Quantum SDP-Solvers: Better upper and lower bounds

Joran van Apeldoorn*, András Gilyén*, Sander Gribling*, and Ronald de Wolf†
*CWI and QuSoft, Amsterdam, the Netherlands
†CWI, QuSoft and University of Amsterdam, Amsterdam, the Netherlands
Email: {apeldoor,gilyen,gribling,rdewolf}@cwi.nl

***Abstract*—Brandão and Svore [1] recently gave quantum algorithms for approximately solving semidefinite programs, which in some regimes are faster than the best-possible classical algorithms in terms of the dimension n of the problem and the number m of constraints, but worse in terms of various other parameters. In this paper we improve their algorithms in several ways, getting better dependence on those other parameters. To this end we develop new techniques for quantum algorithms, for instance a general way to efficiently implement smooth functions of sparse Hamiltonians, and a generalized minimum-finding procedure.**

We also show limits on this approach to quantum SDP-solvers, for instance for combinatorial optimizations problems that have a lot of symmetry. Finally, we prove some general lower bounds showing that in the worst case, the complexity of every quantum LP-solver (and hence also SDP-solver) has to scale linearly with mn when m is approximately n, which is the same as classical.

***Keywords*-Quantum algorithms, Semidefinite programs, Linear programs, Lower bounds**

I. Introduction

A. Semidefinite programs

In the last decades, particularly since the work of Grötschel, Lovász, and Schrijver [2], *semidefinite programs* (SDPs) have become an important tool for designing efficient optimization and approximation algorithms. SDPs generalize and strengthen the better-known *linear* programs (LPs), but (like LPs) they are still efficiently solvable. The basic form of an SDP is the following:

$$\begin{aligned} \max \quad & \mathrm{Tr}(CX) \\ \text{s.t.} \quad & \mathrm{Tr}(A_j X) \leq b_j \quad \text{for all } j \in [m], \\ & X \succeq 0, \end{aligned} \tag{1}$$

where $[m] := \{1, \ldots, m\}$. The input to the problem consists of Hermitian $n \times n$ matrices $C, A_1, \ldots, A_m$ and reals $b_1, \ldots, b_m$. For normalization purposes we assume $\|C\|, \|A_j\| \leq 1$. The number of constraints is m (we do not count the standard $X \succeq 0$ constraint for this). The variable X of this SDP is an $n \times n$ positive semidefinite (psd) matrix. LPs are the case where all matrices are diagonal.

A famous example is the algorithm of Goemans and Williamson [3] for approximating the size of a maximum cut in a graph $G = ([n], E)$: the maximum, over all subsets S of vertices, of the number of edges between S and its complement $\bar{S}$. Computing MAXCUT(G) exactly is NP-hard. It corresponds to the following integer program

$$\begin{aligned} \max \quad & \frac{1}{2} \sum_{\{i,j\} \in E} (1 - v_i v_j) \\ \text{s.t.} \quad & v_j \in \{+1, -1\} \quad \text{for all } j \in [n], \end{aligned}$$

using the fact that $(1 - v_i v_j)/2 = 1$ if v_i and v_j are different signs, and $(1 - v_i v_j)/2 = 0$ if they are the same. We can relax this integer program by replacing the signs v_j by unit vectors, and replacing the product $v_i v_j$ in the objective function by the dot product $v_i^T v_j$. We can implicitly optimize over such vectors (of unspecified dimension) by explicitly optimizing over an $n \times n$ psd matrix X whose diagonal entries are 1. This X is the Gram matrix of the vectors $v_1, \ldots, v_n$, so $X_{ij} = v_i^T v_j$. The resulting SDP is

$$\begin{aligned} \max \quad & \frac{1}{2} \sum_{\{i,j\} \in E} (1 - X_{ij}) \\ \text{s.t.} \quad & \mathrm{Tr}(E_{jj} X) = 1 \quad \text{for all } j \in [n], \\ & X \succeq 0, \end{aligned} \tag{2}$$

where E_{jj} is the $n \times n$ matrix that has a 1 at the (j, j)-entry, and 0s elsewhere. This SDP is a relaxation of a maximization problem, so it may overshoot the correct value, but Goemans and Williamson showed that an optimal solution to the SDP can be rounded to a cut in G whose size is within a factor ≈ 0.878 of MAXCUT(G) (which is optimal under the Unique Games Conjecture [4]). This SDP can be massaged into the form of (1) by replacing the equality $\mathrm{Tr}(E_{jj} X) = 1$ by inequality $\mathrm{Tr}(E_{jj} X) \leq 1$ (so $m = n$) and letting C be a properly normalized version of the Laplacian of G.

B. Classical solvers for LPs and SDPs

Ever since Dantzig's development of the simplex algorithm for solving LPs in the 1940s [5], much work has gone into finding faster solvers, first for LPs and then also for SDPs. The simplex algorithm for LPs (with some reasonable pivot rule) is usually fast in practice, but has worst-case exponential runtime. Ellipsoid methods and interior-point methods solve LPs and SDPs in polynomial time; they will typically *approximate* the optimal value to arbitrary precision. The best known general SDP-solvers [6] approximate the optimal value OPT with additive error ε in complexity

$$\mathcal{O}(m(m^2 + n^\omega + mns)\,\mathrm{polylog}(m, n, R, 1/\varepsilon)),$$

where $\omega \in [2, 2.373)$ is the (still unknown) matrix multiplication exponent; s is the *sparsity*: the maximal number of non-zero entries per row of the input matrices; and R is an upper bound on the trace of an optimal X.[1] The assumption here is that the rows and columns of the matrices of SDP (1) can be accessed as adjacency lists: we can query, say, the ℓth non-zero entry of the kth row of A_j in constant time.

Arora and Kale [7] gave an alternative way to approximate OPT, using a matrix version of the "multiplicative weights update" method.[2] In Section II-A we will describe their framework in more detail, but in order to describe our result we will start with an overly simplified sketch here. The algorithm goes back and forth between candidate solutions to the primal SDP and to the corresponding *dual* SDP, whose variables are non-negative reals $y_1, \ldots, y_m$:

$$\begin{aligned} \min \quad & b^T y \\ \text{s.t.} \quad & \sum_{j=1}^{m} y_j A_j - C \succeq 0, \\ & y \geq 0. \end{aligned} \tag{3}$$

Under assumptions that will be satisfied everywhere here, strong duality applies: the primal SDP (1) and dual SDP (3) will have the same optimal value OPT. The algorithm does a binary search for OPT by trying different guesses α for it. Suppose we have fixed some α, and want to find out whether α is bigger or smaller than OPT. Start with some candidate solution $X^{(1)}$ for the primal, for example a multiple of the identity matrix ($X^{(1)}$ has to be psd but need not be a *feasible* solution to the primal). $X^{(1)}$ induces the following polytope:

$$\mathcal{P}_\varepsilon(X^{(1)}) := \{y \in \mathbb{R}^m : \ y \geq 0, \ b^T y \leq \alpha, \\ \text{Tr}\left(\Big(\sum_{j=1}^{m} y_j A_j - C\Big) X^{(1)}\right) \geq -\varepsilon\}.$$

Think of this polytope as a relaxation of the feasible region of the dual SDP with the extra constraint that OPT $\leq \alpha$: instead of requiring that $\sum_j y_j A_j - C$ is psd, we merely require its inner product with the particular psd matrix $X^{(1)}$ is not too negative. The algorithm then calls an "oracle" that provides a $y^{(1)} \in \mathcal{P}_\varepsilon(X^{(1)})$, or outputs "fail" if $\mathcal{P}_0(X^{(1)})$ is empty (how to efficiently implement such an oracle depends on the application). In the "fail" case we know there is no dual-feasible y with objective value $\leq \alpha$, so we can increase our guess α for OPT, and restart. In case the oracle produced a $y^{(1)}$, this is used to define a Hermitian matrix $H^{(1)}$ and a new candidate solution $X^{(2)}$ for the primal, which is proportional to $e^{-H^{(1)}}$. Then the oracle for the polytope $\mathcal{P}_\varepsilon(X^{(2)})$ induced by this $X^{(2)}$ is called to produce a candidate $y^{(2)} \in \mathcal{P}_\varepsilon(X^{(2)})$ for the dual (or "fail"), this is used to define $H^{(2)}$ and $X^{(3)}$ proportional to $e^{-H^{(2)}}$, etc.

Surprisingly, the average of $y^{(1)}, y^{(2)}, \ldots$ converges to a nearly-dual-feasible solution. Let R be an upper bound on the trace of an optimal X of the primal, r be an upper bound on the sum of entries of an optimal y for the dual, and w^* be the "width" of the oracle for a certain SDP: the maximum of $\left\|\sum_{j=1}^{m} y_j A_j - C\right\|$ over all psd matrices X and all vectors y that the oracle may output for the corresponding polytope $\mathcal{P}_\varepsilon(X)$. In general we will not know the width of an oracle exactly, but only an upper bound $w \geq w^*$, that may depend on the SDP; this is, however, enough for the Arora-Kale framework. Lemma 4 in Section II-A will show that without loss of generality we may assume the oracle returns a y such that $\|y\|_1 \leq r$. Because we assumed $\|A_j\|, \|C\| \leq 1$, we then have $w^* \leq r+1$ as an easy width-bound. General properties of the multiplicative weights update method guarantee that after $T = \widetilde{\mathcal{O}}(w^2 R^2/\varepsilon^2)$ iterations[3], if no oracle call yielded "fail", then the vector $\frac{1}{T}\sum_{t=1}^{T} y^{(t)}$ is close to dual-feasible and satisfies $b^T y \leq \alpha$. This vector can then be turned into a dual-feasible solution by tweaking its first coordinate, certifying that OPT $\leq \alpha + \varepsilon$, and we can decrease our guess α for OPT accordingly.

