Quantum algorithms for conformal bootstrap

Ning Bao$^{|\psi\rangle}$ **Junyu Liu**$^{\Delta, J}$

$^{|\psi\rangle}$*Berkeley Center for Theoretical Physics,*
University of California, Berkeley, CA 94720, USA
$^{\Delta}$*Walter Burke Institute for Theoretical Physics,*
California Institute of Technology, Pasadena, CA 91125, USA
^{J}*Institute for Quantum Information and Matter,*
California Institute of Technology, Pasadena, CA 91125, USA

E-mail: ningbao75@gmail.com, jliu2@caltech.edu

ABSTRACT: With the help of recent developments in quantum algorithms for semidefinite programming, we discuss the possibility for quantum speedup for the numerical conformal bootstrap in conformal field theory. We show that quantum algorithms may have significant improvement in the computational performance for several numerical bootstrap problems.

Contents

1   Introduction

Quantum computing is a rapidly growing area both theoretically and experimentally. In the near future, people will be able to manipulate $\mathcal{O}(50) \sim \mathcal{O}(100)$ qubits, a regime which could potentially run algorithms that can outperform classical computers, even with quite high levels of noise. The coming era, the so called Noisy Intermediate-Scale Quantum(NISQ) era, may therefore allow physicists to simulate quantum physics and solve physics problems too difficult for classical computers [2].

One of the earliest motivation for designing quantum computers was the simulation of quantum field theory [1]. With infinite degrees of freedom and complicated mathematical structures, this is a problem that has traditionally been difficult for classical computing. Some interesting work has been pursued as to whether quantum algorithms, by contrast, could be designed to simulate field theory processes, such as time evolution of wave packets, and scattering problems in high energy theory (for example, see [3–7]). Indeed, foundational aspects of quantum algorithmic design (such as complexity theory) could correspond to deep concepts in quantum field theory, holography or quantum gravity (for example, see [8–10]).

For special classes of quantum field theories, such as theories with conformal symmetry, non-perturbative technologies are widely used, that sometimes allow for a solution to the theory through either analytical or numerical techniques. A leading example of this prevalent

in the current high energy theory community is called the conformal bootstrap. The conformal bootstrap is an approach that extracts physical data, such as dimensions of states and Operator Product Expansion (OPE) coefficients, from internal non-trivial consistency relations called crossing equations derived from conformal symmetry. Currently, the conformal bootstrap is widely used in conformal field theory (CFT), string theory and statistical physics (for example, see [11–18]). One of the most important developments in this field has been the advent of the numerical bootstrap as a method for finding high precision numerical solutions for critical components in the three dimensional Ising model, an old but interesting puzzle in statistical physics [11].

For completeness, we review the basic form of the conformal bootstrap problem. In its simplest case, the crossing equation, derived from the crossing symmetry of the conformal four point function, can be written in the following form

$$F_{\Delta_0, J_0}(x) + \sum_{\Delta, J} \lambda_{\Delta, J}^2 F_{\Delta, J}(x) = 0 \tag{1.1}$$

where the x's are particular combinations of spacetime points, F is a known function, and the summation over Δ and J (corresponding to the spectrum of the operator appearing in the OPE) are not known. Alternatively, we could also have known operators in the spectrum. Typically, however, we only know the identity operator, Δ_0 and J_0.

The goal is then to solve the spectrum from such a crossing equation. In general, the function F is complicated enough such that determining the spectrum analytically is intractible. Despite this, however, numerical methods have proven successful for this problem. The particular method used here is based on semidefinite programming, a widely used method for operational research and related areas. We can begin by writing a generic form of the linear functional

$$\alpha = \sum_m \alpha_m \partial_x^m |_{x=x_0} \tag{1.2}$$

and applying this functional to each side of the crossing equation. If for $\Delta > \Delta_*$, one can always find a linear functional such that

$$\alpha \circ F_{\Delta_0, J_0}(x) > 0$$
$$\alpha \circ F_{\Delta, J}(x) > 0 \tag{1.3}$$

then a contradiction is reached. This would therefore mean that there must exist a state with dimension $\Delta < \Delta_*$. Thus, we could search for the minimal value of Δ_*, which corresponds to the gap of the theory. The functional searching with positivity condition, can be transformed to a standard form of semidefinite programming [13].

In this paper, we will address the possibility for quantum speedup for the numerical conformal bootstrap. This is motivated for several reasons. First, some bootstrap problems are extremely difficult to solve (for instance, [19]) on classical computers, and it is possible that quantum computing techniques are better adapted to these questions. Second, solutions of certain numberical bootstrap problems may require higher precision in order to compare against Monte Carlo results and experiments for statistical models. Recently, rapid developments have shown that semidefinite programming indeed has the potential for dramatic quantum speedup [21–24]. We will make use of those algorithms to address specific problems in the numerical conformal bootstrap.

This paper is organized as follows. In Section 2 we will review the conformal bootstrap, the Positive Matrix Program(PMP), and its connection to semidefinite programming(SDP). In Section 3 we will discuss the fasted current quantum algorithms for solving SDPs. In Section 4 we will study the particular speedups in the context of numerical bootstrap problems by quantum algorithms. In Section 5 we will give an explicit example, the relatively well-studied three dimensional Ising bootstrap, to discuss in more specificity the potential for quantum speedup. Finally, in Section 6, we conclude and discuss several future research directions.

2 Conformal bootstrap and semidefinite programming

In this section, we will review some basics about the conformal bootstrap program and semidefinite programming from [13]. The conformal bootstrap problem can typically be written in the following form. Consider a given symmetric polynomial matrix set

$$M_j^n(x) = \begin{pmatrix} P_{j,11}^n(x) & \cdots & P_{j,1m_j}^n(x) \\ \vdots & \ddots & \vdots \\ P_{j,m_j1}^n(x) & \cdots & P_{j,m_jm_j}^n(x) \end{pmatrix} \tag{2.1}$$

where $n \in \{0, 1, \cdots, N\}$ and $j \in \{1, \cdots, J\}$, and the $P_{j,rs}^n(x)$ are all polynomial. Thus, given $b \in \mathbb{R}^N$, one can run the following Positive Matrix Program(PMP):

PMP: maximize: $b \cdot y$ for $y \in \mathbb{R}^N$

such that: $M_j^0(x) + \sum_{n=1}^N y_n M_j^n(x) \succeq 0$ for all $x \geq 0$ and $1 \leq j \leq J$ $\tag{2.2}$

Where the symbol $M \succeq 0$ means that M is positive semi-definite.

Here we can give an example about how the conformal bootstrap can be transformed into the PMP. Consider the 3D Ising CFT. In this theory, the crossing equation for $\langle \sigma\sigma\sigma\sigma \rangle$, $\langle \sigma\sigma\epsilon\epsilon \rangle$ and $\langle \epsilon\epsilon\epsilon\epsilon \rangle$ will be given by

$$\begin{pmatrix} 1 & 1 \end{pmatrix} \vec{V}_{+,0,0} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \sum_{O^+} \begin{pmatrix} \lambda_{\sigma\sigma O} & \lambda_{\epsilon\epsilon O} \end{pmatrix} \vec{V}_{+,\Delta,\ell} \begin{pmatrix} \lambda_{\sigma\sigma O} \\ \lambda_{\epsilon\epsilon O} \end{pmatrix} + \sum_{O^-} \lambda_{\sigma\epsilon O}^2 \vec{V}_{-,\Delta,\ell} = 0 \tag{2.3}$$

