# Quantum Speed-ups for Semidefinite Programming

Fernando G.S.L. Brandão

Caltech

Krysta Svore

Microsoft Research

QIP 2017

# Quantum Algorithms

Exponential speed-ups:

Simulate quantum physics, factor big numbers (Shor's algorithm), ...,

Polynomial Speed-ups:

Searching (Grover's algorithm), ...

Heuristics:

Quantum annealing, adiabatic optimization, ...

# Quantum Algorithms

Exponential speed-ups:

Simulate quantum physics, factor big numbers (Shor's algorithm), ...,

Polynomial Speed-ups:

Searching (Grover's algorithm), ...

Heuristics:

Quantum annealing, adiabatic optimization, ...

This Talk:

Solving Semidefinite Programming belongs here

# Semidefinite Programming

… is an important class of convex optimization problems

$$\begin{aligned} & \max \operatorname{tr}(CX) \\ \forall j \in [m], \qquad & \operatorname{tr}(A_j X) \le b_j \\ & X \ge 0. \end{aligned}$$

Input: $n$ x $n$, $s$-sparse matrices $C$, $A_1$, ..., $A_m$ and numbers $b_1$, ..., $b_m$

Output: $X$

# Semidefinite Programming

… is an important class of convex optimization problems

$$\begin{aligned} & \max \operatorname{tr}(CX) \\ \forall j \in [m], \qquad & \operatorname{tr}(A_j X) \leq b_j \\ & X \succeq 0. \end{aligned}$$

Input: $n \times n$, $s$-sparse matrices $C, A_1, \ldots, A_m$ and numbers $b_1, \ldots, b_m$
Output: $X$

Linear Programming: special case
Many applications (combinatorial optimization, operational research, ....)
Natural in quantum (density matrices, ...)

# Semidefinite Programming

… is an important class of convex optimization problems

$$\begin{aligned} & \max \operatorname{tr}(CX) \\ \forall j \in [m], \qquad & \operatorname{tr}(A_j X) \leq b_j \\ & X \geq 0. \end{aligned}$$

Input: n x n, s-sparse matrices $C, A_1, \ldots, A_m$ and numbers $b_1, \ldots, b_m$

Output: X

Linear Programming: special case

Many applications (combinatorial optimization, operational research, ….)

Natural in quantum (density matrices, …)

Algorithms Interior points: $O((m^2 ns + mn^2)\log(1/\delta))$

Multiplicative Weights: $O((mns\,(\omega R)/\delta^2))$

width

size of solution

error

# Semidefinite Programming

... is an important class of convex optimization problems

$$\max \operatorname{tr}(CX)$$
$$\forall j \in [m], \quad \operatorname{tr}(A_j X) \le b_j$$

Are there quantum speed-ups for SDPs/LPs?

Natural question. But unexplored so far

Input: n ... $b_m$

Output: ...

Linear Pr...

Many ap... ....)

Natural in quan... (density matrices)

Algorithms Interior points: O((m²ns + mn²)log(1/δ))

Multiplicative Weights: O((mns (ωR)/δ²))

width

size of solution

error

# SDP Duality

**Primal:**

$$\forall j \in [m], \qquad \begin{aligned} &\max \operatorname{tr}(CX) \\ &\operatorname{tr}(A_j X) \leq b_j \\ &X \succeq 0. \end{aligned}$$

**Dual:**

$$\begin{aligned} &\min b.y \\ &\sum_{j=1}^{m} y_j A_j \succeq C \\ &y \geq 0. \end{aligned}$$

*y*: *m*-dimensional vector

Under mild conditions: $\text{Opt}_{\text{primal}} = \text{Opt}_{\text{dual}}$

# Size of Solutions

**Primal:**

$$\max \operatorname{tr}(CX)$$
$$\forall j \in [m], \qquad \operatorname{tr}(A_j X) \leq b_j$$
$$X \geq 0.$$

R parameter: $\operatorname{Tr}(X_{opt}) \leq R$

**Dual:**

$$\min b.y$$
$$\sum_{j=1}^{m} y_j A_j \geq C$$
$$y \geq 0.$$

r parameter: $\sum_i (y_{opt})_i \leq r$

# SDP Lower Bounds

Even to write down optimal solutions take time:

**Primal** ($n$ x $n$ PSD matrix $X$): $\Omega(n^2)$
**Dual** ($m$ dim vector $y$): $\Omega(m)$

# SDP Lower Bounds

Even to write down optimal solutions take time:

**Primal** (*n* x *n* PSD matrix *X*): $\Omega(n^2)$
**Dual** (*m* dim vector *y*): $\Omega(m)$

Even just to compute optimal value requires:

**Classical:** $\Omega(n+m)$ (for constant r, R, s, δ)
**Quantum:** $\Omega(n^{1/2} + m^{1/2})$ (for constant r, R, s, δ)

Easy reduction to search problem

(Apeldoorn, Gilyen, Gribling, de Wolf)
**Quantum:** $\Omega(nm)$ if $n \cong m$ (R, s, δ = O(1) but *not* r)
$\min(m, n) (\max(m, n))^{1/2}$ (R, s, δ = O(1) but *not* r)

**See poster this afternoon**

# Quantum Algorithm for SDP

**Result 1:** There is a quantum algorithm for solving SDPs running in time $\mathbf{n^{1/2}\, m^{1/2}\, s^2\, poly(log(n, m), R, r, \delta)}$

Input: $n$ x $n$, $s$-sparse matrices $C, A_1, \ldots, A_m$ and numbers $b_1, \ldots, b_m$

# Quantum Algorithm for SDP

**Result 1:** There is a quantum algorithm for solving SDPs running in time **$n^{1/2} m^{1/2} s^2$ poly(log(n, m), R, r, $\delta$)**

Input: $n \times n$, $s$-sparse matrices $C, A_1, \ldots, A_m$ and numbers $b_1, \ldots, b_m$

Normalization: $||A_i||, ||C|| \leq 1$

Output: Samples from $y/||y||_1$ and value $||y||_1$ and/or
Quantum Samples from $X/tr(X)$ and value $tr(X)$

Value $opt \pm \delta$

(output form similar to HHL Q. Algorithm for linear equations)

# Quantum Algorithm for SDP

**Result 1:** There is a quantum algorithm for solving SDPs running in time $n^{1/2} m^{1/2} s^2 \, \text{poly}(\log(n, m), R, r, \delta)$

Oracle Model: We assume there's an oracle that outputs a chosen non-zero entry of C, $A_1$, ..., $A_m$ at unit cost:

$$|j, k, l, z\rangle \rightarrow |j, k, l, z \oplus (A_j)_{k f_{jk}(l)}\rangle \qquad f_{jk} : [r] \rightarrow [n]$$

choice of $A_j$ row $k$ $l$ non-zero element

# Quantum Algorithm for SDP

**Result 1:** There is a quantum algorithm for solving SDPs running in time $\mathbf{n^{1/2}\, m^{1/2}\, s^2\, poly(\log(n, m), R, r, \delta)}$

**The good:**

Unconditional Quadratic speed-ups in terms of $n$ and $m$

Close to optimal: $\Omega(n^{1/2} + m^{1/2})$ lower bound

# Quantum Algorithm for SDP

**Result 1:** There is a quantum algorithm for solving SDPs running in time $\mathbf{n^{1/2}\, m^{1/2}\, s^2\, poly(log(n, m), R, r, \delta)}$

**The good:**

Unconditional Quadratic speed-ups in terms of $n$ and $m$

Close to optimal: $\Omega(n^{1/2} + m^{1/2})$ q. lower bound

**The bad:**

Terrible dependence on other parameters:

$\mathrm{poly}(\log(n, m), R, r, \delta) \leq \mathbf{(Rr)^{32}\, \delta^{-18}}$

Close to optimal: no general super-polynomial speed-ups

# Quantum Algorithm for SDP

**Result 1:** There is a quantum algorithm for solving SDPs running in time **$n^{1/2} m^{1/2} s^2$ poly(log(n, m), R, r, δ)**

**Special case:**

If the SDP is s.t. $b_i \geq 1$ for all *i*,
there is no dependence on r (size of dual solution)

# Larger Speed-ups?

**Result 2:** There is a quantum algorithm for solving SDPs running in time $\mathbf{T_{Gibbs}\, m^{1/2} poly(\log(n, m), s, R, r, \delta)}$

# Larger Speed-ups?

**Result 2:** There is a quantum algorithm for solving SDPs running in time **$T_{Gibbs}\, m^{1/2}\mathrm{poly}(\log(n, m), s, R, r, \delta)$**

