mathcal{L}\colon & S & \longmapsto & \mathbb{C} \\ & w & \mapsto & y_w \end{array} \tag{4.1}$$

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

$$\mathcal{M}\,(v, w) = \mathcal{L}\left(v^\dagger w\right) = y_{v^\dagger w} \tag{4.2}$$

4.2 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 (A complex matrix X is Hermitian if $X^\dagger = X$). There are two cases in which all operators are projectors or observables. Suppose we partition $\mathcal{S}$ into k disjoint subsets

$$\mathcal{S} = \bigcup_{i=1}^{k} C_i \tag{4.3}$$

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:

$$\begin{aligned} \mathcal{P}\colon\ \mathcal{S} &\longmapsto \{1,2,\ldots,k\} \\ E &\mapsto i \end{aligned} \tag{4.4}$$

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

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

$$\mathcal{S} = \bigcup_{i=1}^{m} S_i \tag{4.5}$$

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 an input group with index i. In this case, we define another function $\mathcal{I}$ as follows:

$$\begin{aligned} \mathcal{I}\colon\ \mathcal{S} &\longmapsto \{1,2,\ldots,m\} \\ E &\mapsto i \end{aligned} \tag{4.6}$$

where i is the index of the input group that E belongs to.

Let P be a fixed polynomial such that

$$P = \sum_{i=1}^{t} p_w w \tag{4.7}$$

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

$$\vartheta = \sup_{\psi \in \mathcal{H},\ \mathcal{S} \subset \mathbb{B}(\mathcal{H})} \langle \psi | P | \psi \rangle \tag{4.8}$$

In addition, in the case where the equality can achieved, we want to extract the quantum state ψ_0 and the set of operators $\mathcal{S}_0$ that yields the optimal value.

By the paper by Pironio et al. (Pironio et al., 2010), we know that ϑ 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 projectors, then consider the following optimization problem at level d where d is such that $2d \geq \deg(P)$:

$$\begin{array}{lll} \omega_d = & \min & \sum_w p_w y_w \\ & \text{subject to} & y_{\mathbb{I}} = 1 \\ & & \mathcal{M}_{W_d} \succeq 0 \\ & & E_a E_b = 0 \text{ if } \mathcal{I}(E_a) = \mathcal{I}(E_b) \text{ and } (E_a)^2 = E_a \text{ otherwise (Orthogonality constraint)} \\ & & \sum_{E_a \in S_i} E_a = \mathbb{I} \text{ (Completeness or Identity constraint)} \\ & & E_a E_b = E_b E_a \text{ if } \mathcal{P}(E_a) \neq \mathcal{P}(E_b) \text{ (Commutativity constraint)} \end{array} \tag{4.9}$$

where w is a monomial of P. 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 $P = \sum_{i=1}^{t} p_w w$ (4.7).
If $\mathcal{S}$ contains observables, then consider the following optimization problem at level d where d is such that $2d \geq \deg(P)$:

$$\begin{array}{lll} \omega_d = & \min & \sum_w p_w y_w \\ & \text{subject to} & y_{\mathbb{I}} = 1 \\ & & \mathcal{M}_{W_d} \succeq 0 \\ & & (E_a)^2 = \mathbb{I} \text{ (Observable constraint)} \\ & & E_a E_b = E_b E_a \text{ if } \mathcal{P}(E_a) \neq \mathcal{P}(E_b) \text{ (Commutativity constraint)} \end{array} \tag{4.10}$$

It is noted that the entry $y_{u^\dagger v}$ in $\mathcal{M}_{W_d}$ in (4.9) or (4.10) corresponds to $\langle\phi| u^\dagger v |\phi\rangle$ where $\phi \in \mathcal{H}$. Therefore, each constraint in (4.9) or (4.10) corresponds to an equality involving entries in the moment matrix.
For example, 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 (4.10) 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 (4.9) or (4.10) and their effect give 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 (4.9) or (4.10) can be transformed into

$$\begin{array}{lll} \omega_d = & \min & \sum_w p_w y_w = tr(P\mathcal{M}_{W_d}) \\ & \text{subject to} & y_{\mathbb{I}} = tr(Q\mathcal{M}_{W_d}) = 1 \\ & & \mathcal{M}_{W_d} \succeq 0 \\ & & tr(Q_i \mathcal{M}_{W_d}) = 0 \end{array} \tag{4.11}$$

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

It can be proved (Wehner et al., 2008) that the sequence $\{\omega_d\}$ is non-increasing and it eventually converges to ϑ, that is,

$$\lim_{d\to\infty} \omega_d = \vartheta \tag{4.12}$$

Chapter 5

Implementation

In this chapter, we discuss our first implementation of the MATLAB package used to solve the quantum bound problem.

