Quantum speedup of Monte Carlo methods

Ashley Montanaro

Department of Computer Science and School of Mathematics,
University of Bristol

27 August 2015

arXiv:1504.06987

Proc. R. Soc. A 2015 471 20150301

University of BRISTOL

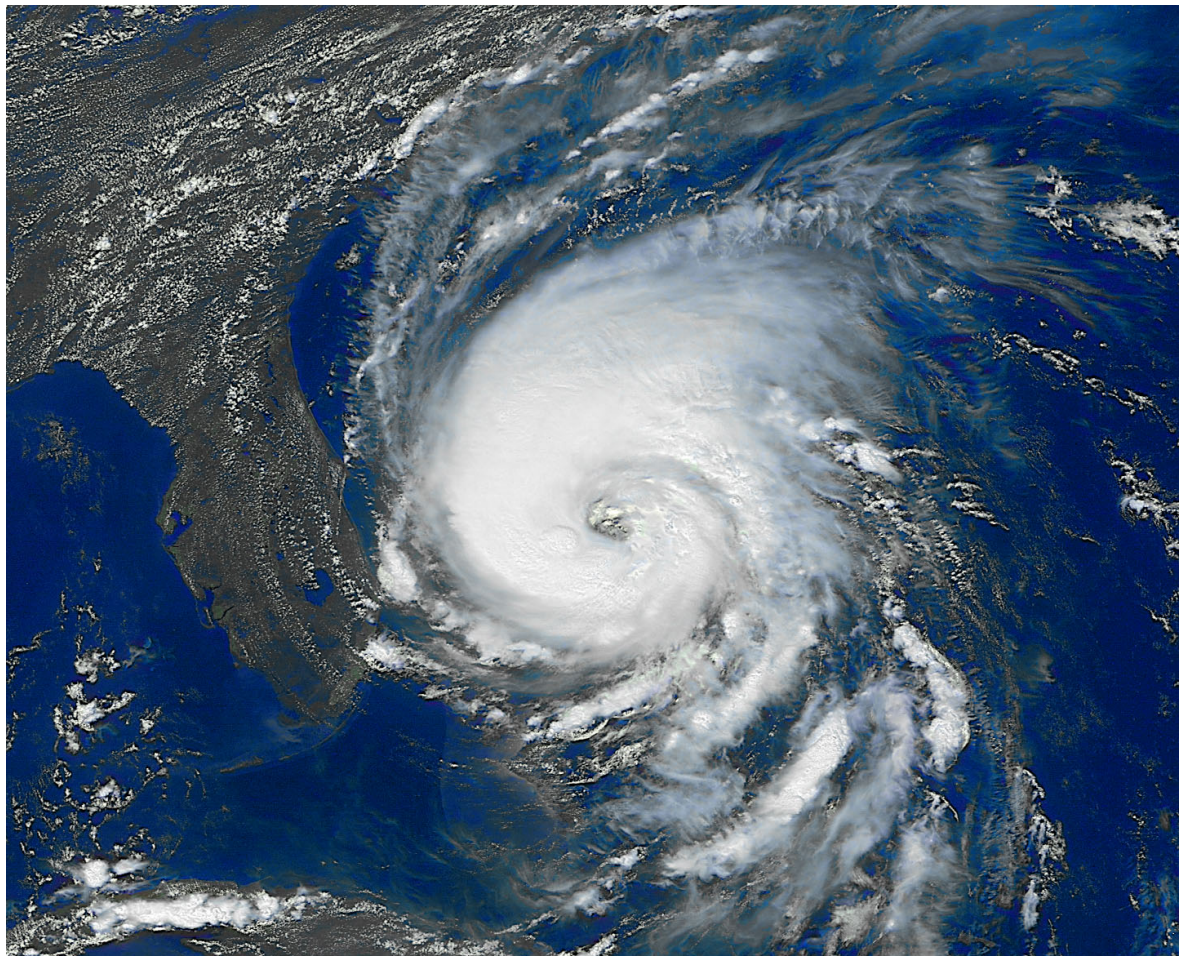
EPSRC
Engineering and Physical Sciences Research Council

Monte Carlo methods

Monte Carlo methods use randomness to estimate numerical properties of systems which are too large or complicated to analyse deterministically.