The framework of Arora and Kale is really a meta-algorithm, because it does not specify how to implement the oracle. They themselves provide oracles that are optimized for special cases, which allows them to give a very low width-bound for these specific SDPs. For example for the MAXCUT SDP, they obtain a solver with near-linear runtime in the number of edges of the graph. They also observed that the algorithm can be made more efficient by not explicitly calculating the matrix $X^{(t)}$ in each iteration: the algorithm can still be made to work if instead of providing the oracle with $X^{(t)}$, we feed it good estimates of $\text{Tr}(A_j X^{(t)})$ and $\text{Tr}(C X^{(t)})$. Arora and Kale do not describe oracles for general SDPs, but one can get a general classical SDP-solver in their framework with complexity

$$\widetilde{\mathcal{O}}\left(nms\left(\frac{Rr}{\varepsilon}\right)^4 + ns\left(\frac{Rr}{\varepsilon}\right)^7\right). \tag{4}$$

Compared to the complexity of the SDP-solver of [6], this has much worse dependence on R and ε, but better dependence on m and n. Using the Arora-Kale framework is thus preferable over standard SDP-solvers for the case where Rr is small compared to mn, and a rough approximation to OPT (say, small constant ε) is good enough.

C. Quantum SDP-solvers: the Brandão-Svore algorithm

Given the speed-ups that *quantum* computers give over classical computers for various problems [11], [12], [13],

[1] See Lee, Sidford, and Wong [6, Section 10.2 of arXiv version 2], and note that our m, n are their n, m, their S is our mns, and their M is our R. The bounds for other SDP-solvers that we state later also include another parameter r; the assumptions of [6, Theorem 45 of arXiv version 2] imply $r \leq mR$ in their setting, so r is absorbed in their polylog factor.

[2] See also [8] for a subsequent survey; the same algorithm was independently discovered around the same time in learning theory [9], [10].

[3] The $\widetilde{\mathcal{O}}(\cdot)$ notation hides polylogarithmic factors in all parameters.

[14], [15], it is natural to ask whether quantum computers can solve LPs and SDPs more efficiently as well. Very little was known about this, until recently Brandão and Svore [1] discovered quantum algorithms that significantly outperform classical SDP-solvers in certain regimes. Because of the general importance of quickly solving LPs and SDPs, and the limited number of quantum algorithms known, this is a very interesting development.

The key idea of the Brandão-Svore algorithm is to take the Arora-Kale approach and to replace two of its steps by more efficient quantum subroutines. First, given the vectors $y^{(1)}, \ldots, y^{(t-1)}$, it turns out one can use "Gibbs sampling" to prepare the new primal candidate $X^{(t)} \propto e^{-H^{(t-1)}}$ *as a* $\log(n)$*-qubit quantum state* $\rho^{(t)} := X^{(t)}/\mathrm{Tr}(X^{(t)})$ in much less time than needed to compute $X^{(t)}$ as an $n \times n$ matrix. Second, one can implement the oracle for $\mathcal{P}_\varepsilon(X^{(t)})$ based on a number of copies of $\rho^{(t)}$, using those copies to estimate $\mathrm{Tr}(A_j\rho^{(t)})$ and $\mathrm{Tr}(A_j X^{(t)})$ when needed (note that $\mathrm{Tr}(A\rho)$ is the expectation value of operator A for the quantum state ρ). This is based on something called "Jaynes's principle", and requires fewer estimations of the $\mathrm{Tr}\,(A_j\rho)$ quantities. The resulting oracle is weaker than what is used classically, in the sense that it outputs a sample $j \sim y_j/\left\|y\right\|_1$ rather than the whole vector y. However, such sampling still suffices to make the algorithm work (it also means we can assume the vector $y^{(t)}$ to be quite sparse).

Using these ideas, Brandão and Svore obtain a quantum SDP-solver of complexity

$$\widetilde{\mathcal{O}}(\sqrt{mn}s^2R^{32}/\delta^{18}),$$

with *multiplicative* error $1\pm\delta$ for the special case where $b_j \geq 1$ for all $j \in [m]$, and $\mathrm{OPT} \geq 1$ (the latter assumption allows them to convert additive error ε to multiplicative error δ) [1, Corollary 5 in arXiv version 4]. They describe a reduction to transform a general SDP of the form (1) to this special case, but that reduction significantly worsens the dependence of the complexity on the parameters R, r, and δ.

Compared to the runtime (4) of our general instantiation of the original Arora-Kale framework, there are quadratic improvements in both m and n, corresponding to the two quantum modifications made to Arora-Kale. However, the dependence on R, r, s, and $1/\varepsilon$ is much worse than in (4). This quantum algorithm thus provides a speed-up only in regimes where $R, r, s, 1/\varepsilon$ are fairly small compared to mn (finding good examples of such SDPs is an open problem).

D. Our results

In this paper we present two sets of results: improvements to the Brandão-Svore algorithm, and better lower bounds for the complexity of quantum LP-solvers (and hence for quantum SDP-solvers as well).

1) Improved quantum SDP-solver: Our quantum SDP-solver, like the Brandão-Svore algorithm, works by quantizing some aspects of the Arora-Kale algorithm. However, the way we quantize is different and faster than theirs.

First, we give a more efficient procedure to estimate the quantities $\mathrm{Tr}(A_j\rho^{(t)})$ required by the oracle. Instead of first preparing some copies of a Gibbs state $\rho^{(t)} \propto e^{-H^{(t-1)}}$ as a mixed state, we coherently prepare a purification of $\rho^{(t)}$, which can then be used to estimate $\mathrm{Tr}(A_j\rho^{(t)})$ more efficiently using amplitude-estimation techniques. Also, our purified Gibbs sampler has logarithmic dependence on the error, which is exponentially better than the Gibbs sampler of Poulin and Wocjan [16] that Brandão and Svore invoke. Chowdhury and Somma [17] also gave a Gibbs sampler with logarithmic error-dependence, but assuming query access to the entries of $\sqrt{H}$ rather than H itself.

Second, we have a different implementation of the oracle, without using Gibbs sampling or Jaynes's principle (though, as mentioned above, we still use purified Gibbs sampling for approximating the $\mathrm{Tr}(A\rho)$ quantities). We observe that the vector $y^{(t)}$ can be made very sparse: *two* non-zero entries suffice.[4] We then show how we can efficiently find such a 2-sparse vector (rather than merely sampling from it) using two applications of the well-known quantum minimum-finding algorithm of Dürr and Høyer [18].

These modifications both simplify and speed up the quantum SDP-solver, resulting in complexity

$$\widetilde{\mathcal{O}}(\sqrt{mn}s^2(Rr/\varepsilon)^8).$$

The dependence on m, n, and s is the same as in Brandão-Svore, but our dependence on R, r, and $1/\varepsilon$ is substantially better. Note that each of the three parameters R, r, and $1/\varepsilon$ now occurs with the same 8th power in the complexity. This is no coincidence: as we show in our full version [19, Appendix E], these three parameters can all be traded for one another, in the sense that we can massage the SDP to make each one of them small at the expense of making the others proportionally bigger. These trade-offs suggest we should actually think of Rr/ε as *one* parameter of the primal-dual pair of SDPs, not three separate parameters. For the special case of LPs we can improve to

$$\widetilde{\mathcal{O}}(\sqrt{mn}(Rr/\varepsilon)^5).$$

Like in Brandão-Svore, our quantum oracle produces very sparse vectors y, in our case even of sparsity 2. This means that after T iterations, the final ε-optimal dual-feasible vector (which is a slightly tweaked version of the average of the T y-vectors produced in the T iterations) has only $\mathcal{O}(T)$ non-zero entries. Such sparse vectors have some advantages, for example they take much less space to store than arbitrary

[4] Independently of us, Ben David, Eldar, Garg, Kothari, Natarajan, and Wright (at MIT), and separately Ambainis observed that in the special case where all b_i are at least 1, the oracle can even be made 1-sparse, and the one entry can be found using one Grover search over m points (in both cases personal communication 2017). The same happens implicitly in our Section II-C in this case. In general, two non-zero entries are necessary.

$y \in \mathbb{R}^m$. In fact, to get a sublinear running time in terms of m, this is necessary. However, this sparsity of the algorithm's output also points to a weakness of these methods: if *every* ε-optimal dual-feasible vector y has many non-zero entries, then the number of iterations needs to be large. For example, if every ε-optimal dual-feasible vector y has $\Omega(m)$ non-zero entries, then these methods require $T = \Omega(m)$ iterations before they can reach an ε-optimal dual-feasible vector. Since $T = \mathcal{O}\left(\frac{R^2 r^2}{\varepsilon^2} \ln(n)\right)$ this would imply that $\frac{Rr}{\varepsilon} = \Omega(\sqrt{m/\ln(n)})$, and hence many classical SDP-solvers would have a better complexity than our quantum SDP-solver. As we show in Section III, this will actually be the case for families of SDPs that have a lot of symmetry.