Note here that O^+ runs over the $\mathbb{Z}_2$ even operators with even spin, while O^- runs over the $\mathbb{Z}_2$ odd operators with all possible spins. Further, Δ, ℓ are the dimension and spin of the operator O. The vectors $\vec{V}$ is then defined with the following form

$$
\vec{V}_{-,\Delta,\ell} =
\begin{pmatrix}
0 \\
0 \\
F^{\sigma\epsilon,\sigma\epsilon}_{-,\Delta,\ell}(u,v) \\
(-1)^\ell F^{\epsilon\sigma,\sigma\epsilon}_{-,\Delta,\ell}(u,v) \\
-(-1)^\ell F^{\epsilon\sigma,\sigma\epsilon}_{+,\Delta,\ell}(u,v)
\end{pmatrix}
\qquad
\vec{V}_{+,\Delta,\ell} =
\begin{pmatrix}
\begin{pmatrix} F^{\sigma\sigma,\sigma\sigma}_{-,\Delta,\ell}(u,v) & 0 \\ 0 & 0 \end{pmatrix} \\
\begin{pmatrix} 0 & 0 \\ 0 & F^{\epsilon\epsilon,\epsilon\epsilon}_{-,\Delta,\ell}(u,v) \end{pmatrix} \\
\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\
\begin{pmatrix} 0 & \frac{1}{2}F^{\sigma\sigma,\epsilon\epsilon}_{-,\Delta,\ell}(u,v) \\ \frac{1}{2}F^{\sigma\sigma,\epsilon\epsilon}_{-,\Delta,\ell}(u,v) & 0 \end{pmatrix} \\
\begin{pmatrix} 0 & \frac{1}{2}F^{\sigma\sigma,\epsilon\epsilon}_{+,\Delta,\ell}(u,v) \\ \frac{1}{2}F^{\sigma\sigma,\epsilon\epsilon}_{+,\Delta,\ell}(u,v) & 0 \end{pmatrix}
\end{pmatrix}
\tag{2.4}
$$

while the function F is given by a combination of g, the conformal blocks, which are known functions:

$$
F^{ij,kl}_{\mp,\Delta,J}(u,v) = v^{(\Delta_k+\Delta_j)/2} g^{\Delta_{ij},\Delta_{kl}}_{\Delta,J}(u,v) \mp u^{(\Delta_k+\Delta_j)/2} g^{\Delta_{ij},\Delta_{kl}}_{\Delta,J}(v,u)
\tag{2.5}
$$

where we denote $\Delta_{ij} = \Delta_i - \Delta_j$. Now consider a functional vector $\vec{\alpha} = (\alpha^i)$, for $i = \{1,2,\cdots,5\}$. We now act α on the crossing equation

$$
\begin{pmatrix} 1 & 1 \end{pmatrix} \vec{\alpha} \cdot \vec{V}_{+,0,0} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \sum_{O^+} \begin{pmatrix} \lambda_{\sigma\sigma O} & \lambda_{\epsilon\epsilon O} \end{pmatrix} \vec{\alpha} \cdot \vec{V}_{+,\Delta,\ell} \begin{pmatrix} \lambda_{\sigma\sigma O} \\ \lambda_{\epsilon\epsilon O} \end{pmatrix} + \sum_{O^-} \lambda^2_{\sigma\epsilon O} \vec{\alpha} \cdot \vec{V}_{-,\Delta,\ell} = 0
\tag{2.6}
$$

Then, if we could find an α such that

$$
\begin{pmatrix} 1 & 1 \end{pmatrix} \vec{\alpha} \cdot \vec{V}_{+,0,0} \begin{pmatrix} 1 \\ 1 \end{pmatrix} > 0
$$
$$
\vec{\alpha} \cdot \vec{V}_{+,\Delta,\ell} \succeq 0
$$
$$
\vec{\alpha} \cdot \vec{V}_{-,\Delta,\ell} \succeq 0
\tag{2.7}
$$

then the theory is ruled out; this method can therefore be used to rule out some range of Δ's from being in the allowed theory space.

In order to transform this problem to a PMP, we will consider the following linear functional

$$
\alpha^i(f) = \sum_{m \geq n, m+n \leq \Lambda} a^i_{mn} \partial^m_z \partial^n_{\bar{z}} f(z,\bar{z})|_{z,\bar{z}=1/2}
\tag{2.8}
$$

where $u = z\bar{z}$ and $v = (1-z)(1-\bar{z})$. Note that derivatives of function F are well approximated by polynomials, with positive prefactors

$$\partial_z^m \partial_{\bar{z}}^n F_{\pm\Delta,\ell}^{ij,kl}(z,\bar{z})|_{z,\bar{z}=1/2} \approx \chi_\ell(\Delta) P_{\pm,\ell}^{ij,kl;mn}(\Delta) \tag{2.9}$$

where Ps are polynomials and χs are positive prefactors. Thus we arrive at a standard form of the PMP,

$$
\begin{aligned}
\texttt{3DIsing:} \quad &\text{find: } a_{mn}^i \\
&\text{such that: } \begin{pmatrix} 1 & 1 \end{pmatrix} Z_0 \begin{pmatrix} 1 \\ 1 \end{pmatrix} > 0 \\
&Z_\ell(\Delta) \succeq 0 \text{ for all } \mathbb{Z}_2\text{-even operators with even spin} \\
&Y_\ell(\Delta) \succeq 0 \text{ for all } \mathbb{Z}_2\text{-odd operators}
\end{aligned} \tag{2.10}
$$

where

$$Y_\ell(\Delta) \equiv \sum_{mn} \left[a_{mn}^3 P_{-,\ell}^{\sigma\epsilon,\sigma\epsilon;mn}(\Delta) + a_{mn}^4 (-1)^\ell P_{-,\ell}^{\epsilon\sigma,\sigma\epsilon;mn}(\Delta) - a_{mn}^5 (-1)^\ell P_{+,\ell}^{\epsilon\sigma,\sigma\epsilon;mn}(\Delta) \right]$$

$$Z_\ell(\Delta) \equiv \sum_{mn} \begin{pmatrix} a_{mn}^1 P_{-,\ell}^{\sigma\sigma,\sigma\sigma;mn}(\Delta) & \frac{1}{2}\left(a_{mn}^4 P_{-,\ell}^{\sigma\sigma,\epsilon\epsilon;mn}(\Delta) + a_{mn}^5 P_{+,\ell}^{\sigma\sigma,\epsilon\epsilon;mn}(\Delta) \right) \\ \frac{1}{2}\left(a_{mn}^4 P_{-,\ell}^{\sigma\sigma,\epsilon\epsilon;mn}(\Delta) + a_{mn}^5 P_{+,\ell}^{\sigma\sigma,\epsilon\epsilon;mn}(\Delta) \right) & a_{mn}^2 P_{-,\ell}^{\epsilon\epsilon,\epsilon\epsilon;mn}(\Delta) \end{pmatrix} \tag{2.11}$$

Note that Δ could be shifted to some minimal value $\Delta_{\min}(\ell)$ plus x, and that we can demand $x \geq 0$ to replace the requirement that $\Delta \geq \Delta_{\min}(\ell)$.

There are several thing to note in this example. First, we only care about feasibility, and therefore we are allowed to set the objective function to zero in PMP. Other bootstrap problems, for instance the finding of bounds over OPE coefficients, could in general need a nontrivial objective function. Second, the set of spin is infinitely large, and so we should truncate it to be a finite set, for instance ranging from 0 towards $\ell_{\max}$. For the bootstrap, if $\ell_{\max}$ is sufficiently large, the positivity for all spins is typically ensured.

Finally, we will review how to relate PMP's to SDP's. Here we will use SDP's of the following form:

$$
\begin{aligned}
\texttt{SDP:} \quad &\text{maximize: } \text{Tr}(CY) + b \cdot y \text{ on } y \in \mathbb{R}^N, \ Y \in \mathcal{S}^K \\
&\text{such that: } \text{Tr}(A_* Y) + By = c \\
&\text{and } Y \succeq 0
\end{aligned} \tag{2.12}
$$

for given $c \in \mathbb{R}^P$, $B \in \mathbb{R}^{P \times N}$, and $A_1, A_2, \cdots, A_P, C \in \mathcal{S}^K$. Here S^K is the space of the real $K \times K$ symmetric matrix, and we write $A_* = (A_1, A_2, \cdots A_P)$. To transform our PMP problem into this form, we will eventually set $C = 0$, although we will allow for it to remain general

for now.