5.1 Data Structure for Monomials

Clearly, to solve the optimization problem (4.9) or (4.10), we need to deal with monomials in non-commutative variables and their operations frequently. Therefore, one of the compulsory requirements in the implementation is to find an efficient representation of such monomials. 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.

5.1.1 Monomial Object

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

- varOrdering: a cell [1] to indicate the ordering of variables. The cell has k elements which correspond to k partitions (described in Section refsec:solution). Each element in the cell is an array of integers that represents the ordering of variables in each partition. Note that we map variables to integers so that they can be represented easily.
- degree: an integer to indicate the degree of a monomial.
- coeff: a real number to indicate the coefficient of a monomial.

Example 5.1.1. *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).*

As mentioned in Section refsec:solution, variables in different parition commute; hence, the relative ordering of variables in different partitions does not matter. We arrange the partitions by their indices in the cell. It is worth pointing out that our monomial representation specifically takes into account the commutativity constraint in the SDP (4.9) or (4.10). However, such a monomial representation can still be applied in the general case when we have only one partition.

For consisistency, the zero monomial has coefficient as 0, degree as -2147483648[2], and its variable ordering is an empty cell, that is, {}. Also, the monomial of degree 0 has variable

[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

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 {[] [] []}.

5.1.2 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. In fact, 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 in the cell.

Example 5.1.2. *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 constraint of the variables. It is 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 input group, that monomial becomes zero.

5.1.3 Adjoint of a monomial

This monomial representation also simplifies the process of computing the conjugate transpose (adjoint) of a monomial provided the variables are Hermitian. We simply reverse each array in the cell.

Example 5.1.3. *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: FindAdjoint(M)

```
input : M: a monomial object. It is assumed that M is a monomial of Hermitian
        variables.
output: AdjointM: the adjoint monomial of M.
1 Let AdjointM be a new monomial object
2 $AdjointM.degree \leftarrow M.degree$
3 $AdjointM.coeff \leftarrow M.coeff$
4 Let N be the number of partitions.
5 for $index \leftarrow 1$ to $N$ do
6     $AdjointM.varOrdering(index) \leftarrow$ ReverseList(M.varOrdering(index))
```

5.1.4 Polynomial Object

A polynomial in non-commutative variables is in fact a sum of monomials in non-commutative variables. In our implementation, a polynomial is an object with the following attributes:

- listMonomial: list of monomials.
- degree: the degree of the polynomial.
- varType: 0 (if variables are projectors), 1 (if variables are observables)
- $\mathcal{P}$, $\mathcal{I}$: functions described in Section 4.2. They are used to tell which partition or which input group each variable belongs to.

Currently, polynomial arithmetic is not needed in our solution to the quantum bound problem. Therefore, polynomial operations are not implemented.

5.2 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 16. Details of important steps in the algorithm will be explained in subsequent sections.

5.3 Generating a list of monomials

Based on the optimization problem (4.9) or (4.10), 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 $B_j \in \{B_1, B_2\}$, $A_iB_j = B_jA_i$. Suppose we want to start the SDP at level 2. Then we need to generate $W_2 = \{\mathbb{I}, A_1, A_2, B_1, B_2, A_1A_2, A_2A_1, B_1B_2, B_2B_1, A_1B_1, A_1B_2, A_2B_1, A_2B_2\}$. In fact, this can be done by a simple recursive brute-force algorithm. To improve the performance, we choose to implement an iterative version. Note that again we take into account the orthogonality or observable, and commutativity constraint of the variables.

5.4 Establishing relations between entries in the moment matrix

As we mentioned in Section 4.2, to solve the quantum bound problem, we must transform the optimization problem (4.9) or (4.10) into a semidefinite program of the form (4.11) 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.