2) Tools that may be of more general interest: Along the way to our improved SDP-solver, we developed some new techniques that may be of independent interest.

Implementing smooth functions of a given Hamiltonian: We develop a general technique to apply a function $f(H)$ of a sparse Hamiltonian H to a given state $|\phi\rangle$ (Theorem 8). Roughly speaking, what this means is that we want a unitary circuit that maps $|0\rangle|\phi\rangle$ to $|0\rangle f(H)|\phi\rangle + |1\rangle|*\rangle$. If need be, we can then combine this with amplitude amplification to boost the $|0\rangle f(H)|\phi\rangle$ part of the state. If the function $f : \mathbb{R} \to \mathbb{C}$ can be approximated well by a low-degree Fourier series, then our preparation will be efficient in the sense of using few queries to H and few other gates. The novelty of our approach is that we construct a good Fourier series from a polynomial that approximates f (for example a truncated Taylor series for f). Our Theorem 8 can be easily applied to various smooth functions without using involved integral approximations, unlike previous works.

Here we mostly care about the functions $f(x) = e^{-x}$ and $f(x) = \sqrt{x}$; the first is used for generating purified Gibbs states, and together these two functions are used for estimating quantities like $\mathrm{Tr}(A\rho)$. However, our techniques apply much more generally. For example, they also simplify the analysis of the improved linear-systems solver of Childs et al. [20], where the relevant function is $f(x) = 1/x$. As in their work, the Linear Combination of Unitaries technique of Childs et al. [21], [22], [23] is a crucial tool for us.

A generalized minimum-finding algorithm: Dürr and Høyer [18] showed how to find the minimal value of a function $f : [N] \to \mathbb{R}$ using $\mathcal{O}(\sqrt{N})$ queries to f, by repeatedly using Grover search to find smaller and smaller elements of the range of f. In [19, Theorem 49] we construct a more general minimum-finding procedure, which roughly does the following. Suppose we have a unitary U which prepares a quantum state $U|0\rangle = \sum_{k=1}^{N} |\psi_k\rangle|x_k\rangle$. Our procedure can find the minimum value x_{k^*} among the x_k's that have support in the second register, using roughly $\mathcal{O}(1/\|\psi_{k^*}\|)$ applications of U and U^{-1}. Upon finding the minimal value k^*, the procedure actually outputs the state $|\psi_{k^*}\rangle|x_{k^*}\rangle$. This immediately gives the Dürr-Høyer result as a special case if we take U to produce $U|0\rangle = \frac{1}{\sqrt{N}} \sum_{k=1}^{N} |k\rangle|f(k)\rangle$. Unlike Dürr-Høyer, we need not assume direct query access to the individual values $f(k)$.

More interestingly for us, for a given n-dimensional Hamiltonian H, if we combine our minimum-finder with phase estimation using unitary $U = e^{iH}$ on one half of a maximally entangled state, then we obtain an algorithm for estimating the smallest eigenvalue of H (and preparing its ground state) using roughly $\mathcal{O}(\sqrt{n})$ applications of phase estimation with U. A similar result on approximating the smallest eigenvalue of a Hamiltonian was already shown by Poulin and Wocjan [24], but we improve on their analysis to be able to apply it as a subroutine in our procedure to estimate $\mathrm{Tr}(A_j\rho)$.

3) Lower bounds: What about lower bounds for quantum SDP-solvers? Brandão and Svore already proved that a quantum SDP-solver has to make $\Omega(\sqrt{n} + \sqrt{m})$ queries to the input matrices, for some SDPs. Their lower bound is for a family of SDPs where $s, R, r, 1/\varepsilon$ are all constant, and is by reduction from a search problem.

In this paper we prove lower bounds that are quantitatively stronger in m and n, but for SDPs with non-constant R and r. The key idea is to consider a Boolean function F on $N = abc$ input bits that is the composition of an a-bit majority function with a b-bit OR function with a c-bit majority function. The known quantum query complexities of majority and OR, combined with composition properties of the adversary lower bound, imply that every quantum algorithm that computes this functions requires $\Omega(a\sqrt{bc})$ queries. We define a family of LPs, with constant $1/\varepsilon$ but non-constant r and R (we could massage this to make R or r constant, but not Rr/ε), such that constant-error approximation of OPT computes F. Choosing a, b, and c appropriately, this implies a lower bound of

$$\Omega\left(\sqrt{\max\{n,m\}}\,(\min\{n,m\})^{3/2}\right)$$

queries to the entries of the input matrices for quantum LP-solvers. Since LPs are SDPs with sparsity $s = 1$, we get the same lower bound for quantum SDP-solvers. If m and n are of the same order, this lower bound is $\Omega(mn)$, the same scaling with mn as the classical general instantiation of Arora-Kale (4). In particular, this shows that we cannot have an $O(\sqrt{mn})$ upper bound without simultaneously having polynomial dependence on Rr/ε. The construction of our lower bound implies that for the case $m \approx n$, this polynomial dependence has to be at least $(Rr/\varepsilon)^{1/4}$.

II. An improved quantum SDP-solver

Here we describe our quantum SDP-solver. In Section II-A we describe the framework designed by Arora and Kale for solving semidefinite programs. As in the recent work by Brandão and Svore, we use this framework to design an efficient quantum algorithm for solving SDPs. In particular, we show that the key subroutine needed in the

Arora-Kale framework can be implemented efficiently on a quantum computer. Our implementation uses different techniques than the quantum algorithm of Brandão and Svore, allowing us to obtain a faster algorithm. The techniques required for this are developed in Sections II-B and II-C. In Section II-D we put everything together to prove the main theorem of this section, Theorem 13. See [19, Section 2] for proofs omitted from this section due to space constraints.

Notation/Assumptions: We use $\log$ to denote the logarithm in base 2. We denote the all-zero matrix and vector by 0. Throughout we assume each element of the input matrices can be represented by a bitstring of size $\text{poly}(\log n, \log m)$. We use s as the sparsity of the input matrices, that is, the maximum number of non-zero entries in a row (or column) of any of the matrices $C, A_1, \ldots, A_m$ is s. Recall that for normalization purposes we assume $\|A_1\|, \ldots, \|A_m\|, \|C\| \leq 1$. We furthermore assume that $A_1 = I$ and $b_1 = R$, that is, the trace of primal-feasible solutions is bounded by R (and hence also the trace of primal-optimal solutions is bounded by R). The analogous quantity for the dual SDP (3), an upper bound on $\sum_{j=1}^{m} y_j$ for an optimal dual solution y, will be denoted by r. However, we do not add the constraint $\sum_j y_j \leq r$ to the dual. We will assume $r \geq 1$. For r to be well-defined we have to make the explicit assumption that the optimal solution in the dual is attained. In Section III it will be necessary to work with the best possible upper bounds: we let R^* be the smallest trace of an optimal solution to SDP (1), and we let r^* be the smallest ℓ_1-norm of an optimal solution to the dual. These quantities are well-defined; both the primal and dual optimum are attained: the dual optimum is attained by assumption, and due to the assumption $A_1 = I$, the dual SDP is strictly feasible, hence the optimum in (1) is attained.

Unless specified otherwise, we always consider *additive* error. In particular, an ε-optimal solution to an SDP will be a feasible solution whose value is within error ε of OPT.

Input oracles: We assume sparse black-box access to the elements of the matrices $C, A_1, \ldots, A_m$ defined in the following way: for input $(j, \ell) \in [n] \times [s]$ we can query the location and value of the ℓth non-zero entry in the jth row of the matrix M. Specifically, as described in [23], for each $M \in \{A_1, \ldots, A_m, C\}$ we assume access to an oracle O_M^I, which serves the purpose of sparse access. O_M^I calculates the index : $[n] \times [s] \to [n]$ function, which for input (j, ℓ) gives the column index of the ℓth non-zero element in the jth row. We assume this oracle computes the index "in place":

$$O_M^I |j, \ell\rangle = |j, \text{index}(j, \ell)\rangle. \tag{5}$$

(In the degenerate case where the jth row has fewer than ℓ non-zero entries, $\text{index}(j, \ell)$ is defined to be ℓ together with some special symbol.) Also assume we can apply $(O_M^I)^{-1}$.

We also need another oracle O_M, returning a bitstring representation of M_{ji} for any $j, i \in [n]$:

$$O_M |j, i, z\rangle = |j, i, z \oplus M_{ji}\rangle. \tag{6}$$

This slightly unusual "in place" setup is not too demanding. In particular, if instead we had an oracle that computed the non-zero entries of a row in order, then we could implement both O_M^I and its inverse using $\log(s)$ queries (we can compute ℓ from j and $\text{index}(j, \ell)$ using binary search) [23].

Computational model: As our computational model, we assume a slight relaxation of the usual quantum circuit model: a classical control system that can run quantum subroutines. We limit the classical control system so that its number of operations is at most a polylogarithmic factor bigger than the gate complexity of the quantum subroutines, i.e., if the quantum subroutines use C gates, then the classical control may use at most $\mathcal{O}(C \,\text{polylog}(C))$ operations.