We quote the theorem for transforming PMP to SDP reviewed and discussed in [13]:

Theorem 2.1. *Given a* PMP *problem, if one sets p to be tuples (i, j, k, l) with $0 \leq r \leq s < m_j$, $0 \leq k \leq d_j$ and $1 \leq j \leq J$, then this tranforms to an* SDP *of the form*

$$
A_{(j,r,s,k)} = \begin{pmatrix} 0 \cdots & & 0 & 0 & \cdots & 0 \\ \vdots \ddots & & \vdots & \vdots & & \vdots \\ 0 \cdots & Q_{\delta_{j1}}(x_k) \otimes E^{rs} & & 0 & \cdots & 0 \\ 0 \cdots & & 0 & x_k Q_{\delta_{j2}}(x_k) \otimes E^{rs} & \cdots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 \cdots & & 0 & 0 & \cdots & 0 \end{pmatrix}
$$

$$
B_{(j,r,s,k),n} = -P_{j,rs}^n(x_k)
$$
$$
c_{(j,r,s,k)} = P_{j,rs}^0(x_k)
$$
$$
C = 0 \tag{2.13}
$$

where $d_j = \max_{n=0}^{N}(\deg(M_j^n(x)))$, $\delta_{j1} = \lfloor d_j/2 \rfloor$ and $\delta_{j2} = \lfloor (d_j - 1)/2 \rfloor$ and the x_k are some chosen points. We also define $(E^{rs})_{ij} = \frac{1}{2}(\delta_i^r \delta_j^s + \delta_i^s \delta_j^r)$ and $Q_\delta(x) = \vec{q}_\delta(x) \vec{q}_\delta(x)^T$, with $\vec{q}_\delta = (q_0(x), \cdots q_\delta(x))$, is a collection of polynomials with degree $0, 1, \cdots, \delta$.

Based on the theorem, we know that the volume of matrices, as input, are given by

$$
P = \mathcal{O}(Jm^2 d)
$$
$$
K = \mathcal{O}(Jmd) \tag{2.14}
$$

where we assume that M is $m \times m$ matrix with maximal degree d, and N in the PMP is the same as that of the SDP.

There are several polynomial time classical algorithms for solving SDP problems. In the current conformal bootstrap community, people often use SDPB, an efficient PMP solver proposed in [13] to solve bootstrap problems based on the above transformation from PMP to SDP. The SDP solver in [13] is based on the exact interior point method. The complexity of the algorithm, within a reasonably small error, is known to be $\mathcal{O}(Jm^6 d^3 + N^2 Jm^2 d + N^3)$.

3 Quantum algorithms

Recently rapid progress has been made in finding a quantum version of an SDP solver. The first quantum SDP solver was giving in [21], and further developments refinements and speedups were found in [23]. In addition, an independent group obtain related results with different formulations and input models [22, 24]. In this section, we will describe two quantum algorithms most following [23] for solving SDP that could be useful for the numerical conformal

bootstrap, while an improvement with a stronger input model in [24] is also mentioned.

When constructing quantum algorithms, we first need to specify how to input the elements of the matrices and vectors for a given SDP into the quantum computer, namely, how to compile the information of matrix and vectors into the quantum circuits. Those are called *Input models* or *Oracles*. The first algorithm we will describe uses an oracle called the plain model, while in the second algorithm uses a quantum input model. Those two oracles, perhaps unsurprisingly, lead to algorithms with different complexities. The quantum input model is a stronger oracle that applies in a more restricted set of cases, and thus the corresponding algorithm is faster in its scaling with certain parameters. If we have a good quantum computer, which oracle one chooses will sensitively depend on the detailed quantum architecture at hand.

Before going into detailed discussion, let us first remark on two issues. First, in the current literature for quantum SDP solvers, people use the language of complex(Hermitian) matrices instead of real(symmetric) matrices. However, with some simple modifications (for instance, replacing the Hermitian matrix with a real symmetric matrix in the proof of theorems, or using a real Gibbs sampler instead of a complex one, etc.), one can easily adapt those algorithms without changing the complexity analysis.

Second, people use different forms to describe equivalent SDP problems. In the following descriptions, the first two algorithms are written in terms of SDP with the form of the Semidefinite Programming (Approximate) Feasibility Problem (SDPFP)

$$\text{SDPFP: Find } \text{Tr}(\hat{A}_{\hat{i}}\hat{X}) \leq \hat{a}_{\hat{i}} + \epsilon \text{ for all } \hat{i} \in \{1, 2, \cdots, \hat{m}\}$$
$$\text{such that } \hat{X} \succeq 0 \qquad \text{Tr}(\hat{X}) = 1$$
$$\text{for given } \epsilon > 0 \ , \ \hat{A}_{\hat{i}} \in \mathcal{S}^{\hat{n}} \ , \ -I \preceq \hat{A}_{\hat{i}} \preceq I \tag{3.1}$$

Here, we use the hat notation to distinguish from the variables we used above. The above conditions define a convex region S_ϵ for given ϵ. The result of the algorithm will be: (1) If $S_0 = \varnothing$ we will output Fail; else, (2) if $S_0 \neq \varnothing$, output an $\hat{X}$. In the above notation, $\hat{m}$ is the number of constraints, $\hat{n}$ is the number of dimensions, and ϵ denotes the additive error of a solution. Further, we use $A \succeq B$ to denote the condition that $A - B$ is positive semidefinite, while similarly $A \preceq B$ denotes that $B - A$ is positive semidefinite. I is the identity matrix. We choose the dimension of matrix $\hat{n}$, number of constraint $\hat{m}$, and the number *sparsity* (namely, take the maximal number of non-zero elements in each row of each matrix) of $\hat{A}_i$ to be $\hat{s}$.

Several versions of SDP problems are equivalent and simply related. We discuss them in Appendix A.

3.1 Quantum SDP solver with plain model

Here we describe our first algorithm, the quantum solver for SDPFP in [23]. This algorithm is based on the following building blocks,

Oracle (plain model): The plain model is a quantum oracle $\mathcal{P}_{\hat{A}}$ that gives the map

$$\left|\hat{j},\hat{k},\hat{l},\hat{z}\right\rangle \rightarrow \left|\hat{j},\hat{k},\hat{l},\hat{z}\oplus(\hat{A}_{\hat{j}})_{\hat{k}\hat{f}_{\hat{j}\hat{k}}(\hat{l})}\right\rangle \tag{3.2}$$

where $\hat{j}\in\{1,2,\cdots,\hat{m}\}$, $\hat{k}\in\{1,2,\cdots,\hat{n}\}$ and $\hat{l}\in\{1,2,\cdots,\hat{s}\}$. Here $\hat{z}$ is a auxiliary bit, and $\hat{A}$ is the bit string representation. $\hat{f}_{\hat{j}\hat{k}}$ is a function with indices $\hat{j}$ and $\hat{k}$ that gives the $\hat{l}$th non-zero element in the row $\hat{k}$.

Trace estimator: The input of the trace estimator is a real symmetric matrix H, satisfying $\|H\|\leq\Gamma$, and a density matrix ρ. Then one can compute $\mathrm{Tr}(H\rho)$, with additive error ϵ, and success rate at least $2/3$. We write $\mathcal{S}_{\mathrm{Tr}}(\hat{s},\Gamma,\epsilon)$ to denote the sample complexity.

Gibbs sampler: The Gibbs sampler takes as input a a real symmetric matrix H, satisfying $\|H\|\leq\Gamma$ and prepares as output a Gibbs state $\rho=\frac{e^{-H}}{\mathrm{Tr}(e^{-H})}$ with additive error ϵ with the help of plain model oracle.

Now based on those building blocks, we will briefly describe the algorithm constructed in [23].

Algorithm 3.2. *Quantum* `SDP` *solver with plain model:*

- *Choose an initial weight matrix $W^{(1)}=I_{\hat{n}}$, and choose $T=\frac{16\hat{n}}{\epsilon^2}$.*

- *Start a loop, that for $t=1$, $t\leq T$, $t=t+1$:*

  - *Apply the Gibbs sampler $\mathcal{O}(\log\hat{m}\times\mathcal{S}_{\mathrm{Tr}}(\frac{\hat{s}\log\hat{n}}{\epsilon^2},\frac{\log\hat{n}}{\epsilon},\epsilon))$ times, and get several $\rho^{(t)}=\frac{W^{(t)}}{\mathrm{Tr}(W^{(t)})}$.*

  - *Use these copies of $\rho^{(t)}$ to find a $\hat{j}^{(t)}$ such that $\mathrm{Tr}(\hat{A}_{\hat{j}^{(t)}}\rho^{(t)})>\hat{a}_{\hat{j}^{(t)}}+\epsilon$, where we use the trace estimator to compute the trace. If we find such $\hat{j}^{(t)}$, we update $M^{(t)}=\frac{1}{2}(I_n-\hat{A}_{\hat{j}^{(t)}})$. If we cannot find it, then we say that $\mathcal{S}_\epsilon$ is not empty and output $\rho^{(t)}$ as $\hat{X}$, the solution.*

  - *We update $W^{(t+1)}=\exp\left(\frac{\epsilon}{4}\times\sum_{\tau=1}^{t}M^{(\tau)}\right)$.*

- *We say that $\mathcal{S}_\epsilon$ is empty.*

We note that there is an important procedure for finding a suitable $\hat{j}^{(t)}$. This procedure can be implemented by a fast quantum OR lemma. For details, see [23].