5.4.1 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_ω corresponding to a monomial ω fast. This can be done using hashing. In general, we need a hash table that maps a monomial ω to the location of the entry y_ω 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

```
Algorithm 2: FindQuantumBound(PolyData, SDPLevel)
   input :
        • PolyData: an object that represents a polynomial.
        • SDPLevel: It is assumed that $2 * SDPLevel < PolyData.degree$
   output:
        • OptimalVal: the optimal upperbound of the polymonial given.
        • RunLog: information about the execution returned by SeDuMi solver.
   // STEP 1:
 1 Generate a list of monomials of degree less than or equal to SDPLevel. Denote the list as
   $W_{SDPLevel}$
   // ADDITION STEP: Declaration of essential variables:
 2 $lenMonoList \leftarrow$ LengthList($W_{SDPLevel}$)
   // 1) The moment matrix (Here 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
 3 $momentMatrix \leftarrow$ 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 momentMatrix(1, 1) (at row 1, column 1) to be 1
 4 $constraintSet \leftarrow [momentMatrix \succeq 0, momentMatrix(1,1) == 1]$
   // 3) A data structure that maps a monomial to its location in the moment
      matrix
 5 $mapLocation \leftarrow EMPTY$
   // STEP 2:  It involves three tasks described below
 6 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:
 7 Establishing relations between entries in the moment matrix based on identity contraints
   (only applicable to the case variables are projectors. Add these constraints to
   constraintSet
   // STEP 4:
   // Describe the variable we want to maximize
 8 $OptimalVal = 0$
 9 foreach monomial m in PolyData.ListMono do
10     $OptimalVal = OptimalVal + momentMatrix(mapLocation.getData(m))$
   // Invoke SeDuMi solver indirectly through YALMIP function solvesdp
11 solvesdp(constraintSet, -OptimalVal, sdpsettings('solver','sedumi','sedumi.eps',1e-10))
```

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 5.4.1. *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.

5.4.2 Dealing with orthogonality, observable and commutativity constraints

We present the pseudocode of the algorithm to establish orthogonality, observable and commutativity constraint. See Algorithm 3 on page 18. Note that in the pseudocode, for general purposes, mapLocation is an abstract data structure that maps a monomial ω to the location of the entry y_ω 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 ω 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 ω. 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.

5.4.3 Dealing with identity / completeness constraints

Identity (or completeness) constraints are not applicable to observable variables. For the projector case, they are the most difficult constraints to deal with. Before discussing the algorithm we use to generate all identity constraints, let us consider a concrete example.

Example 5.4.2. *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 input groups. 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 ω. Clearly, each instance of A_0^0 in ω gives us an identity constraint. To establish an identity constraint, we replace an instance of A_0^0 by other projectors in the same input group. 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 ω and other monomials obtained by replacing A_0^0 by A_0^1, and the right-hand side is a monomial obtained by removing A_0^0 (due to identity effect). In this example, we have two identity constraints:*

$$\begin{aligned} A_0^0 A_1^1 A_0^0 B_1^0 + A_0^1 A_1^1 A_0^0 B_1^0 &= (A_0^0 + A_0^1) A_1^1 A_0^0 B_1^0 \\ &= \mathcal{I} A_1^1 A_0^0 B_1^0 \\ &= A_1^1 A_0^0 B_1^0 \end{aligned} \tag{5.1}$$

Algorithm 3: 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.
- SDPLevel: It is assumed that $2 * SDPLevel < PolyData.degree$
- $W_{SDPLevel}$: The list of monomials of degree less than or equal to SDPLevel.
- momentMatrix
- 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 $row \leftarrow 1$ to $lenMonoList$ do
2      for $col \leftarrow 1$ to $lenMonoList$ do
           // GetAdjoint(A) returns the adjoint of a matrix A
3          $adjointRow \leftarrow$ GetAdjoint($W_{SDPLevel}(row)$)
           // MultiplyMono(A, B, $\mathcal{P}$, $\mathcal{I}$) returns the result of A * B (A times
              B). It needs the arguments $\mathcal{P}$ and $\mathcal{I}$ described in Section 4.2 to
              simplify the result
4          $mulResult \leftarrow$
           MultiplyMono($adjointRow$, $W_{SDPLevel}(col)$, $PolyData.varType$, $PolyData.\mathcal{P}$, $PolyData.\mathcal{I}$)