When we talk about gate complexity, we count the number of two-qubit quantum gates needed for implementation of the quantum subroutines. Additionally, we assume for simplicity that there exists a unit-cost QRAM gate that allows us to store and retrieve qubits in a memory, by means of a swap of two registers indexed by another register:

$$\begin{aligned} QRAM : |i, x, r_1, \ldots, r_K\rangle \\ \mapsto |i, r_i, r_1, \ldots, r_{i-1}, x, r_{i+1}, \ldots, r_K\rangle, \end{aligned}$$

where the registers $r_1, \ldots, r_K$ are only accessible through this gate. The QRAM gate can be seen as a quantum analogue of pointers in classical computing. The reason we need QRAM is that we need a data structure that allows efficient access to the non-zero entries of a sum of sparse matrices; for the special case of LPs it is not needed.

A. The Arora-Kale framework for solving SDPs

In this section we give a short introduction to the Arora-Kale framework for solving semidefinite programs. We refer to [7], [8] for a more detailed description and omitted proofs.

The key building block is the Matrix Multiplicative Weights (MMW) algorithm. This can be seen as a strategy for you in the following game between you and an adversary. There is a number of rounds T. In each round you present a density matrix ρ to an adversary, the adversary replies with a loss matrix M satisfying $\|M\| \leq 1$. After each round you have to pay $\text{Tr}(M\rho)$. Your objective is to pay as little as possible. The MMW algorithm is a strategy for you (i.e., an update rule for ρ), that allows you to lose not too much, in a sense that is made precise by the following theorem.

Theorem 1 ([7, Theorem 3.1])**.** *For every adversary, the sequence $\rho^{(1)}, \ldots, \rho^{(T)}$ of density matrices constructed using the Matrix Multiplicative Weights Algorithm satisfies*

$$\begin{aligned} \sum_{t=1}^{T} \text{Tr}\left(M^{(t)} \rho^{(t)}\right) \leq \lambda_{\min}\left(\sum_{t=1}^{T} M^{(t)}\right) \\ + \eta \sum_{t=1}^{T} \text{Tr}\left((M^{(t)})^2 \rho^{(t)}\right) + \frac{\ln(n)}{\eta}. \end{aligned}$$

Arora and Kale use the MMW algorithm to construct an SDP-solver. For that, they construct an adversary who promises to satisfy an additional condition: in each round t, the adversary returns a matrix $M^{(t)}$ whose trace inner product with the density matrix $\rho^{(t)}$ is non-negative. The above theorem shows that then, after T rounds, the average of the adversary's responses satisfies the stronger condition that its smallest eigenvalue is not too negative: $\lambda_{\min}\left(\frac{1}{T}\sum_{t=1}^{T} M^{(t)}\right) \geq -\eta - \frac{\ln(n)}{\eta T}$. More explicitly, the MMW algorithm is used to build a vector $y \geq 0$ such that

$$\frac{1}{T}\sum_{t=1}^{T} M^{(t)} \propto \sum_{j=1}^{m} y_j A_j - C$$

and $b^T y \leq \alpha$. That is, the smallest eigenvalue of the matrix $\sum_{j=1}^{m} y_j A_j - C$ is only slightly below zero and y's objective value is at most α. Since $A_1 = I$, increasing the first coordinate of y makes the smallest eigenvalue of $\sum_j y_j A_j - C$ bigger, so that this matrix becomes psd and hence dual-feasible. By the above we know how much the minimum eigenvalue has to be shifted, and with the right choice of parameters it can be shown that this gives a dual-feasible vector $\overline{y}$ that satisfies $b^T\overline{y} \leq \alpha + \varepsilon$. In order to present the algorithm formally, we require some definitions.

Given a candidate solution $X \succeq 0$ for the primal problem (1) and a parameter $\varepsilon \geq 0$, define the polytope

$$\mathcal{P}_\varepsilon(X) := \{y \in \mathbb{R}^m : y \geq 0,\ b^T y \leq \alpha, \\ \operatorname{Tr}\left(\Big(\sum_{j=1}^{m} y_j A_j - C\Big)X\right) \geq -\varepsilon\}.$$

The Arora-Kale framework for solving SDPs uses the MMW algorithm, where the role of the adversary is taken by an ε-approximate oracle, whose requirements are given in Algorithm 1 below. Much of the work in the Arora-Kale framework lies in implementing this.

Input An $n \times n$ psd matrix X, a parameter ε, and the input matrices and reals of (3).
Output Either the $\mathsf{Oracle}_\varepsilon$ returns a vector y from the polytope $\mathcal{P}_\varepsilon(X)$ or it outputs "fail". It may only output fail if $\mathcal{P}_0(X) = \emptyset$.

Algorithm 1. Requirements for an ε-approximate $\mathsf{Oracle}_\varepsilon$

As we will see later, the runtime of the Arora-Kale framework depends on a property of the oracle called the *width*:

Definition 2 (*Width* of $\mathsf{Oracle}_\varepsilon$)**.** *The* width *of $\mathsf{Oracle}_\varepsilon$ for an SDP is the smallest $w^* \geq 0$ such that for every primal candidate $X \succeq 0$, the vector y returned by $\mathsf{Oracle}_\varepsilon$ satisfies $\left\|\sum_{j=1}^{m} y_j A_j - C\right\| \leq w^*$.*

In practice, the width of an oracle is not always known. However, it suffices to work with an upper bound $w \geq w^*$: as we can see in Meta-Algorithm 2, the purpose of the width is to rescale the matrix $M^{(t)}$ in such a way that it forms a valid response for the adversary in the MMW algorithm.

The following theorem shows the correctness of the Arora-Kale primal-dual meta-algorithm for solving SDPs, stated in Meta-Algorithm 2:

Theorem 3 ([7, Theorem 4.7])**.** *Given an SDP of the form* (1) *with input matrices $A_1 = I, A_2, \ldots, A_m$ and C having operator norm at most* 1*, and input reals $b_1 = R, b_2, \ldots, b_m$. If Meta-Algorithm 2 does not output "fail" in any of the rounds, then the returned vector $\overline{y}$ is feasible for the dual* (3) *with objective value at most $\alpha+\varepsilon$. If $\mathsf{Oracle}_{\varepsilon/3}$ outputs "fail" in the t-th round then a suitably scaled version of $X^{(t)}$ is primal-feasible with objective value at least α.*

The SDP-solver uses $T = \left\lceil \frac{9w^2R^2\ln(n)}{\varepsilon^2} \right\rceil$ iterations. Each iteration has several steps. The most expensive two steps are computing the matrix exponential of the matrix $-\eta H^{(t)}$ and the application of the oracle. Note that the only purpose of computing the matrix exponential is to allow the oracle to compute the values $\operatorname{Tr}(A_j X)$ for all j and $\operatorname{Tr}(CX)$, since the polytope $\mathcal{P}_\varepsilon(X)$ depends on X only through those values. To obtain faster algorithms it is important to note, as was done already by Arora and Kale, that the primal-dual algorithm also works if we provide a (more accurate) oracle with approximations of $\operatorname{Tr}(A_j X)$. In fact, it will be convenient to work with $\operatorname{Tr}(A_j\rho) = \operatorname{Tr}(A_j X)/\operatorname{Tr}(X)$. To be more precise, given a list of reals $a_1, \ldots, a_m, c$ and a parameter $\theta \geq 0$, such that $|a_j - \operatorname{Tr}(A_j\rho)| \leq \theta$ for all j, and $|c - \operatorname{Tr}(C\rho)| \leq \theta$, define the polytope

$$\tilde{\mathcal{P}}(a, c') := \{y \in \mathbb{R}^m :\ y \geq 0, b^T y \leq \alpha, \\ \sum_{j=1}^{m} y_j \leq r, \\ \sum_{j=1}^{m} a_j y_j \geq c'\},$$

where for convenience we denote $a = (a_1, \ldots, a_m)$ and $c' := c - (r+1)\theta$. Notice that $\tilde{\mathcal{P}}$ also contains a new type of constraint: $\sum_j y_j \leq r$. Recall that r is defined as a positive real such that there exists an optimal solution y to SDP (3) with $\|y\|_1 \leq r$. Hence, using that $\mathcal{P}_0(X)$ is a *relaxation* of the feasible region of the dual (with bound α on the objective value), we may restrict our oracle to return only such y:

$$\mathcal{P}_0(X) \neq \emptyset \Rightarrow \mathcal{P}_0(X) \cap \{y \in \mathbb{R}^m : \sum_{j=1}^{m} y_j \leq r\} \neq \emptyset.$$

Due to this restriction and the assumption on the norms of the input matrices, an oracle that always returns a vector with ℓ_1-norm $\leq r$, automatically has a width $w^* \leq r + 1$.

Input The input matrices and reals of SDP (1) and trace bound R. The current guess α of the optimal value of the dual (3). An additive error tolerance $\varepsilon > 0$. An $\frac{\varepsilon}{3}$-approximate oracle $\mathsf{Oracle}_{\varepsilon/3}$ as in Algorithm 1 with width-bound w.

Output Either "Lower" and a vector $\overline{y} \in \mathbb{R}^m_+$ feasible for (3) with $b^T\overline{y} \leq \alpha + \varepsilon$, or "Higher" and a symmetric $n \times n$ matrix X that, when scaled suitably, is primal-feasible with objective value at least α.

$T := \left\lceil \frac{9w^2R^2\ln(n)}{\varepsilon^2} \right\rceil$.

$\eta := \sqrt{\frac{\ln(n)}{T}}$.