The gate complexity we need for running this algorithm is given by $\mathcal{O}(\hat{s}^2(\frac{\sqrt{\hat{m}}}{\epsilon^{10}}+\frac{\sqrt{\hat{n}}}{\epsilon^{12}}))$, if we use the real version of the Gibbs sampler from [25] and the trace estimator from [21].

In [24], the corresponding algorithm is similar, but they use a different but stronger input model, the quantum operator model, defined by the following.

Oracle (quantum operator model): For matrices $\hat{A}_{\hat{j}}$, find an operator U such that $\hat{A}_{\hat{j}} = \alpha(\langle 0|^{\otimes a} \otimes I) U_{\hat{j}} (|0\rangle^{\otimes a} \otimes I)$ for some given α and a. The oracle O_U is then defined such that $O_U \left| \hat{j}, \psi \right\rangle = \left| \hat{j} \right\rangle U_{\hat{j}} |\psi\rangle$.

With this input model, the runtime required is reduced to $\mathcal{O}(\hat{s}^2(\frac{\sqrt{\hat{m}}}{\epsilon^4} + \frac{\sqrt{\hat{n}}}{\epsilon^5}))$.

3.2 Quantum SDP solver with quantum input model

Now let us consider another quantum solver, this time with a quantum input model. Taking advantage of this input model, the gate complexity will be reduced for some input parameters.

We assume that $\hat{A}_{\hat{j}}$ can be decomposed as $\hat{A}_{\hat{j}} = \hat{A}_{\hat{j}}^+ - \hat{A}_{\hat{j}}^-$, while $\hat{A}_{\hat{j}}^\pm \succeq 0$. This assumption will always hold when $\hat{A}_{\hat{j}}$ is a real symmetric matrix. In fact, one can take ζ large enough such that $\hat{A}_{\hat{j}} + \zeta I$ is a diagonally dominant matrix. Since it is diagonally dominant and it is symmetric, it is positive semidefinite. Thn we can simply take $\hat{A}_{\hat{j}} + \zeta I = \hat{A}_{\hat{j}}^+$ and $\zeta I = \hat{A}_{\hat{j}}^-$. In our problem, we know the input is bounded by

$$-I \preceq \hat{A}_{\hat{i}} \preceq I \tag{3.3}$$

A naive choice is to set $\zeta = 1$; better decompositions may exist, however, for specific problems.

We will now describe the building blocks of the algorithm.

Oracle (quantum input model for trace of $\hat{A}_{\hat{j}}$): We define a quantum oracle O_{Tr} such that $O_{\mathrm{Tr}} \left| \hat{j}, 0, 0 \right\rangle = \left| \hat{j}, \hat{A}_{\hat{j}}^+, \hat{A}_{\hat{j}}^- \right\rangle$ for all $\hat{j}$.

Oracle (quantum input model for $\hat{A}_{\hat{j}}$): We define a quantum oracle O such that

$$O \left| \hat{j} \right\rangle \left\langle \hat{j} \right| \otimes |0\rangle \langle 0| \otimes |0\rangle \langle 0| O^\dagger = \left| \hat{j} \right\rangle \left\langle \hat{j} \right| \otimes \left| \psi_{\hat{j}}^+ \right\rangle \left\langle \psi_{\hat{j}}^+ \right| \otimes \left| \psi_{\hat{j}}^- \right\rangle \left\langle \psi_{\hat{j}}^- \right| \tag{3.4}$$

where $\left| \psi_{\hat{j}}^\pm \right\rangle$ are purifications of $\frac{\hat{A}_{\hat{j}}^\pm}{\mathrm{Tr}\,\hat{A}_{\hat{j}}^\pm}$.

Oracle (quantum input model for $\hat{a}_{\hat{j}}$): We define a quantum oracle $O_{\hat{a}}$ such that

$$O_{\hat{a}} \left| \hat{j} \right\rangle \left\langle \hat{j} \right| \otimes |0\rangle \langle 0| O_{\hat{a}}^\dagger = \left| \hat{j} \right\rangle \left\langle \hat{j} \right| \otimes \left| \hat{a}_{\hat{j}} \right\rangle \left\langle \hat{a}_{\hat{j}} \right| \tag{3.5}$$

We assume that in the input we have $\hat{A}_{\hat{j}}$ has rank at most $\hat{r}$ and $\mathrm{Tr}(\hat{A}_{\hat{j}}^+) + \mathrm{Tr}(\hat{A}_{\hat{j}}^-) \leq \hat{B}$ where

$\hat{B}$ can be understood as the upper bound of the norm for all inputs. In our problem we have

$$\hat{B} \leq \mathcal{O}(\dim I) = \mathcal{O}(\hat{n}) \tag{3.6}$$

Moreover, we define

Trace estimator: The input of the trace estimator is a state ρ, and the output will give, within success probability $1 - \mathcal{O}(1/\hat{m})$, whether $\text{Tr}(\hat{A}_{\hat{j}}\rho) > \hat{a}_{\hat{j}} + \epsilon$ or $\text{Tr}(\hat{A}_{\hat{j}}\rho) < \hat{a}_{\hat{j}}$. We denote $\mathcal{S}_{\text{Tr}}(\hat{B}, \epsilon)$ as the sample complexity. The estimator is defined based on the quantum input models defined above.

Gibbs sampler: The Gibbs sampler is defined as, we assume $\hat{K} = \hat{K}^+ + \hat{K}^-$, where $\hat{K}^{\pm} = \sum_{\hat{j} \in \hat{S}} \hat{c}_{\hat{j}} A_{\hat{j}}^{\pm}$. Here $\hat{S}$ is a subset of $\{1, 2, \cdots, \hat{m}\}$, and it satisfies $\left| \hat{S} \right| \leq \Phi$, and coefficients $\hat{c}_{\hat{j}} > 0$. Then if we know $\text{Tr}(\hat{K}^+) + \text{Tr}(\hat{K}^-) \leq \hat{B}_{\hat{K}}$, and we have that the rank of $\hat{K}^{\pm}$ is at most $\hat{r}_{\hat{K}}$, then the Gibbs sampler will prepare the Gibbs state $\rho_{\text{G}} = \exp(-\hat{K}) / \text{Tr}(\exp(-\hat{K}))$ with error ϵ, using the quantum input oracles above.