**$T_{Gibbs}$** := Time to prepare on quantum computer Gibbs states of the form

$$\exp\left(\sum_{i=1}^{m} \nu_i A_i + \nu_0 C\right) / \mathrm{tr}(\ldots)$$

for real numbers $|\nu_i| \leq O(\log(n), \mathrm{poly}(1/\delta))$

# Larger Speed-ups?

**Result 2:** There is a quantum algorithm for solving SDPs running in time $T_{Gibbs}\, m^{1/2} \mathrm{poly}(\log(n, m), s, R, r, \delta)$

$T_{Gibbs}$ := Time to prepare on quantum computer Gibbs states of the form

$$\exp\left(\sum_{i=1}^{m} \nu_i A_i + \nu_0 C\right) / \mathrm{tr}(\ldots)$$

for real numbers $|v_i| \leq O(\log(n), \mathrm{poly}(1/\delta))$

Can use **Quantum Gibbs Sampling** (e.g. Quantum Metropolis) as heuristic. Exponential Speed-up if thermalization is quick (poly #qubits = polylog(n))

Gives application of quantum Gibbs sampling outside simulating physical systems

# Larger Speed-ups with “quantum data”

**Result 3:** There is a quantum algorithm for solving SDPs running in time **$m^{1/2}$poly(log(n, m), s, R, r, δ, rank)** with data in quantum form

Quantum Oracle Model: There is an oracle that given *i*, outputs the eigenvalues of $A_i$ and its eigenvectors as quantum states

rank := max ($\max_i$ rank($A_i$), rank(C))

# Larger Speed-ups with "quantum data"

**Result 3:** There is a quantum algorithm for solving SDPs running in time **$m^{1/2}$poly(log(n, m), s, R, r, δ, rank)** with data in quantum form

Quantum Oracle Model: There is an oracle that given *i,* outputs the eigenvalues of $A_i$ and its eigenvectors as quantum states

rank := max (max$_i$ rank($A_i$), rank(C))

Idea: in this case one can easily perform the Gibbs sampling in poly(log(n), rank) time

Limitation: Not clear the relevance of the model.
How to compare with classical methods in a meaningful way?

# Special Case: Max Eigenvalue

Computing the max eigenvalue of C is a SDP

$\max \mathrm{tr}(CX)$: $\mathrm{tr}(X) = 1$, $X \geq 0$

# Special Case: Max Eigenvalue

Computing the max eigenvalue of C is a SDP

$$\max \operatorname{tr}(CX): \quad \operatorname{tr}(X) = 1, \quad X \geq 0$$

This is a well studied problem:

**Quantum Annealing** (cool down -C):

If we can prepare $e^{\beta C}/\operatorname{tr}(e^{\beta C})$ for $\beta = O(\log(n)/\delta)$ can compute max eigenvalue to error $\delta$

# Special Case: Max Eigenvalue

(Poulin, Wocjan '09) Can prepare $e^{\beta C}/\mathrm{tr}(e^{\beta C})$ for s-sparse C in time Õ(s $n^{1/2}$) on quantum computer

**Idea:** Phase estimation + Amplitude amplification

$$C|\psi_i\rangle = E_i|\psi_i\rangle$$

$$\sum_i |\psi_i\rangle|\psi_i^*\rangle \to \sum_i |\psi_i\rangle|\psi_i^*\rangle|E_i\rangle \to \sum_i e^{-E_i/2}|\psi_i\rangle|\psi_i^*\rangle|E_i\rangle|0\rangle + \ldots$$

phase estimation

Post-selecting on "0" gives a purification of Gibbs state with Pr > O(1/n)

Using amplitude amplification can boost Pr > 1-o(1) with O($n^{1/2}$) iterations

# General Case: Quantizing Arora-Kale Algorithm

The quantum algorithm is based on a classical algorithm for SDP due to Arora and Kale (2007) based on the multiplicative weight method. Let's review their method

Assumptions:

We assume $b_i \geq 1$.
Can reduce general case to this with blow up of poly(r) in complexity

We also assume w.l.o.g. $A_1 = I, b_1 = R$

# The Oracle

The Arora-Kale algorithm has an auxiliary algorithm (the ORACLE) which solves a simple linear programming:

$\mathsf{ORACLE}(\rho)$

Searches for a vector *y* s.t.