$\rho^{(1)} := I/n$

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

  Run $\mathsf{Oracle}_{\varepsilon/3}$ with $X^{(t)} = R\rho^{(t)}$.

  if $\mathsf{Oracle}_{\varepsilon/3}$ outputs "fail" **then**

    return "Higher" and a description of $X^{(t)}$.

  end if

  Let $y^{(t)}$ be the vector generated by $\mathsf{Oracle}_{\varepsilon/3}$.

  Set $M^{(t)} = \frac{1}{w}\left(\sum_{j=1}^m y_j^{(t)} A_j - C\right)$.

  Define $H^{(t)} = H^{(t-1)} + M^{(t)} = \sum_{\tau=1}^t M^{(\tau)}$.

  Update the state matrix:

  $\rho^{(t+1)} := \exp\left(-\eta H^{(t)}\right) / \mathrm{Tr}\left(\exp\left(-\eta H^{(t)}\right)\right)$.

end for

If $\mathsf{Oracle}_{\varepsilon/3}$ does not output "fail" in any of the T rounds, then output the dual solution $\overline{y} = \frac{\varepsilon}{R}e_1 + \frac{1}{T}\sum_{t=1}^T y^{(t)}$ where $e_1 = (1, 0, \ldots, 0) \in \mathbb{R}^m$.

Meta-Algorithm 2. Primal-Dual Algorithm for solving SDPs

The following shows that an oracle that always returns a vector $y \in \tilde{\mathcal{P}}(a, c')$ if one exists, is a $4Rr\theta$-approximate oracle as defined in Algorithm 1.

Lemma 4. *Let $a_1, \ldots, a_m$ and c be θ-approximations of $\mathrm{Tr}(A_1\rho), \ldots, \mathrm{Tr}(A_m\rho)$ and $\mathrm{Tr}(C\rho)$, respectively, where $X = R\rho$. Then the following holds:*

$$\mathcal{P}_0(X) \cap \{y \in \mathbb{R}^m : \sum_{j=1}^m y_j \leq r\} \subseteq \tilde{\mathcal{P}}(a, c') \subseteq \mathcal{P}_{4Rr\theta}(X).$$

We have now seen the Arora-Kale framework for solving SDPs. To obtain a quantum SDP-solver it remains to provide a quantum oracle subroutine. By the above discussion it suffices to set $\theta = \varepsilon/(12Rr)$ and to use an oracle that is based on θ-approximations of $\mathrm{Tr}(A\rho)$ (for $A \in \{A_1, \ldots, A_m, C\}$), since with that choice of θ we have $\mathcal{P}_{4Rr\theta}(X) = \mathcal{P}_{\varepsilon/3}(X)$. In the next section, we first give a quantum algorithm for approximating $\mathrm{Tr}(A\rho)$ efficiently. In Section II-C, we provide an oracle using those estimates based on a simple geometric idea. In Section II-D we conclude with an overview of the runtime of our quantum SDP-solver.

B. Approximating $\mathrm{Tr}(A\rho)$

In this section we give an efficient quantum algorithm to approximate quantities of the form $\mathrm{Tr}(A\rho)$. We are going to work with Hermitian matrices $A, H \in \mathbb{C}^{n\times n}$, such that ρ is the Gibbs state $e^{-H}/\mathrm{Tr}\left(e^{-H}\right)$. Note the analogy with quantum physics: in physics terminology $\mathrm{Tr}(A\rho)$ is simply called the "expectation value of observable A" for a quantum system in a thermal state corresponding to H.

The general approach is to separately estimate $\mathrm{Tr}\left(Ae^{-H}\right)$ and $\mathrm{Tr}\left(e^{-H}\right)$, and then to use the ratio of these as an approximation of $\mathrm{Tr}(A\rho) = \mathrm{Tr}\left(Ae^{-H}\right)/\mathrm{Tr}\left(e^{-H}\right)$. Both estimations are obtained using state preparation to prepare a pure state with a flag, such that the probability that the flag is 0 is proportional to the quantity we want to estimate. We then use amplitude estimation to estimate that probability.

As we will show in Lemma 5, it suffices to construct a unitary $U_{A,H}$ which, if applied to the state $|0\ldots0\rangle$, gives a probability $\frac{\mathrm{Tr}\left((I+A/2)e^{-H}\right)}{4n}$ of measurement outcome 0 for the first qubit. That is:

$$\|(\langle 0| \otimes I)U_{A,H}|0\ldots0\rangle\|^2 = \frac{\mathrm{Tr}\left((I + \frac{A}{2})e^{-H}\right)}{4n}.$$

(To clarify the notation: if $|\psi\rangle$ is a 2-register state, then $(\langle 0| \otimes I)|\psi\rangle$ is the (unnormalized) state in the 2nd register that results from projecting on $|0\rangle$ in the 1st register.)

In practice we will not be able to construct such a $U_{A,H}$ exactly, instead we will construct a $\tilde{U}_{A,H}$ that yields a sufficiently close approximation of the correct probability.

Lemma 5. *Suppose we are given the positive numbers $z \leq \mathrm{Tr}\left(e^{-H}\right)$, $\theta \in (0, 1]$, and unitary circuits $\tilde{U}_{0,H}$ and $\tilde{U}_{A,H}$ (with $\|A\| \leq 1$), each acting on at most q qubits, such that for each $A' \in \{0, A\}$ we have*

$$\left| \left\| (\langle 0| \otimes I)\tilde{U}_{A',H}|0\ldots0\rangle \right\|^2 - \frac{\mathrm{Tr}\left((I + \frac{A'}{2})e^{-H}\right)}{4n} \right| \leq \frac{\theta z}{144n}$$

Then there is a procedure that gives an additive θ-approximation of $\mathrm{Tr}(A\rho)$ with high probability, using $\mathcal{O}\left(\frac{1}{\theta}\sqrt{\frac{n}{z}}\right)$ applications of $\tilde{U}_{A,H}$, $\tilde{U}_{0,H}$ and their inverses, and $\mathcal{O}\left(\frac{q}{\theta}\sqrt{\frac{n}{z}}\right)$ additional gates.

Notice the $1/\sqrt{z} \geq 1/\sqrt{\mathrm{Tr}\left(e^{-H}\right)}$ factor in the complexity statement of this lemma. To make sure this factor is not too large, we would like to ensure $\mathrm{Tr}\left(e^{-H}\right) = \Omega(1)$. This can be achieved by substituting $H_+ = H - \lambda_{\min}(H)I$, where $\lambda_{\min}(H)$ is the smallest eigenvalue of H. It is easy to verify that this will not change the value $\mathrm{Tr}\left(Ae^{-H}/\mathrm{Tr}\left(e^{-H}\right)\right)$.

Below we show how to compute $\lambda_{\min}(H)$ (Section II-B1) and how to apply $\tilde{U}_{A,H}$ (Section II-B2).[5] Using the results

[5]The state-preparation will be the hardest part, because we need not know the diagonalizing bases for A and H, and A and H may not be simultaneously diagonalizable. In the special case where all matrices are diagonal (i.e., the case of LP-solving), we do know these bases, making the proofs simpler and the complexity bounds better.

from those sections we get the following:

Theorem 6. *Let $A, H \in \mathbb{C}^{n\times n}$ be Hermitian matrices such that $\|A\| \leq 1$ and $\|H\| \leq K$ for a known bound $K \in \mathbb{R}_+$. Assume A is s-sparse and H is d-sparse with $s \leq d$. We can compute an additive θ-approximation of*

$$\mathrm{Tr}\left(A\rho\right) = \frac{Ae^{-H}}{\mathrm{Tr}\left(e^{-H}\right)}$$

using $\widetilde{\mathcal{O}}\left(\frac{\sqrt{n}dK}{\theta}\right)$ queries (to A and H) and other gates.

Proof: Start by computing an estimate $\tilde{\lambda}_{\min}$ of $\lambda_{\min}(H)$, up to additive error $\varepsilon = 1/2$ using Lemma 7 (below). Define $H_+ := H - (\tilde{\lambda}_{\min} - 3/2)I$, so that $I \preceq H_+$ but $2I \not\preceq H_+$. Applying Lemma 9 (below) and then Lemma 5 to A, H_+ with $z = e^{-2}$ gives the bound. ∎

1) Computing minimum eigenvalues: As mentioned in Section I-D2, we developed new techniques that generalize minimum-finding. The lemma below applies these techniques to the problem of finding the minimum eigenvalue of a Hamiltonian H. Poulin and Wocjan [24] gave a similar complexity for minimum-eigenvalue estimation, but we improve on their analysis to fit our framework better. We assume sparse oracle access to the Hamiltonian H as described in Section II, and count queries to these oracles.