Based on those building blocks we have

Algorithm 3.3. *Quantum* SDP *solver with quantum input model:*

- *Choose an initial weight matrix $W^{(1)} = I_{\hat{n}}$, and choose $T = \frac{16\hat{n}}{\epsilon^2}$.*

- *Start a loop, that for $t = 1$, $t \leq T$, $t = t + 1$:*

    - *Apply the Gibbs sampler $\mathcal{O}(\mathcal{S}_{\text{Tr}}(\hat{B}, \epsilon))$ times, and get several $\rho^{(t)} = \frac{W^{(t)}}{\text{Tr}(W^{(t)})}$.*

    - *Use these copies of $\rho^{(t)}$ to find a $\hat{j}^{(t)}$ such that $\text{Tr}(\hat{A}_{\hat{j}^{(t)}} \rho^{(t)}) > \hat{a}_{\hat{j}^{(t)}} + \epsilon$, where we use the trace estimator to compute the trace. If we find such an $\hat{j}^{(t)}$, we then update $M^{(t)} = \frac{1}{2}(I_n - \hat{A}_{\hat{j}^{(t)}})$. If we cannot find it, then we say that $\mathcal{S}_{\epsilon}$ is not empty and output $\rho^{(t)}$ as $\hat{X}$, the solution.*

    - *We update $W^{(t+1)} = \exp\left(\frac{\epsilon}{4} \times \sum_{\tau=1}^{t} M^{(\tau)}\right)$.*

- *We say that $\mathcal{S}_{\epsilon}$ is empty.*

This algorithm is very efficient based on the Gibbs sampler and the trace estimator that are constructed in [23]. Its complexity is given by

$$\mathcal{O}((\sqrt{\hat{m}} + \text{poly}(\hat{r})) \times \text{poly}(\log \hat{m}, \log \hat{n}, \hat{B}, \epsilon^{-1}))$$
$$\leq \mathcal{O}((\sqrt{\hat{m}} + \text{poly}(\hat{r})) \times \text{poly}(\log \hat{m}, \log \hat{n}, \mathcal{O}(\hat{n}), \epsilon^{-1})) \tag{3.7}$$

A remarkable fact here is that it only has the polylog dependence on $\hat{m}$ and $\hat{n}$ regardless of $\hat{B}$.

4   Specifying bootstrap problems

When discussing the bootstrap problem, we can start from `PMP` problems. For the `PMP` parameter setup, we estimate that the `SDPFP` parameters are (For transformations of several forms of `SDP`, see Appendix A)

$$
\begin{aligned}
\hat{m} &= \mathcal{O}(Jm^2 d) \\
\hat{n} &= \mathcal{O}(Jmd + N)
\end{aligned}
\tag{4.1}
$$

Moreover, assuming a generic `PMP` setup (namely assuming that the inputs of derivatives are not empty), the sparsity is

$$
\hat{s} = \mathcal{O}(d)
\tag{4.2}
$$

Classical algorithms are able to solve `PMP` and `SDP` in polynomial time. The current practical programming tool, `SDPB`, has runtime estimated to be $\mathcal{O}(Jm^6 d^3 + N^2 Jm^2 d + N^3)$. If one uses Arora and Kale's multiplicative weight method [26] to solve `PMP`, it gives $\mathcal{O}(J^2 m^3 d^3 + N Jm^2 d^2)$ if we only care about manifest dependence on numbers, dimensions and sparsities of matrices.

Similarly, the quantum algorithm with plain input model works for the `SDP` problem, and thus works for `PMP` problems. The runtime estimate here would be

$$
\mathcal{O}(d^{2.5} J^{0.5} m + J^{0.5} m^{0.5} d^{2.5} + d^2 N^{0.5})
\tag{4.3}
$$

Thus, generically the quantum algorithm with plain input model will provide a quantum speedup.

For the quantum input model, we know that the rank of the input $\hat{A}_{\hat{j}}$ is at most,

$$
\hat{r} \leq \mathcal{O}(d + N)
\tag{4.4}
$$

Thus in the quantum input model, the time cost will be

$$
\begin{aligned}
&\mathcal{O}((md^{0.5} J^{0.5} + \text{poly}(\hat{r})) \times \text{poly}(\log(J), \log(m), \log(d), \log(N), \hat{B})) \\
&\leq \mathcal{O}((md^{0.5} J^{0.5} + \text{poly}(d + N)) \times \text{poly}(\log(J), \log(m), \log(d), \log(N), \mathcal{O}(Jmd + N)))
\end{aligned}
\tag{4.5}
$$

Now let us consider some specific cases:

- If we assume that $\hat{r}$ and $\hat{B}$ are small enough, in this case the time cost is

$$
\mathcal{O}(md^{0.5} J^{0.5} \times \text{poly}(\log(J), \log(m), \log(d), \log(N)))
\tag{4.6}
$$

  So it is always faster than the two given classical algorithms. For large J, it is slower than the plain input model, but it is faster for large d, m, N (holding all other parameters fixed).

- In the worst case we consider $\hat{r} = \mathcal{O}(d + N)$ and $\hat{B} = \mathcal{O}(Jmd + N)$. We denote the degree of polynomial dependence on $\hat{r}$ and $\hat{B}$ to be d_r and d_B separately. Thus, whether the algorithm is fast or not, is based on the following conditions

Requirement	large d	large J	large m	large N
Faster than Arora-Kale	$d_r + d_B < 2$	$d_B < 1.5$	$d_B < 2$	$d_r + d_B < 1$
Faster than SDPB	$d_r + d_B < 2$	$d_B < 0.5$	$d_B < 5$	$d_r + d_B < 3$
Faster than plain model	$d_r + d_B < 1.5$	not possible	not possible	$d_r + d_B < 0.5$

Now we take a look at the bootstrap problem itself. A typical bootstrap problem will have the parameter set

$$N = \mathcal{O}(\Lambda^2)$$
$$d = \mathcal{O}(\Lambda)$$
$$J = \mathcal{O}(J)$$
$$m = \mathcal{O}(1) \tag{4.7}$$

where Λ is the cutoff of the number derivative. It scales as a square because we have both z and $\bar{z}$. Thus we have complexities

$$\texttt{SDPB}: \mathcal{O}(J\Lambda^3 + \Lambda^6)$$
$$\text{classical Arora-Kale}: \mathcal{O}(J^2\Lambda^3 + J\Lambda^4)$$
$$\text{plain input model}: \mathcal{O}(J^{0.5}\Lambda^{2.5} + \Lambda^3) \tag{4.8}$$

And generically we expect

$$\text{quantum input model}: \mathcal{O}((\Lambda J^{0.5} + \text{poly}_r(\mathcal{O}(\Lambda^2)))\text{poly}(\log J, \log \Lambda, \mathcal{O}(J\Lambda + \Lambda^2))) \tag{4.9}$$

Thus for the plain input model, generically we have speedup for all parameters, while for the quantum input model we have, in the worst case,

Requirement	large Λ	large J
Faster than Arora-Kale	$d_B < 1.5$ and $d_r + d_B < 2$	$d_B < 1.5$
Faster than SDPB	$d_B < 2.5$ and $d_r + d_B < 3$	$d_B < 0.5$
Faster than plain model	$d_B < 1$ and $d_r + d_B < 1.5$	not possible

We should comment briefly on those results. First, we see that requiring a quantum speed up and advantage over the plain input model strongly constrains the allowed quantum oracle algorithms. The reason is that for bootstrap problems, we typically require the input matrix to have a large rank, which effectively negates the exponential advantage of the quantum oracle algorithm. On the other hand, bootstrap problems often have low $\hat{s}$ (namely, the input is very sparse), because the sparsity $\hat{s}$ has a very weak dependence on the large numbers (linear in d). That means that the square root speedup of the plain input model algorithm will be generically robust for such problems.

5 Examples

It is valuable to know how parameters scale in the actual bootstrap problems. We will provide some examples here.

Crossing of $\langle \sigma\sigma\sigma\sigma \rangle$ **in 3D Ising:** Here we consider only the first crossing equation in the `3DIsing` problem. In this special case, positivity of matrices become positivity of numbers. Namely, the problem now becomes

$$
\begin{aligned}
\texttt{3DIsing(reduced):} \quad &\text{find: } \tilde{a}_{mn} \\
&\text{such that } \tilde{Z}_0 > 0 \\
&\tilde{Z}_\ell(\Delta) > 0 \text{ for all } \mathbb{Z}_2\text{-even operators with even spin}
\end{aligned}
\tag{5.1}
$$

where

$$
\tilde{Z}_\ell(\Delta) \equiv \sum_{mn} \tilde{a}_{mn} P_{-,\ell}^{\sigma\sigma,\sigma\sigma;mn}(\Delta)
\tag{5.2}
$$

In this problem, we will label the maximal number of derivatives to be `nmax`, while the maximal spin we take to be `lmax` (The relation between `nmax` and Λ is given by $\Lambda = 2 \times \texttt{nmax} - 1$). For simplicity, we consider `lmax` to be even and take the parameters to not be too small. The parameters in the theory will then scale as the following,

Parameters	Values
J	$0.5 \times \texttt{lmax} + 1$
m	1
d	$2 \times \texttt{nmax} + 18$
N	$0.5 \times \texttt{nmax}^2 + 0.5 \times \texttt{nmax} + 1$
P	$\texttt{nmax} \times \texttt{lmax} + 9.5 \times \texttt{lmax} + 2 \times \texttt{nmax}$
K	$\texttt{nmax} \times \texttt{lmax} + 9.5 \times \texttt{lmax} + 2 \times \texttt{nmax}$
$\hat{m}$	$2 \times \texttt{nmax} \times \texttt{lmax} + 19 \times \texttt{lmax} + 4 \times \texttt{nmax} + 1$
$\hat{n}$	$0.5 \times \texttt{nmax}^2 + \texttt{nmax} \times \texttt{lmax} + 9.5 \times \texttt{lmax} + 2.5 \times \texttt{nmax} + 1$
$\hat{s}$	$\texttt{nmax} + 10$
$\hat{r}$	$0.5 \times \texttt{nmax}^2 + 0.5 \times \texttt{nmax} + 2$

The results agree with out expectation of the generic analysis in Section 4. We plot the `SDPFP` parameter $\hat{m}, \hat{n}, \hat{s}, \hat{r}$ in Figure 1.