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

11             else
12                 $mapLocation.add(mulResult, (row, col))$
```

and

$$\begin{aligned} A_0^0 A_1^1 A_0^0 B_1^0 + A_0^0 A_1^1 A_0^1 B_1^0 &= A_0^0 A_1^1 (A_0^0 + A_0^1) B_1^0 \\ &= A_0^0 A_1^1 \mathcal{I} B_1^0 \\ &= A_0^0 A_1^1 B_1^0 \end{aligned} \tag{5.2}$$

Following the method mentioned in the example 5.4.2, we now state the algorithm we employ in our implementation. Consider the SDP level d. Note that after dealing with orthogonality and commutativity constraints, we obtain all monomials in W_{2d} (i.e. all monomials of degree less than or equal to $2d$).

1. For each input group, we choose any variable in that group to be a representative, namely x_s.
2. We filter all the monomials that contain at least one occurrence of x_s in W_{2d} and store the satisfying monomials in L.
3. Consider a monomial $\omega_s \in L$. Let m be the number of variables in the input group we are considering.
4. For each position of x_s in $\omega_s \in L$, we can obtain an equality regarding the identity constraints by doing the following.
   (a) We replace x_s by other variables in the same input group and obtain new monomials.
   (b) Using the hash table, we can locate the positions of all the resulting monomials. Hence, we obtain the sum of m entries in the moment matrix for the left hand side of the equation.
   (c) To obtain the right hand side, we remove (or delete x_s from the variable ordering of ω_s and simplify the resulting monomial to obtain ω.
   (d) If ω is the zero monomial, we let the left hand side equal to 0.
   (e) If ω is not zero, then again using the hash table, we find the location of the correspoding entry to ω in the moment matrix.
   (f) After that we let the sum on the left hand side equal to that entry in the matrix.
5. Continuing this way, we will establish all equalities regarding identity constraints although the cost is quite expensive.

5.5 Testing

We have tested the software package with the CHSH Inequality (both observable and projector versions), I3322 Inequality up to level 3. So far, the results computed by the software are the same as those that are in the paper by Wehner et al. (Wehner et al., 2008).

Chapter 6

Plan for the second phase of the project

In this chapter, we mention what we plan to do in the second phase of the project.

6.1 Stopping criteria and Extraction of quantum state and measurements

At the moment, when using our package, the user specifies the level d of the SDP he or she wants the program to run at. However, the function does not tell him or her whether the result generated, ω_d, is actually ϑ (the supremum); thus, he or she can stop at this level. In the paper by Pironio et al. (Pironio et al., 2010), for the two-partition inequality, the rank loop method is introduced to detect whether the sequence $\{\omega_d\}$ converges to ϑ 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 look occurs. We will implement rank loop in our software package.

6.2 Improving performance

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 a SDP has no significant importance. However, by reducing the size of the list of monomials, we can reduce the size of the SDP; hence, 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 $d+1$ which is costly. However, we can introduce the concept of 'intermediate' level. That is, after processing level d, we add some specific subset 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 ω_{d+1}, and can be computed faster.

6.3 Improving user-friendliness

Currently, to use the software package, users have to manually map their variables to integer values. They also need to expand the polynomial expression they want to optimize to the sum of monomials. Then they declare monomials and polynomials before calling a function to solve the quantum bound problem. If time permits, we plan to allow users to enter a string expression of the polynomial they want to find its bound. We will create another interface whose aim is to parse the input string provided by users, then to expand it to the sum of monomials. This interface also declares necessary polynomials and monomials, and automatically invokes the quantum bound solver. This will enhance user-friendliness of the software package.

6.4 Using the package to research into open questions

We also plan to use the software package to examine some open questions regarding quantum probabilities. One option is that we will run some experiments with the help of our package to understand the mod-3 game. The second option is to use the package to research into topics regarding uncertainty relations. However, before that, we must extend our software package to deal with more general constraints than those in (4.9) or (4.10).

References

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.

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

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

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.

Sturm, J. Sedumi, a matlab toolbox for optimization over symmetric cones.

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

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.