Lemma 7. *If $H = \sum_{j=1}^{n} E_j|\phi_j\rangle\langle\phi_j|$, with eigenvalues $E_1 \leq E_2 \leq \ldots \leq E_n$, satisfies $\|H\| \leq K$, $\varepsilon \leq K/2$, and H is given in d-sparse oracle form, then we can compute an E such that with probability $\geq 2/3$, $|E_1 - E| \leq \varepsilon$, using*

$$\mathcal{O}\left(\frac{Kd\sqrt{n}}{\varepsilon}\mathrm{polylog}\left(\frac{Kn}{\varepsilon}\right)\right) \text{ queries and gates.}$$

2) Applying smooth functions of Hamiltonians: To construct a circuit for $\tilde{U}_{A,H}$, the basic idea is that we first prepare a maximally entangled state $\sum_{i=1}^{n}|i\rangle|i\rangle/\sqrt{n}$, and then apply the (norm-decreasing) maps $e^{-H/2}$ and $\sqrt{\frac{I+A/2}{4}}$ to the first register, so that we end up with a state

$$\left(\sum_{i=1}^{n}\left(\sqrt{\frac{I+A/2}{4}}e^{-H/2}|i\rangle\right)|i\rangle/\sqrt{n}\right)|0\rangle + |\phi\rangle|1\rangle.$$

One can verify that the probability of measuring the flag 0 is indeed $\mathrm{Tr}\left(\frac{I+A/2}{4n}e^{-H}\right)$. These two norm-decreasing maps both correspond to smooth functions. Instead of providing a circuit for each of them separately, we give a general result:

Theorem 8 (Implementing a smooth function of a Hamiltonian)**.** *Let $x_0 \in \mathbb{R}$ and $r > 0$ be such that $f(x_0 + x) = \sum_{\ell=0}^{\infty} a_\ell x^\ell$ for all $x \in [-r, r]$. Suppose $B > 0$ and $\delta \in (0, r]$ are such that $\sum_{\ell=0}^{\infty}(r+\delta)^\ell|a_\ell| \leq B$. If $H \in \mathbb{C}^{n\times n}$ is a Hermitian matrix such that $\|H - x_0 I\| \leq r$, and $\varepsilon \in \left(0, \frac{1}{2}\right]$, then we can implement a unitary $\tilde{U}$ such that $\left\|(\langle 0|\otimes I)\tilde{U}(|0\rangle\otimes I) - \frac{f(H)}{B}\right\| \leq \varepsilon$, using $\mathcal{O}\left(\frac{r}{\delta}\log\left(r/(\delta\varepsilon)\right)\log\left(1/\varepsilon\right)\right)$ gates and a single use of a circuit that can do controlled simulation of H (with error $\leq \varepsilon/2$ in operator norm) for up to $\mathcal{O}(\frac{r}{\delta}\log(1/\varepsilon))$ time-steps each of duration $\mathcal{O}(1/r)$.*

If $\|H\| \leq K$, H is d-sparse and is accessed via oracles (5)-(6)*, and $r = \mathcal{O}(K)$, then (based on the Hamiltonian simulation of [23]) this $\tilde{U}$ can be implemented using*

$$\mathcal{O}\left(\frac{Kd}{\delta}\log\left(\frac{K}{\delta\varepsilon}\right)\log\left(\frac{1}{\varepsilon}\right)\right) \text{ queries and}$$

$$\mathcal{O}\left(\frac{Kd}{\delta}\log\left(\frac{K}{\delta\varepsilon}\right)\log\left(\frac{1}{\varepsilon}\right)\left[\log(n)+\log^{\frac{5}{2}}\left(\frac{K}{\delta\varepsilon}\right)\right]\right) \text{ gates.}$$

Applying to $f(x) = e^{-x/2}$ and $f(x) = \sqrt{1 + x/2}$ gives:

Lemma 9. *Let $A, H \in \mathbb{C}^{n\times n}$ be Hermitian matrices such that $\|A\| \leq 1$ and $I \preceq H \preceq KI$ for a known $K \in \mathbb{R}_+$. Assume A is s-sparse and H is d-sparse with $s \leq d$. For every $\mu > 0$, there exists a unitary $\tilde{U}_{A,H}$ such that*

$$\left|\left\|(\langle 0|\otimes I)\tilde{U}_{A,H}|0\ldots 0\rangle\right\|^2 - \mathrm{Tr}\left(\frac{I+\frac{A}{2}}{4n}e^{-H}\right)\right| \leq \mu$$

that uses $\widetilde{\mathcal{O}}\left(Kd\right)$ queries (to A and H) and other gates.

C. An efficient 2-sparse oracle

In this section we describe our quantum oracle. Remember that a_j is an additive θ-approximation to $\mathrm{Tr}\left(A_j\rho\right)$, c is a θ-approximation to $\mathrm{Tr}\left(C\rho\right)$ and $c' = c - r\theta - \theta$. Due to the results from the last section we may now assume access to an oracle O_a that computes the entries a_j of $a = (a_1, \ldots, a_m)$. Our goal is now to find a $y \in \tilde{\mathcal{P}}(a, c')$, i.e., a y such that

$$\begin{aligned} b^T y &\leq \alpha \\ a^T y &\geq c' \\ \|y\|_1 &\leq r \\ y &\geq 0 \end{aligned}$$

If $\alpha \geq 0$ and $c' \leq 0$, then $y = 0$ is a solution. If not, then write $y = Nq$ with $N = \|y\|_1 > 0$ and hence $\|q\|_1 = 1$. So we want an N and q such that

$$\begin{aligned} b^T q &\leq \alpha/N \\ a^T q &\geq c'/N \\ \|q\|_1 &= 1 \\ q &\geq 0 \\ 0 < N &\leq r \end{aligned} \tag{7}$$

We can now view $q \in \mathbb{R}_+^m$ as the coefficients of a convex combination of the points $p_i = (b_i, a_i)$ in the plane. We want such a combination that lies to the upper left of $g_N = (\alpha/N, c'/N)$ for some $0 < N \leq r$. Let $\mathcal{G}_N$ denote the upper-left quadrant of the plane starting at g_N.

Lemma 10. *If there is a $y \in \tilde{\mathcal{P}}(a, c')$, then there is a 2-sparse $y' \in \tilde{\mathcal{P}}(a, c')$ such that $\|y\|_1 = \|y'\|_1$.*

This shows that we can restrict our search to 2-sparse y. Let $\mathcal{G} = \bigcup_{N\in(0,r]} \mathcal{G}_N$. We want to find two points p_j, p_k such that their convex combination lies in $\mathcal{G}$, since this implies that a scaled version of their convex combination gives a $y \in \tilde{\mathcal{P}}(a, c')$ and $\|y\|_1 \leq r$.

Lemma 11. *There is an oracle that returns a 2-sparse vector $y \in \tilde{\mathcal{P}}(a, c')$, if one exists, using one search and two minimizations over the m points $p_j = (b_j, a_j)$.*

Proof sketch: The algorithm is as follows:

1) Check if $\alpha \geq 0$ and $c' \leq 0$. If so, output $y = 0$.
2) Check if $\exists i : p_i \in \mathcal{G}$. If so, let $q = e_i$ and $N = \frac{c'}{a_i}$.
3) Find p_j, p_k so that the line segment $\overline{p_j p_k}$ goes through $\mathcal{G}$. This gives coefficients q of a convex combination that can be scaled by $N = \frac{c'}{a^T q}$ to get y.

An example of $\mathcal{G}$, when $\alpha, c' > 0$, is drawn in Figure 3. In general $\mathcal{G}$ is always the intersection of at most 2 halfspaces, hence steps 1-2 of the algorithm are easy to perform when given access to the coordinates of the points p_j.

It remains to consider step 3. Denote the two edges of $\mathcal{G}$ by L_1 and L_2. Furthermore, let ℓ_j and ℓ_k be the lines from $(\alpha/r, c'/r)$ to p_j and p_k. Looking at Figure 4, it is clear that the line $\overline{p_j p_k}$ intersects with $\mathcal{G}$ if and only if $\angle \ell_j L_1 + \angle L_1 L_2 + \angle L_2 \ell_k \leq \pi$. In particular, if any choice of j and k will cause $\overline{p_j p_k}$ to intersect with $\mathcal{G}$, then so will the choice that minimizes $\angle \ell_j L_1 + \angle L_1 L_2 + \angle L_2 \ell_k$. Clearly we can minimize this expression by separately minimizing $\angle \ell_j L_1$ and $\angle L_2 \ell_k$. Hence one search and two minimizations using the coordinates of the p_j suffice to implement the oracle. ■

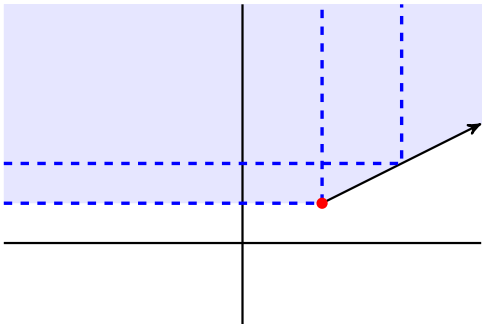

Figure 3. The region $\mathcal{G}$ in light blue. The borders of two quadrants $\mathcal{G}_N$ have been drawn by thick dashed blue lines. The red dot at the beginning of the arrow is the point $(\alpha/r, c'/r)$.

Corollary 12. *There is a quantum algorithm that returns a vector $y \in \tilde{\mathcal{P}}(a, c')$, if one exists, using $\mathcal{O}(\sqrt{m})$ calls to the subroutine for the entries of a, and $\widetilde{\mathcal{O}}(\sqrt{m})$ other gates.*

D. Total runtime

We can now add our quantum trace calculators and the oracle to the classical Arora-Kale framework.