The full 3D Ising problem: A more serious analysis can be done for the full `3DIsing` problem. Note that here in the crossing equation we have to consider the possibility that O^+ is equal to σ and O_- is equal to ϵ in the sum. As a result we obtain a similar table,
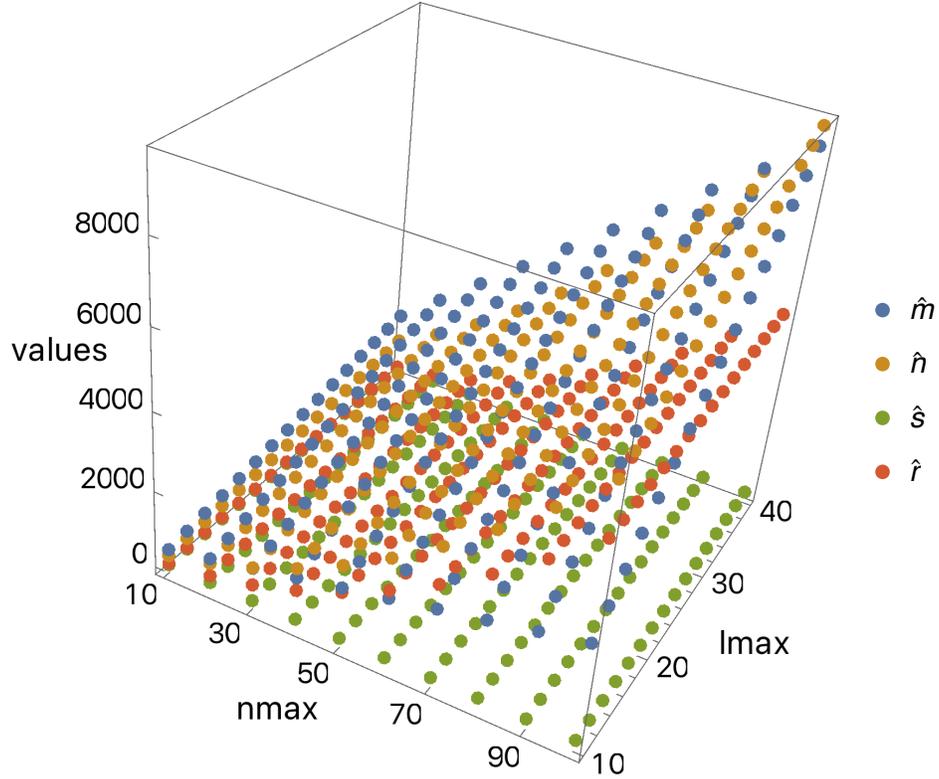
Figure 1: Analytic expression of parameters in `SDPFP` for the `3DIsing(reduced)` setup.

Parameters	Values
J	$1.5 \times \texttt{lmax} + 3$
m	2
d	$2 \times \texttt{nmax} + 19$
N	$2.5 \times \texttt{nmax}^2 + 2.5 \times \texttt{nmax} - 1$
P	$5 \times \texttt{nmax} \times \texttt{lmax} + 48.5 \times \texttt{lmax} + 8 \times \texttt{nmax} - 13$
K	$4 \times \texttt{nmax} \times \texttt{lmax} + 39 \times \texttt{lmax} + 6 \times \texttt{nmax} - 14$
$\hat{m}$	$10 \times \texttt{nmax} \times \texttt{lmax} + 97 \times \texttt{lmax} + 16 \times \texttt{nmax} - 25$
$\hat{n}$	$2.5 \times \texttt{nmax}^2 + 4 \times \texttt{nmax} \times \texttt{lmax} + 39 \times \texttt{lmax} + 8.5 \times \texttt{nmax} - 13$
$\hat{s}$	$\texttt{nmax} + 10$
$\hat{r}$	$\approx 2.5 \times \texttt{nmax}^2 + 2.5 \times \texttt{nmax} + 2$

Similarly, we plot the $\hat{m}, \hat{n}, \hat{s}, \hat{r}$ parameter in Figure 2. The value of N matches the dimension of α coefficients around equation (3.15) of [13]. Moreover, we could use the parameter setup used in Figure 1 of [13], which is given in the following table. As we mentioned before, we see that the sparsity growth is relatively slow (linear in `nmax` with low slope). This helps to retain the advantage of the plain input model, since it depends on sparsity by $\hat{s}^2$, but it depends on $\hat{m}$ and $\hat{n}$ with square roots.
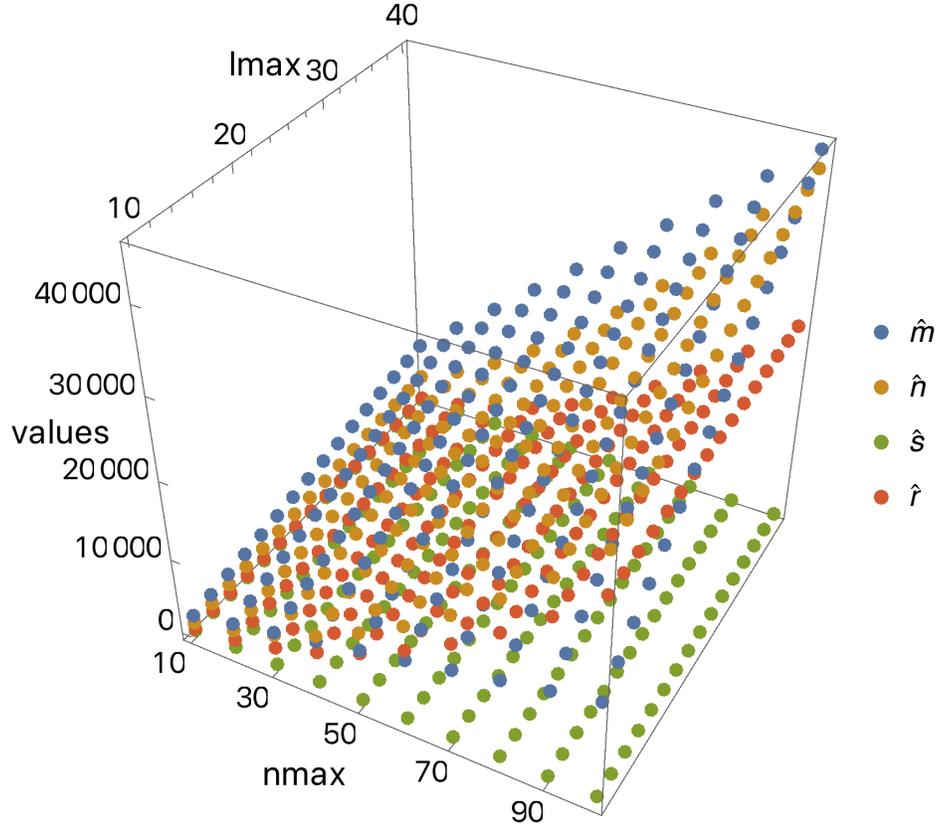
Figure 2: Analytic expression of parameters in `SDPFP` for the `3DIsing` setup.

nmax	10	14	18	22
lmax	28	38	56	76
$\hat{m}$	6×10^3	9×10^3	1×10^4	2×10^4
$\hat{n}$	3×10^3	4×10^3	7×10^3	1×10^4
$\hat{s}$	20	24	28	32
$\hat{r}$	3×10^2	5×10^2	9×10^2	1×10^3

6   Conclusion and outlook

In this paper, we address the possibility of using quantum algorithms for `SDP` problems to numerically solve crossing equations for the conformal bootstrap. Generically, we show that quantum algorithms will provide significant speedup over classical `SDP` algorithms for specific conformal bootstrap setups. We expect that these methods may have possible applications in the coming NISQ era, and beyond, to help the bootstrap community obtain more accurate and faster numerical implementations.