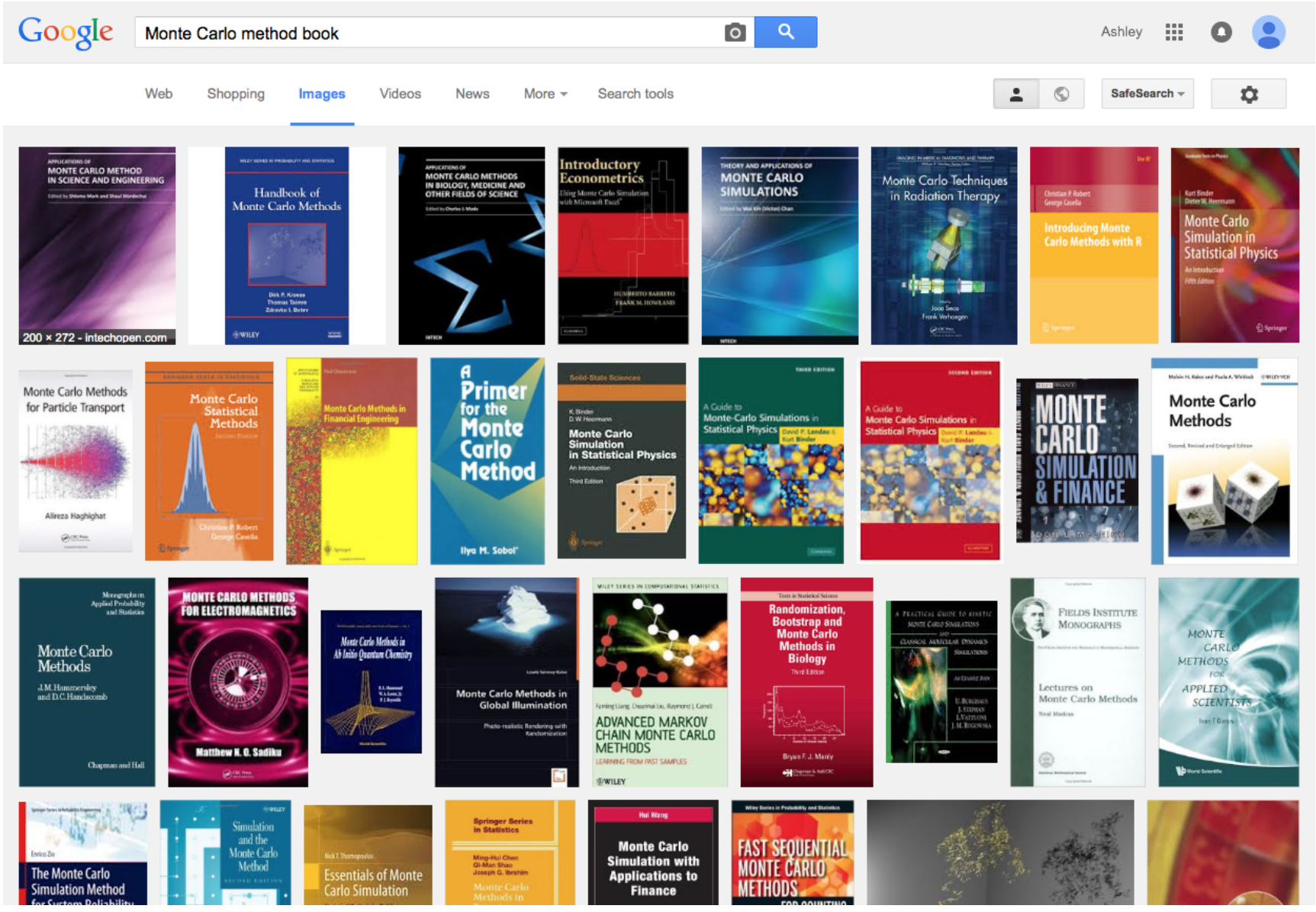
Monte Carlo methods

Monte Carlo methods use randomness to estimate numerical properties of systems which are too large or complicated to analyse deterministically.