i) $y \in D_\alpha := \{y : y \geq 0,\ b.y \leq \alpha\}$

ii) $\sum_{j=1}^{m} \mathrm{tr}(A_j \rho) y_j - \mathrm{tr}(C\rho) \geq 0$

# Arora-Kale Algorithm

$\rho^1 = I/n,\ \varepsilon = \dfrac{\delta}{2R},\ \varepsilon' = -\ln(1-\varepsilon),\ T = \dfrac{8R^2 \ln(n)}{\delta^2}$

For $t = 1, \ldots, T$

1. $y^t \leftarrow \text{ORACLE}(\rho^t)$

2. $M^t = \left( \sum_{j=1}^{m} y_j^t A_j - C + RI \right) /2R$

3. $W^{t+1} = \exp\left( -\varepsilon' \left( \sum_{\tau=1}^{t} M^\tau \right) \right)$

4. $\rho^{t+1} = W^{t+1}/\mathrm{tr}(W^{t+1})$

Output: $\overline{y} = \frac{\delta\alpha}{R} e_1 + \frac{1}{T} \sum_{t=1}^{T} y^t$ $\qquad e_1 = (1, 0, \ldots, 0)$

# Arora-Kale Algorithm

$\rho^1 = I/n,\ \varepsilon = \dfrac{\delta}{2R},\ \varepsilon' = -\ln(1-\varepsilon),$ $= \dfrac{8R^2 \ln(n)}{\delta^2}$

For $t = 1, \ldots, T$

1. $y^t \leftarrow \mathrm{ORACLE}(\rho^t)$

2.

3.

4. $\rho^{t+1} = W^{t+1}/\mathrm{tr}(W^{t+1})$

Output: $\overline{y} = \frac{\delta\alpha}{R} e_1 + \frac{1}{T}\sum_{t=1}^{T} y^t$ $\qquad e_1 = (1, 0, \ldots, 0)$

**Thm** (Arora-Kale '07) ȳ.b ≤ (1+δ) α

Can find optimal value by binary search

# Why Arora-Kale works?

Since $y_t \in D_\alpha := \{y : y \geq 0, b.y \leq \alpha\}$

$$\overline{y}.b \leq \frac{\delta\alpha}{R} b_1 + \frac{1}{T}\sum_{t=1}^{T} y^t.b \leq (1+\delta)\alpha$$

---

Must check $\overline{y}$ is feasible

From Oracle, for all *t*: $\operatorname{tr}\left(\left(\sum_{j=1}^{m} y_j^t A_j - C\right)\rho^t\right) \geq 0$

We need: $\lambda_{\min}\left(\left(\sum_{j=1}^{m}\left(\frac{1}{T}\sum_{t=1}^{T} y_j^t\right)A_j - C\right)\right) \geq 0$

# Matrix Multiplicative Weight

MMW (Arora, Kale '07) Given n x n matrices $0 < M^t < I$ and $\varepsilon < 1/2$,

$$\frac{1}{T}\sum_{t=1}^{T}\mathrm{tr}(M^t\rho^t) \leq \left(\frac{1+\varepsilon}{T}\right)\lambda_n\left(\sum_{t=1}^{T}M^t\right) + \frac{\ln(n)}{T\varepsilon}$$

with $\rho^t = \frac{\exp(-\varepsilon'(\sum_{\tau=1}^{t-1}M^\tau))}{\mathrm{tr}(\ldots)}$ and $\varepsilon' = -\ln(1-\varepsilon)$

$\lambda_n$ : min eigenvalue

2-player zero-sum game interpretation:

- Player A chooses density matrix $X^t$
- Player B chooses matrix $0 < M^t < I$

Pay-off: $\mathrm{tr}(X^t M^t)$

"$X^t = \rho^t$ strategy almost as good as global strategy"

# Matrix Multiplicative Weight

MMW (Arora, Kale '07) Given n x n matrices $M^t$ and $\varepsilon < ½$,

$$\frac{1}{T}\sum_{t=1}^{T}\mathrm{tr}(M^t\rho^t) \leq \left(\frac{1+\varepsilon}{T}\right)\lambda_n\left(\sum_{t=1}^{T}M^t\right) + \frac{\ln(n)}{T\varepsilon}$$

with $\rho^t = \dfrac{\exp(-\varepsilon'(\sum_{\tau=1}^{t-1} M^\tau))}{\mathrm{tr}(\ldots)}$ and $\varepsilon' = -\ln(1-\varepsilon)$