The area of quantum `SDP` is still young. In this paper, we mainly discuss two quantum algo-

rithms. However, in the near future, it is reasonable to believe that more efficient algorithms and more realistic constructions with corresponding quantum architectures will be developed.

We will leave the following possible directions to future research.

- Oracles. It will be important to construct oracles (unitaries) to suppy the bootstrap inputs to the quantum computers, and the complexities of implementing these oracles will also constraint the overall algorithmic complexity at hand.

- Upgrading the Gibbs sampler and the trace estimator. One could potentially develop better methods for speeding up the Gibbs sampling and trace estimation, or think about how to realize them in actual quantum devices.

- More specified solver. The bootstrap problems take a special shape of PMP, which is a special category of SDP. It is therefore hypothetically possible to construct algorithms that taken advantage of this specific from to further speed up the computation.

- Realizations. It might be useful to perform the above algorithms in some quantum simulators, or real quantum devices in the future, to empyrically test whether the theoretical predicitions for runtimes are perhaps too conservative. An attempt to adapt classical SDP solvers for the conformal bootstrap is already underway in [20].

Acknowledgments

A   Transforming different SDP problems

This is a short note on different equivalent forms of SDP1(the form of SDP in the main text). From now on we write

$$
\begin{aligned}
\text{SDP1:}\quad &\text{maximize: } \text{Tr}(C^{(1)}Y^{(1)}) + b^{(1)} \cdot y^{(1)} \text{ on } y^{(1)} \in \mathbb{R}^{N^{(1)}},\ Y^{(1)} \in \mathcal{S}^{K^{(1)}} \\
&\text{such that: } \text{Tr}(A_*^{(1)}Y^{(1)}) + B^{(1)}y^{(1)} = c^{(1)} \\
&\text{and } Y^{(1)} \succeq 0
\end{aligned}
\tag{A.1}
$$

where $A_*^{(1)} = (A_1^{(1)}, \cdots, A_{P^{(1)}}^{(1)})$. Firstly we show that it could be transformed as

$$
\begin{aligned}
\texttt{SDP2:} \quad &\text{maximize: } \mathrm{Tr}(C^{(2)}Y^{(2)}) \text{ on } Y^{(2)} \in \mathcal{S}^{K^{(2)}} \\
&\text{such that: } \mathrm{Tr}(A_*^{(2)}Y^{(2)}) = c^{(2)} \\
&\text{and } Y^{(2)} \succeq 0
\end{aligned} \tag{A.2}
$$

In fact we write $y_i^{(1)} = z_i^{(1)} - s^{(1)}$, where $z^{(1)}$ and $s^{(1)}$ are positive. Then we have

$$
\begin{aligned}
&\mathrm{Tr}(C^{(1)}Y^{(1)}) + b^{(1)} \cdot y^{(1)} \\
&= \mathrm{Tr}(C^{(1)}Y^{(1)}) + b^{(1)} \cdot z^{(1)} - s^{(1)} \sum_i b_i^{(1)} \\
&= \mathrm{Tr}\left(\begin{pmatrix} C^{(1)} & & \\ & \mathrm{diag}(b_i^{(1)}) & \\ & & -\sum_i b_i^{(1)} \end{pmatrix} \begin{pmatrix} Y^{(1)} & ? & ? \\ ? & \mathrm{diag}(z_i^{(1)}) & ? \\ ? & ? & s^{(1)} \end{pmatrix} \right)
\end{aligned} \tag{A.3}
$$

and

$$
\begin{aligned}
&\mathrm{Tr}(A_i^{(1)}Y^{(1)}) + B^{(1)}y^{(1)} - c_i^{(1)} \\
&= \mathrm{Tr}(A_i^{(1)}Y^{(1)}) + B_{ij}^{(1)}y_j^{(1)} - c_i^{(1)} \\
&= \mathrm{Tr}(A_i^{(1)}Y^{(1)}) + B_{ij}^{(1)}z_j^{(1)} - s^{(1)} \sum_j B_{ij}^{(1)} - c_i^{(1)} \\
&= \mathrm{Tr}\left(\begin{pmatrix} A_i^{(1)} & & \\ & \mathrm{diag}(B_{ij}^{(1)}) & \\ & & -\sum_j B_{ij}^{(1)} \end{pmatrix} \begin{pmatrix} Y^{(1)} & ? & ? \\ ? & \mathrm{diag}(z_j^{(1)}) & ? \\ ? & ? & s^{(1)} \end{pmatrix} \right) - c_i^{(1)}
\end{aligned} \tag{A.4}
$$

where ? means it is not necessary to know. So we only need to set

$$
C^{(2)} = \begin{pmatrix} C^{(1)} & & \\ & \mathrm{diag}(b_i^{(1)}) & \\ & & -\sum_i b_i^{(1)} \end{pmatrix}
$$

$$
A^{(2)} = \begin{pmatrix} A_i^{(1)} & & \\ & \mathrm{diag}(B_{ij}^{(1)}) & \\ & & -\sum_j B_{ij}^{(1)} \end{pmatrix}
$$

$$
c^{(2)} = c^{(1)} \tag{A.5}
$$

with

$$
K^{(2)} = K^{(1)} + N^{(1)} + 1 \qquad P^{(2)} = P^{(1)} \tag{A.6}
$$

Now one could also show that it could be converted to

$$\texttt{SDP3:}\quad \text{maximize: } \text{Tr}(C^{(3)}Y^{(3)}) \text{ on } Y^{(3)} \in \mathcal{S}^{K^{(3)}}$$
$$\text{such that: } \text{Tr}(A_*^{(3)}Y^{(3)}) \leq c^{(3)}$$
$$\text{and } Y^{(3)} \succeq 0 \tag{A.7}$$

In fact, if we have a solver for $\texttt{SDP3}$, let us take $A^{(3)}$ to be $(A_i^{(2)}, -A_i^{(2)})$ (Note that now the number of $A^{(3)}$ is doubled), and we set $c^{(3)}$ to be $(c^{(2)}, -c^{(2)})$. That means that we impose the constraint

$$c^{(2)} \leq \text{Tr}(A_*^{(2)}Y^{(2)}) \leq c^{(2)} \tag{A.8}$$

So it gives the equality.

These operations mean that we could directly use the solver $\texttt{SDP3}$ to solve the problem of $\texttt{SDP1}$. We need to just set

$$C^{(3)} = \begin{pmatrix} C^{(1)} & & \\ & \text{diag}(b_i^{(1)}) & \\ & & -\sum_i b_i^{(1)} \end{pmatrix}$$

$$A^{(3)} = \left(\begin{pmatrix} A_i^{(1)} & & \\ & \text{diag}(B_{ij}^{(1)}) & \\ & & -\sum_j B_{ij}^{(1)} \end{pmatrix}, - \begin{pmatrix} A_i^{(1)} & & \\ & \text{diag}(B_{ij}^{(1)}) & \\ & & -\sum_j B_{ij}^{(1)} \end{pmatrix} \right)$$

$$c^{(3)} = (c^{(1)}, -c^{(1)}) \tag{A.9}$$

and we have

$$K^{(3)} = K^{(1)} + N^{(1)} + 1 \qquad P^{(3)} = 2P^{(1)} \tag{A.10}$$

Finally we will show that we could solve $\texttt{SDP1}$ using the $\texttt{SDPFP}$ solver,

$$\texttt{SDPFP:}\quad \text{Find } \text{Tr}(\hat{A}_{\hat{i}}\hat{X}) \leq \hat{a}_{\hat{i}} \text{ for all } \hat{i} \in \{1, 2, \cdots, \hat{m}\}$$
$$\text{such that } \hat{X} \succeq 0 \qquad \text{Tr}(\hat{X}) = 1$$
$$\text{for given } \hat{A}_{\hat{i}} \in \mathcal{S}^{\hat{n}} , \ -I \preceq \hat{A}_{\hat{i}} \preceq I \tag{A.11}$$