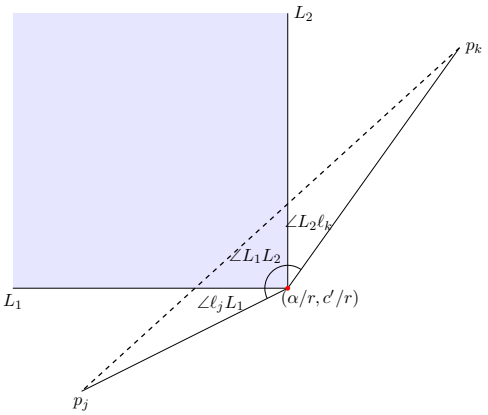

Figure 4. Illustration of $\mathcal{G}$ with the points p_j, p_k and the angles $\angle \ell_j L_1, \angle L_1 L_2, \angle L_2 \ell_k$ drawn in. Clearly the line $\overline{p_j p_k}$ only crosses $\mathcal{G}$ when the total angle is less than π.

Theorem 13. *Instantiating Meta-Algorithm 2 using the trace calculation algorithm from Section II-B and the oracle from Section II-C (with width-bound $w := r + 1$), and using this to do a binary search for* OPT $\in [-R, R]$ *(using different guesses α for* OPT*), gives a quantum algorithm for solving SDPs of the form* (1)*, which (with high probability) produces a feasible solution y to the dual program that is optimal up to an additive error ε, using*

$$\widetilde{\mathcal{O}}\left(\sqrt{nm}s^2\left(\frac{Rr}{\varepsilon}\right)^8\right)$$

queries (to the input matrices) and other gates.

Proof: Using our implementations of the different building blocks, it remains to calculate what the total complexity will be when they are used together.

Cost of the oracle for $H^{(t)}$: The first problem in each iteration is to obtain access to an oracle for $H^{(t)}$. In each iteration the oracle will produce a $y^{(t)}$ that is at most 2-sparse, and hence in the $(t+1)$th iteration, $H^{(t)}$ is a linear combination of $2t$ of the A_j matrices, and the C matrix.

We can write down a sparse representation of the coefficients of the linear combination that gives $H^{(t)}$ in each iteration by adding the new terms coming from $y^{(t)}$. This will clearly not take longer than $\widetilde{\mathcal{O}}(T)$, since there are only a constant number of terms to add. As we will see, this term will not dominate the complexity of the full algorithm.

Using such a sparse representation of the coefficients, one query to a sparse representation of $H^{(t)}$ will cost $\widetilde{\mathcal{O}}(st)$ queries to the input matrices, and $\widetilde{\mathcal{O}}(st)$ other gates.

Cost of the oracle for $\mathrm{Tr}(A_j\rho)$*:* In each iteration $M^{(t)}$ is made to have operator norm at most 1. This means that

$$\left\|-\eta H^{(t)}\right\| \leq \eta \sum_{\tau=1}^{t} \left\|M^{(\tau)}\right\| \leq \eta t.$$

Furthermore we know that $H^{(t)}$ is at most $d := s(2t+1)$-sparse. Calculating $\mathrm{Tr}\,(A_j\rho)$ for one index j up to an additive error of $\theta := \varepsilon/(12Rr)$ can be done using the algorithm from Theorem 6. This will take

$$\widetilde{\mathcal{O}}\left(\sqrt{n}\frac{\|H\|\,d}{\theta}\right) = \widetilde{\mathcal{O}}\left(\sqrt{n}s\frac{\eta t^2 Rr}{\varepsilon}\right)$$

queries to the oracle for $H^{(t)}$ and the same number of other gates. Since each query to $H^{(t)}$ takes $\widetilde{\mathcal{O}}\,(st)$ queries to the input matrices, this means that

$$\widetilde{\mathcal{O}}\left(\sqrt{n}s^2\frac{\eta t^3 Rr}{\varepsilon}\right)$$

queries to the input matrices will be made, and the same number of other gates, for each approximation of a $\mathrm{Tr}\,(A_j\rho)$ (and similarly for approximating $\mathrm{Tr}\,(C\rho)$).

Total cost of one iteration: Corollary 12 tells us that we will use $\widetilde{\mathcal{O}}\,(\sqrt{m})$ calculations of $\mathrm{Tr}\,(A_j\rho)$, and the same number of other gates, to calculate a classical description of a 2-sparse $y^{(t)}$. This brings the total cost of one iteration to

$$\widetilde{\mathcal{O}}\left(\sqrt{nm}s^2\frac{\eta t^3 Rr}{\varepsilon}\right)$$

queries (to the input matrices) and other gates.

Total quantum runtime for SDPs: Since $w \leq r+1$ we can set $T = \widetilde{\mathcal{O}}\left(\frac{R^2r^2}{\varepsilon^2}\right)$. With $\eta = \sqrt{\frac{\ln(n)}{T}}$, summing over all iterations in one run of the algorithm gives total cost

$$\widetilde{\mathcal{O}}\left(\sqrt{nm}s^2\frac{\eta T^4 Rr}{\varepsilon}\right) = \widetilde{\mathcal{O}}\left(\sqrt{nm}s^2\left(\frac{Rr}{\varepsilon}\right)^8\right)$$

queries (to the input matrices) and other gates. ∎

We want to stress again that our solver is meant to work for *all* SDPs. In particular, it does not use the structure of a specific SDP. As we show in the next section, every oracle that works for all SDPs must have large width. To obtain quantum speedups for a *specific* class of SDPs, it will be necessary to develop oracles tuned to that problem. We view this as an important direction for future work. Recall from the introduction that Arora and Kale also obtain fast classical algorithms for problems such as MAXCUT by doing exactly that: they develop specialized oracles for those problems.

III. Downside of this method: general oracles are restrictive

In this section we give two examples illustrating the limitations of a method that uses sparse or general oracles, i.e., ones that are not optimized for the properties of specific SDPs. To illustrate the problem with sparse oracles we consider the classical LP problem (s,t)-maxflow-mincut. Next, we will show that general oracles are bad for solving the Goemans-Williamson SDP relaxation for MAXCUT.

A. Sparse oracles are restrictive for (s,t)-maxflow-mincut

Given a graph $G = (V, E)$ and two vertices $s, t \in V$, the (s,t)-maxflow-mincut problem is to compute the maximum flow that can be sent through G, starting at s and ending at t. Equivalently, one can compute the minimum cut of G with s and t on different sides of the cut. We consider a simple instance of this problem: the union of two complete graphs each of size $z+1$, where s is in one sub-graph and t in the other. The other vertices will be labeled by $\{1, 2, \ldots, 2z\}$. Every partition of the label-set over the two halves gives a unique mincut of value 0 (namely the one which separates the two complete graphs), and every other partition cuts at least z edges. Hence a $z/2$-approximate mincut must be the unique mincut. Since every partition of the $2z$ labels over the two halves is a different problem instance, there are $\binom{2z}{z}$ instances that each require a different unique output.

Now assume we have a family of LPs, one for each problem instance, with the following two properties:

1) There is a function f such that for a $z/2$-approximate dual solution y to one of the LPs, $f(y) = S \subseteq V$ is a $z/2$-approximate mincut for the corresponding instance.
2) The LPs corresponding to the different permutations of the labels only differ by permutations of their constraints and variables.

The first condition states that a dual solution is enough to find a mincut. The second condition states that the LPs treat all points equally (except s and t). Both these conditions hold for the standard (s,t)-maxflow-mincut LP.

A direct consequence of these conditions is that the $\binom{2z}{z}$ dual solutions for the LPs are all just permutations of each other. However, they all need to be different since their image under f (the corresponding mincut) has to be different. A k-sparse vector in $\mathbb{R}^m$ can have k different non-zero entries and hence the number of possible unique permutations of that vector is at most

$$\binom{m}{k}k! = \frac{m!}{(m-k)!} \leq m^k.$$

Hence, using $\binom{2z}{z} \geq \frac{2^{2z}}{2\sqrt{z}}$, we find the following lower bound on the sparsity of near-optimal dual solutions

$$k \geq \frac{\log\binom{2z}{z}}{\log m} \geq \frac{z}{\log m}.$$

If the dual solution needs to have at least $z/\log(m)$ non-zero entries, and the oracle outputs only a constant number of entries in each iteration, then the Arora-Kale framework with that oracle needs $T = \Omega(z/\log(m))$ iterations to build such a dual solution. Since $T = \mathcal{O}(R^2w^2\ln(n)/\varepsilon^2)$, this in turn implies that Rw/ε cannot be small.

Similar symmetry-based arguments can be made for more general classes of LPs and SDPs.

B. General width-bounds are restrictive for MAXCUT

One problem in using the Arora-Kale framework as a general SDP-solver, is that it is hard to give a good upper bound on the width w of the oracle. Here we used $w \leq r+1$ as an upper bound, where r is an upper bound on the smallest ℓ_1-norm r^* among all optimal solutions y to the dual. In general we cannot use a better upper bound on w: in our full version [19, Lemma 21] we show that for every $n \geq 4$, $m \geq 4$, $s \geq 1$, $R^* > 0$, $r^* > 0$, and $\varepsilon \leq 1/2$, there is an SDP with these parameters for which every oracle has width $w \geq \frac{1}{2}r^*$. This shows that every SDP-solver in the Arora-Kale framework which uses a general oracle that only considers those parameters (including our SDP-solver and the one of Brandão and Svore), will have a bad performance on every SDP with a large r^* parameter.