$\lambda_n$ : min eigenvalue

From Oracle: $\mathrm{tr}\left(\left(\sum_{j=1}^{m} y_j^t A_j - C\right)\rho^t\right) \geq 0$

By MMW: $\lambda_{\min}\left(\left(\sum_{j=1}^{m}\left(\frac{1}{T}\sum_{t=1}^{T} y_j^t\right)A_j - C\right)\right) \geq 0$

# Quantizing Arora-Kale Algorithm

We make it quantum as follows:

1. Implement ORACLE by Gibbs Sampling to produce $y^t$ and apply amplitude amplification to solve it in time $\tilde{O}(s^2\, n^{1/2}\, m^{1/2})$

2. Sparsify $M^t$ to be a sum of $O(\log(m))$ terms:

$$\overline{M}^t = \left( \|y^t\|_1 Q^{-1} \sum_{j=1}^{Q} A_{i_j} - C + RI \right) / 2R \qquad \overline{M}^t \approx M^t$$

$$(i_1, \ldots, i_Q) \sim y^t / \|y^t\|_1, \quad Q = O(\log(m))$$

3. Quantum Gibbs Sampling + amplitude amplification to prepare

$$\overline{\rho}^t = \exp\left( -\varepsilon' \sum_{\tau=1}^{t} \overline{M}^\tau \right) / \mathrm{tr}(\ldots) \qquad \overline{\rho}^t \approx \rho^t$$

in time $\tilde{O}(s^2\, n^{1/2})$.

# Quantizing Arora-Kale Algorithm

We make it quantum as follows:

1. Implement ORACLE by Gibbs Sampling to produce $y^t$ and apply amplitude amplification to solve it in time $\tilde{O}(s^2\, n^{1/2}\, m^{1/2})$

We'll show there is a feasible $y^t$ of the form $y^t = Nq^t$ with $q^t := \exp(h)/\mathrm{tr}(\exp(h))$ and

$$h = \sum_{i=1}^{m} \left( \lambda \mathrm{tr}(A_i \rho^t) + \mu b_i \right) |i\rangle\langle i|$$

We need to simulate an oracle to the entries of $h$. We do it by measuring $\rho^t$ with $A_i$.

To prepare each $\rho^t$ takes time $\tilde{O}(s^2\, n^{1/2})$. To sample from $q^t$ requires $\tilde{O}(m^{1/2})$ calls to oracle for $h$. So total time is $\tilde{O}(s^2\, n^{1/2}\, m^{1/2})$

# Quantizing Arora-Kale Algorithm

We make it quantum as follows:

1. Implement ORACLE by Gibbs Sampling to produce $y^t$ and apply amplitude amplification to solve it in time $\tilde{O}(s^2 n^{1/2} m^{1/2})$
2. Sparsify $M^t$ to be a sum of $O(\log(m))$ terms:

$$\overline{M}^t = \left( \|y^t\|_1 Q^{-1} \sum_{j=1}^{Q} A_{i_j} - C + RI \right) / 2R \qquad \overline{M}^t \approx M^t$$

$$(i_1, \ldots, i_Q) \sim y^t / \|y^t\|_1, \quad Q = O(\log(m))$$

Can show it works by Matrix Hoeffding bound: $Z_1, \ldots, Z_k$ independent n x n Hermitian matrices s.t. $E(Z_i)=0$, $||Z_i||<\lambda$. Then

$$\Pr\left( \left\| \frac{1}{k} \sum_{i=1}^{k} Z_i \right\| \geq \varepsilon \right) \leq n . \exp\left( -\frac{k\varepsilon^2}{8\lambda^2} \right)$$

# Quantum Arora-Kale, Roughly

Let $\rho^1 = I/n$, $\varepsilon = \frac{\delta\alpha}{2\omega R}$, $\varepsilon' = -\ln(1-\varepsilon)$, $T = \frac{8\omega^2 R^2 \ln(n)}{\delta^2 \alpha^2}$.