We could start from $\texttt{SDP3}$. We notice that for given $A^{(3)}$ there exists a positive number λ such that

$$-\lambda I \preceq A_i^{(3)}, C^{(3)} \preceq \lambda I \tag{A.12}$$

for all i. Then we could also just redefine $A^{(3)}, C^{(3)}$ by $\frac{A^{(3)}, C^{(3)}}{\lambda}$ and we should also redefine $c^{(3)}$ by $\frac{c^{(3)}}{\lambda}$. For optimization, we could using binary search by assuming for instance,

$\text{Tr}(C^{(3)}Y^{(3)}) \geq q_{\text{bin}}$ or namely, $\text{Tr}(-C^{(3)}Y^{(3)}) \leq -q_{\text{bin}}$ and treat this as a component of $A_*^{(3)}$. We could increase q_{bin} until we cannot find the optimal solution.

Finally, for the condition $\text{Tr}(\hat{X}) = 1$, we could treat it as the following. Firstly, the solver for $\text{Tr}(\hat{X}) = 1$ means that we have a solver for $\text{Tr}(\hat{X}) = \hat{\omega}$ when $\hat{\omega}$ is an arbitrary positive number, since we could redefine $\hat{a}_{\hat{i}}$ by $\frac{\hat{a}_{\hat{i}}}{\hat{\omega}}$. For a given SDP3 problem, it does not matter if we add a constraint $\text{Tr}(Y^{(3)}) \leq \hat{\omega}$ if $\hat{\omega}$ is sufficiently large. Then we could use the slack variable trick by adding variable $\mu \geq 0$ and transforming the inequality to the equality $\text{Tr}(Y^{(3)}) + \mu = \hat{\omega}$.

As a conclusion, for given SDP1 problem, we define

$$C^{(3)} = \begin{pmatrix} C^{(1)} & & \\ & \text{diag}(b_i^{(1)}) & \\ & & -\sum_i b_i^{(1)} \end{pmatrix}$$

$$A^{(3)} = \left(\begin{pmatrix} A_i^{(1)} & & \\ & \text{diag}(B_{ij}^{(1)}) & \\ & & -\sum_j B_{ij}^{(1)} \end{pmatrix}, - \begin{pmatrix} A_i^{(1)} & & \\ & \text{diag}(B_{ij}^{(1)}) & \\ & & -\sum_j B_{ij}^{(1)} \end{pmatrix} \right) \tag{A.13}$$

and then we find a large positive λ such that

$$-\lambda I \preceq A_i^{(3)}, C^{(3)} \preceq \lambda I \tag{A.14}$$

for all i. Then we set

$$\hat{A} = \left(\begin{pmatrix} \begin{pmatrix} A_i^{(1)} & & \\ & \text{diag}(B_{ij}^{(1)}) & \\ & & -\sum_j B_{ij}^{(1)} \\ & & & 0 \end{pmatrix} / \lambda, - \begin{pmatrix} A_i^{(1)} & & \\ & \text{diag}(B_{ij}^{(1)}) & \\ & & -\sum_j B_{ij}^{(1)} \\ & & & 0 \end{pmatrix} / \lambda, \\ - \begin{pmatrix} C^{(1)} & & \\ & \text{diag}(b_i^{(1)}) & \\ & & -\sum_i b_i^{(1)} \\ & & & 0 \end{pmatrix} / \lambda \end{pmatrix} \right) \tag{A.15}$$

and

$$\hat{a} = (c^{(1)}/\lambda\hat{\omega}, -c^{(1)}/\lambda\hat{\omega}, -q_{\text{bin}}/\lambda\hat{\omega}) \tag{A.16}$$

where $\hat{\omega}$ could be chosen as sufficiently large, for size of solutions of given SDP1. And we use binary search to choose q_{bin}. And we have

$$\hat{m} = 2P^{(1)} + 1 = \mathcal{O}(P^{(1)})$$
$$\hat{n} = K^{(1)} + N^{(1)} + 2 = \mathcal{O}(K^{(1)} + N^{(1)}) \tag{A.17}$$

References

[1] R. P. Feynman, Int. J. Theor. Phys. **21**, 467 (1982).

[2] J. Preskill, *Quantum* 2, 79 (2018), [arXiv:1801.00862 [quant-ph]].

[3] S. P. Jordan, K. S. M. Lee and J. Preskill, Science **336**, 1130 (2012) [arXiv:1111.3633 [quant-ph]].

[4] S. P. Jordan, K. S. M. Lee and J. Preskill, Quantum Information and Computation 14, 1014-1080 (2014) [arXiv:1112.4833 [hep-th]].

[5] S. P. Jordan, K. S. M. Lee and J. Preskill, arXiv:1404.7115 [hep-th].

[6] S. P. Jordan, H. Krovi, K. S. M. Lee and J. Preskill, arXiv:1703.00454 [quant-ph].

[7] J. Liu, J. Preskill and B. Şahinoğlu, to appear.

[8] A. R. Brown, D. A. Roberts, L. Susskind, B. Swingle and Y. Zhao, Phys. Rev. Lett. **116**, no. 19, 191301 (2016) [arXiv:1509.07876 [hep-th]].

[9] A. R. Brown, D. A. Roberts, L. Susskind, B. Swingle and Y. Zhao, Phys. Rev. D **93**, no. 8, 086006 (2016) [arXiv:1512.04993 [hep-th]].

[10] A. R. Brown, L. Susskind and Y. Zhao, Phys. Rev. D **95**, no. 4, 045010 (2017) [arXiv:1608.02612 [hep-th]].

[11] S. El-Showk, M. F. Paulos, D. Poland, S. Rychkov, D. Simmons-Duffin and A. Vichi, J. Stat. Phys. **157**, 869 (2014) [arXiv:1403.4545 [hep-th]].

[12] F. Kos, D. Poland and D. Simmons-Duffin, JHEP **1411**, 109 (2014) [arXiv:1406.4858 [hep-th]].

[13] D. Simmons-Duffin, JHEP **1506**, 174 (2015) [arXiv:1502.02033 [hep-th]].

[14] F. Kos, D. Poland, D. Simmons-Duffin and A. Vichi, JHEP **1511**, 106 (2015) [arXiv:1504.07997 [hep-th]].

[15] L. Iliesiu, F. Kos, D. Poland, S. S. Pufu, D. Simmons-Duffin and R. Yacoby, JHEP **1603**, 120 (2016) [arXiv:1508.00012 [hep-th]].

[16] Y. H. Lin, S. H. Shao, D. Simmons-Duffin, Y. Wang and X. Yin, JHEP **1705**, 126 (2017) [arXiv:1511.04065 [hep-th]].

[17] D. Poland and D. Simmons-Duffin, Nature Phys. **12**, no. 6, 535 (2016).

[18] D. Poland, S. Rychkov and A. Vichi, arXiv:1805.04405 [hep-th].

[19] A. Dymarsky, F. Kos, P. Kravchuk, D. Poland and D. Simmons-Duffin, JHEP **1802**, 164 (2018) [arXiv:1708.05718 [hep-th]].

[20] N. Bao, F. Kos, B. Lackey and V. Su, in progress.

[21] F. Brandão and K. Svore, Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on. IEEE, 2017.

[22] J. Apeldoorn, et al. Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on. IEEE, 2017.

[23] F. Brandão, et al, [arXiv:1710.02581 [quant-ph]].

[24] J. Apeldoorn and G. Andras, [arXiv:1804.05058 [quant-ph]].

[25] D. Poulin and P. Wocjan, Physical review letters, 2009, 103(22): 220502.

[26] S. Arora and S. Kale. Proceedings of the thirty-ninth annual ACM symposium on Theory of computing. ACM, 2007.