It turns out that r^* can grow linearly in n and m for many natural classes of SDPs. A simple example comes from the Goemans-Williamson SDP for approximating MAXCUT (see the introduction). Start with a graph $G^{(1)}$, and let $G^{(t)}$ be the graph corresponding to t disjoint copies of $G^{(1)}$. From the form of SDP (2), it is clear that the SDP corresponding to $G^{(t)}$ has the structure of the SDP corresponding to $G^{(1)}$, but copied t times. In particular this implies that the ℓ_1-norm of a dual solution for the $G^{(t)}$-SDP is $r^{*(t)} \geq tr^{*(1)}$. Since n and m are linear in t, we have $r^* = \Omega(n) = \Omega(m)$. This kind of argument can be generalized to other SDP formulations that have similar "combinable" structures.

IV. Lower bounds on quantum query complexity

In the full version of this paper [19, Section 4], we show that every LP-solver (and hence every SDP-solver) that can distinguish two optimal values with high probability needs $\Omega\left(\sqrt{\max\{n,m\}}\left(\min\{n,m\}\right)^{3/2}\right)$ quantum queries in the worst case. This general statement can be shown via reduction from the MAJ_a-OR_b-MAJ_c function, which is the composition of an a-bit majority function with a b-bit OR function with a c-bit majority function.

Due to space constraints, here we will only give a simpler proof for the special case $n = m$.

Theorem 14. *For every integer k, there exists an LP with $n = m = 2k$, such that calculating (with success probability $\geq 2/3$) the optimal value up to an additive error $\varepsilon = 1/3$ takes at least $\Omega(k^2) = \Omega(nm)$ queries to the input matrices.*

Proof: It is known that $\Omega(k^2)$ quantum queries are necessary to compute (with success probability $\geq 2/3$) the majority function on input $Z \in \{0,1\}^{k\times k}$ [25]. Computing this is equivalent to approximating the Hamming weight $|Z| = \sum_{i=1}^{k}\sum_{j=1}^{k} Z_{ij}$ within additive error $\varepsilon = 1/3$. We claim that the optimal value of the following LP equals $|Z|$:

$$\begin{aligned} \max \quad & \sum_{i=1}^{k} w_i \\ \text{s.t.} \quad & \begin{bmatrix} I_k & -Z \\ 0 & I_k \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} \leq \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix} \\ & 0 \leq v, w \end{aligned}$$

where I_k is the $k \times k$ identity matrix. This claim directly implies the theorem, since $n = m = 2k$.

To prove the claim, note that every w_i-variable only appears in one constraint of the form $w_i \leq \sum_{j=0}^{k} Z_{ij}v_j$. Since w_j is maximized, this constraint will be tight in the optimum. Since all entries of Z are non-negative, this implies that each v_j will be maximized. The only upper bound on the v_j-variables is $v_j \leq 1$, which will hence be tight in the optimum. Putting this together, we get

$$\text{OPT} = \sum_{i=1}^{k} w_i = \sum_{i=1}^{k}\sum_{j=1}^{k} Z_{ij}v_j = \sum_{i=1}^{k}\sum_{j=1}^{k} Z_{ij} = |Z|. \qquad \blacksquare$$

V. Conclusion

We gave better algorithms and lower bounds for quantum SDP-solvers, improving upon recent work of Brandão and Svore [1]. Here are a few directions for future work:

- **Better upper bounds.** The runtime of our algorithm, like the earlier algorithm of Brandão and Svore, has better dependence on m and n than the best classical SDP-solvers, but worse dependence on s and on Rr/ε. It may be possible to improve the dependence on s to linear and/or the dependence on Rr/ε to less than our current 8th power.
- **Applications of our algorithm.** As mentioned, both our and Brandão-Svore's quantum SDP-solvers only improve upon the best classical algorithms for a specific regime of parameters, namely where $mn \gg Rr/\varepsilon$. Unfortunately, we don't know particularly interesting problems in combinatorial optimization in this regime. As shown in Section III, many natural SDP formulations will not fall into this regime. However, it would be interesting to find useful SDPs for which our algorithm gives a significant speed-up.
- **New algorithms.** As in the work by Arora and Kale, it might be more promising to look at oracles (now quantum) that are designed for specific SDPs. Such oracles could build on the techniques developed here, or develop totally new techniques. It might also be possible to speed up other classical SDP-solvers, for example those based on interior-point methods.
- **Better lower bounds.** Our lower bounds are probably not optimal, particularly for the case where m and n are not of the same order. The most interesting case would be to get lower bounds that are simultaneously tight in the parameters m, n, s, and Rr/ε.

Acknowledgments: We thank Fernando Brandão for sending us several drafts of [1] and for answering our many questions about their algorithms, Stacey Jeffery for pointing us to [26], and Andris Ambainis and Robin Kothari for useful discussions and comments. We also thank the anonymous FOCS'17 referees for helpful comments that improved the presentation. JvA and SG are supported by the Netherlands Organization for Scientific Research grant number 617.001.351; AG and RdW are supported by ERC Consolidator Grant 615307-QPROGRESS.

References

[1] F. Brandão and K. Svore, "Quantum speed-ups for semidefinite programming," 2016, to appear in Proceedings of FOCS'17. arXiv:1609.05537v3.

[2] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

[3] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the ACM*, vol. 42, no. 6, pp. 1115–1145, 1995, earlier version in STOC'94.

[4] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell, "Optimal inapproximability results for MAX-CUT and other 2-variable CSPs?" *SIAM Journal on Computing*, vol. 37, no. 1, pp. 319–357, 2007, earlier version in FOCS'04.

[5] G. B. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," in *Activity Analysis of Production and Allocation*, ser. Cowles Commission Monograph No. 13. New York, N. Y.: John Wiley & Sons Inc., 1951, pp. 339–347.

[6] Y. T. Lee, A. Sidford, and S. C. Wong, "A faster cutting plane method and its implications for combinatorial and convex optimization," in *Proceedings of 56th IEEE FOCS*, 2015, pp. 1049–1065, arXiv:1508.04874.

[7] S. Arora and S. Kale, "A combinatorial, primal-dual approach to semidefinite programs," *Journal of the ACM*, vol. 63, no. 2, p. 12, 2016, earlier version in STOC'07.

[8] S. Arora, E. Hazan, and S. Kale, "The multiplicative weights update method: a meta-algorithm and applications," *Theory of Computing*, vol. 8, no. 6, pp. 121–164, 2012.

[9] K. Tsuda, G. Rätsch, and M. K. Warmuth, "Matrix exponentiated gradient updates for on-line learning and Bregman projection," *Journal of Machine Learning Research*, vol. 6, pp. 995–1018, 2005, earlier version in NIPS'04.

[10] M. K. Warmuth and D. Kuzmin, "Online variance minimization," *Machine Learning*, vol. 87, no. 1, pp. 1–32, 2012, earlier version in COLT'06.

[11] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997, earlier version in FOCS'94. quant-ph/9508027.

[12] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of 28th ACM STOC*, 1996, pp. 212–219, quant-ph/9605043.

[13] C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla, "Quantum query complexity of some graph problems," *SIAM Journal on Computing*, vol. 35, no. 6, pp. 1310–1328, 2006, earlier version in ICALP'04.

[14] A. Ambainis, "Quantum walk algorithm for element distinctness," *SIAM Journal on Computing*, vol. 37, no. 1, pp. 210–239, 2007, earlier version in FOCS'04. quant-ph/0311001.

[15] A. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for solving linear systems of equations," *Physical Review Letters*, vol. 103, no. 15, p. 150502, 2009, arXiv:0811.3171.

[16] D. Poulin and P. Wocjan, "Sampling from the thermal quantum Gibbs state and evaluating partition functions with a quantum computer," *Physical Review Letters*, vol. 103, p. 220502, 2009, arXiv:0905.2199.

[17] A. N. Chowdhury and R. D. Somma, "Quantum algorithms for Gibbs sampling and hitting-time estimation," 9 Mar 2016, arXiv:1603.02940.

[18] C. Dürr and P. Høyer, "A quantum algorithm for finding the minimum," 18 Jul 1996, quant-ph/9607014.

[19] J. van Apeldoorn, A. Gilyén, S. Gribling, and R. de Wolf, "Quantum SDP-solvers: Better upper and lower bounds," 2017, arxiv:1705.01843.

[20] A. M. Childs, R. Kothari, and R. D. Somma, "Quantum linear systems algorithm with exponentially improved dependence on precision," 7 November 2015, arxiv:1511.02306.

[21] A. M. Childs and N. Wiebe, "Hamiltonian simulation using linear combinations of unitary operations," *Quantum Information and Computation*, vol. 12, no. 11-12, pp. 901–924, 2012, arXiv:1202.5822.

[22] D. W. Berry, A. M. Childs, R. Cleve, R. Kothari, and R. D. Somma, "Simulating Hamiltonian dynamics with a truncated Taylor series," *Physical Review Letters*, vol. 114, p. 090502, 2015, arXiv:1412.4687.

[23] D. W. Berry, A. M. Childs, and R. Kothari, "Hamiltonian simulation with nearly optimal dependence on all parameters," *Proceedings of 56th IEEE FOCS*, pp. 792–809, 2015, arxiv:1501.01715.

[24] D. Poulin and P. Wocjan, "Preparing ground states of quantum many-body systems on a quantum computer," *Physical Review Letters*, vol. 102, p. 130503, 2009, arXiv:0809.2705.

[25] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf, "Quantum lower bounds by polynomials," *Journal of the ACM*, vol. 48, no. 4, pp. 778–797, 2001, earlier version in FOCS'98. quant-ph/9802049.

[26] S. Kimmel, "Quantum adversary (upper) bound," *Chicago Journal of Theoretical Computer Science*, 2013.