For $t = 1, \ldots, T$

1. $y^t \leftarrow \mathsf{ORACLE}(\rho^t)$ Gibbs Sampling
2. $M^t = \sum_{j=1}^{m} (y_j^t A_j - C + \omega I)/2\omega$
3. Sparsify $M^t$ to $(M')^t$
4. $\rho^{t+1} = \exp\left(-\varepsilon'\left(\sum_{\tau=1}^{t} (M')^\tau\right)\right)/\mathrm{tr}(\ldots)$ Gibbs Sampling

Output: $\overline{y} = \frac{\delta\alpha}{R} e_1 + \frac{1}{T}\sum_{t=1}^{T} y^t$

# Implementing Oracle by Gibbs Sampling

$\mathsf{ORACLE}(\rho)$

Searches for a vector *y* s.t.

i) $y \in D_\alpha := \{y : y \geq 0,\ b.y \leq \alpha\}$

ii) $\sum_{j=1}^{m} \mathrm{tr}(A_j \rho) y_j - \mathrm{tr}(C\rho) \geq 0$

# Implementing Oracle by Gibbs Sampling

Searches for (non-normalized) probability distribution *y* satisfying two linear constraints:

$$\mathrm{tr}(BY) \leq \alpha, \quad \mathrm{tr}(AY) \geq \mathrm{tr}(C\rho)$$

$$Y = \sum_i y_i |i\rangle\langle i|, B = \sum_i b_i |i\rangle\langle i|, A = \sum_i \mathrm{tr}(A_i \rho)|i\rangle\langle i|$$

**Claim:** We can take Y to be Gibbs: There are constants *N, λ,* μ s.t.

$$Y = N \frac{\exp(\lambda A + \mu B)}{\mathrm{tr}(\ldots)}$$

# Jaynes' Principle

(Jaynes 57) Let ρ be a quantum state s.t. $\mathrm{tr}(\rho M_i) = c_i$

Then there is a Gibbs state of the form $\exp\left(\sum_i \lambda_i M_i\right)/\mathrm{tr}(\ldots)$

with same expectation values.

Drawback: no control over size of the $\lambda_i$'s.

# Finitary Jaynes' Principle

(Lee, Raghavendra, Steurer '15) Let ρ s.t. $\mathrm{tr}(\rho M_i) = c_i$

Then there is a $\sigma := \dfrac{\exp\left(\sum_i \lambda_i M_i\right)}{\mathrm{tr}(\ldots)}$

with $|\lambda_i| \leq 2\ln(\dim(\rho))/\varepsilon$

s.t. $|\mathrm{tr}(M_i\sigma) - c_i| \leq \varepsilon$

(Note: Used to prove limitations of SDPs for approximating constraints satisfaction problems; **see James Lee's talk**)

# Implementing Oracle by Gibbs Sampling

**Claim** There is a Y of the form $Y = N\frac{\exp(\lambda A + \mu B)}{\mathrm{tr}(\ldots)}$

with λ, μ < log(n)/ε and N < α s.t.

$$\mathrm{tr}(BY) \leq \alpha + N\varepsilon, \quad \mathrm{tr}(AY) \geq \mathrm{tr}(C\rho) - N\varepsilon$$

$$Y = \sum_i y_i |i\rangle\langle i|, B = \sum_i b_i |i\rangle\langle i|, A = \sum_i \mathrm{tr}(A_i\rho)|i\rangle\langle i|$$

# Implementing Oracle by Gibbs Sampling

**Claim** There is a Y of the form $Y = N \frac{\exp(\lambda A + \mu B)}{\mathrm{tr}(\ldots)}$

with $\lambda$, $\mu$ < $\log(n)/\varepsilon$ and $N < \alpha$ s.t.

$$\mathrm{tr}(BY) \le \alpha + N\varepsilon, \;\; \mathrm{tr}(AY) \ge \mathrm{tr}(C\rho) - N\varepsilon$$

Can implement oracle by exhaustive searching over x, y, N for a Gibbs distribution satisfying constraints above

(only $\alpha \log^2(n)/\varepsilon^3$ different triples needed to be checked)

# Conclusion and Open Problems

Quantum computers provide speed-up for SDPs

Many **open questions:**

- Can we improve the parameters (in terms of R, r, δ)?
- Can we find optimal algorithm in terms of n, m and s?
- Can we find relevant settings with superpoly speed-ups?
- Robustness to error?
- Q. computer only used for Gibbs Sampling. Application of small-sized q. computer?

# Conclusion and Open Problems

Quantum computers provide speed-up for SDPs

Many **open questions:**

- Can we improve the parameters (in terms of R, r, $\delta$)?
- Can we find optimal algorithm in terms of n, m and s?
- Can we find relevant settings with superpoly speed-ups?
- Robustness to error?
- Q. computer only used for Gibbs Sampling. Application of small-sized q. computer?

Thanks!