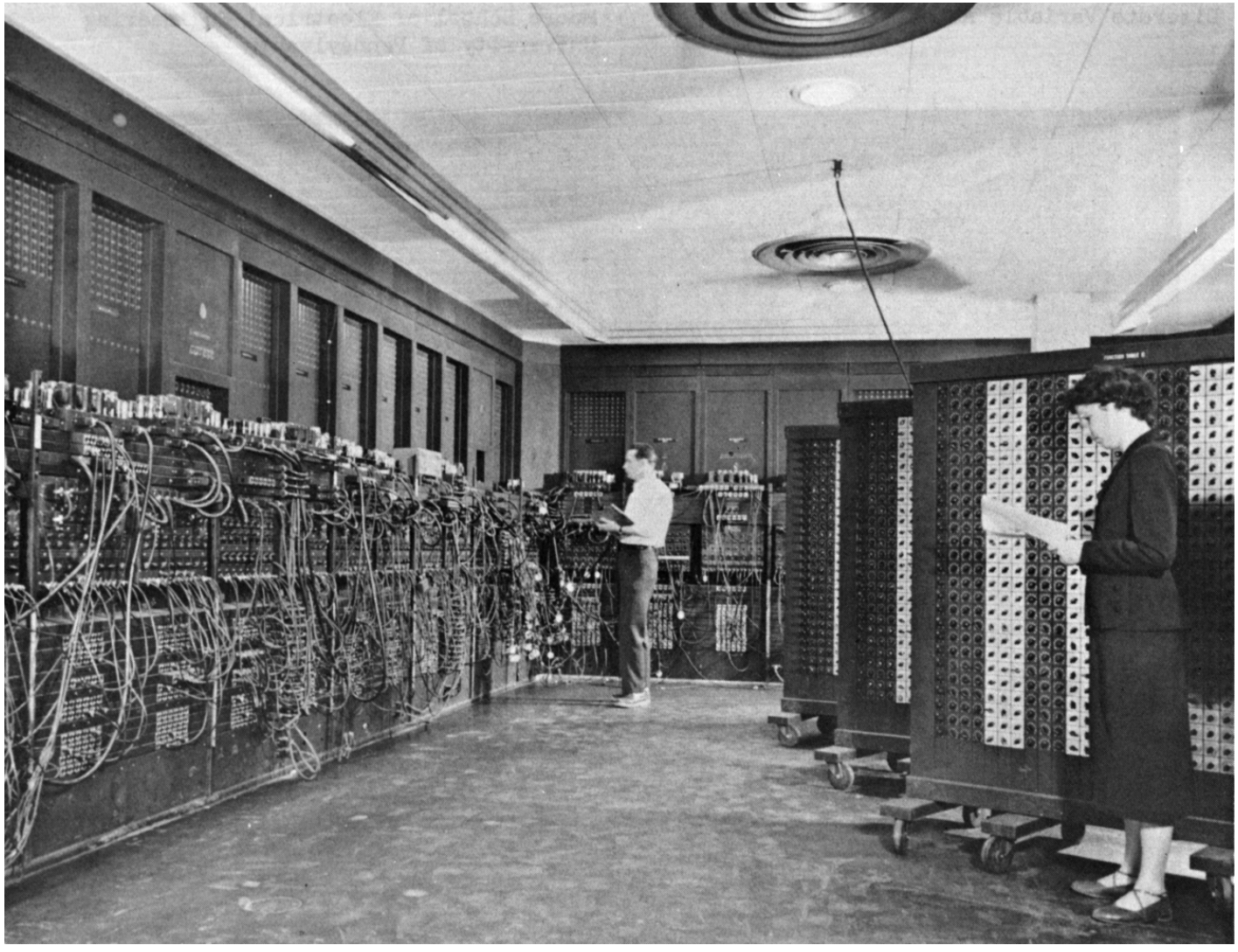


Pic: Wikipedia

These methods are used throughout science and engineering:



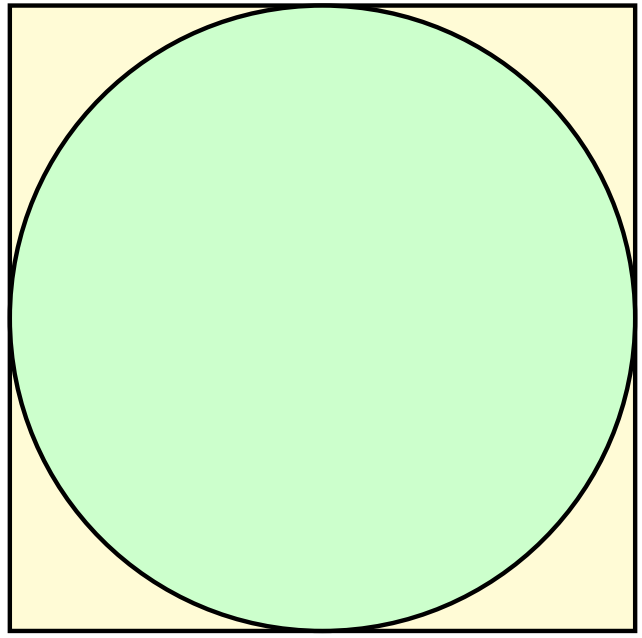
. . . and were an application of the first electronic computers:



Pic: Wikipedia

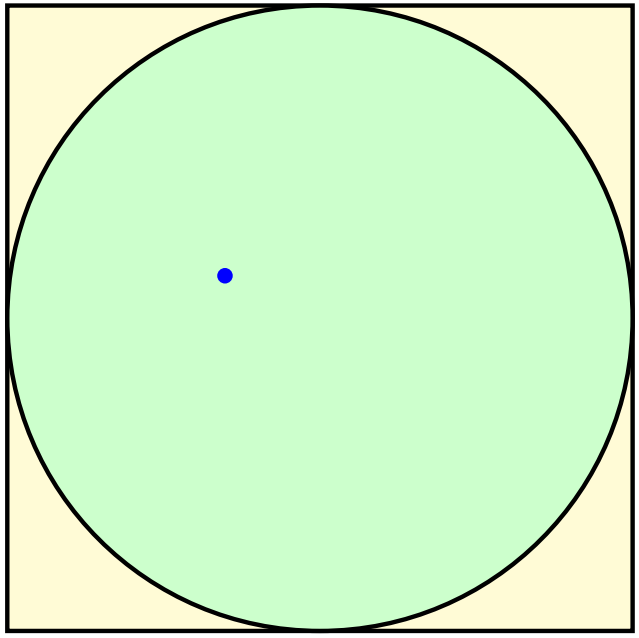
Monte Carlo methods

One very simple example of a Monte Carlo method: approximate π by throwing darts at a dartboard (choosing random points within a square).



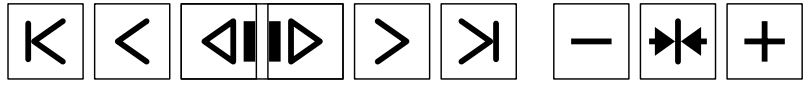
$\Pr[\text{point lands in circle}] = \frac{\pi}{4}$, so $\pi = 4 \cdot \Pr[\text{point lands in circle}]$.

A randomised algorithm for approximating π



Points landed in circle: 1/1.

Approximation to π: 4.0.



$\Pr[\text{point lands in circle}] = \frac{\pi}{4}$, so $\pi = 4 \cdot \Pr[\text{point lands in circle}]$.

This talk

Today I will discuss a quantum algorithm to speed up Monte Carlo methods in a quite general setting.

And also some applications of the algorithm:

1. Partition function problems in statistical physics
2. Approximate counting problems in combinatorics
3. Approximating the distance between probability distributions

Monte Carlo methods

The basic core of many Monte Carlo methods is:

General problem

Given access to a randomised algorithm $\mathcal{A}$, estimate the expected output value μ of $\mathcal{A}$.

Monte Carlo methods

The basic core of many Monte Carlo methods is:

General problem

Given access to a randomised algorithm $\mathcal{A}$, estimate the expected output value μ of $\mathcal{A}$.

- The input is fixed, and the expectation is taken over the internal randomness of $\mathcal{A}$.
- The output value $v(\mathcal{A})$ is a real-valued random variable.

Monte Carlo methods

The basic core of many Monte Carlo methods is:

General problem

Given access to a randomised algorithm $\mathcal{A}$, estimate the expected output value μ of $\mathcal{A}$.

- The input is fixed, and the expectation is taken over the internal randomness of $\mathcal{A}$.
- The output value $v(\mathcal{A})$ is a real-valued random variable.

We assume that we know an upper bound on the variance of this random variable:

$$\mathsf{Var}(v(\mathcal{A})) \leqslant \sigma^2.$$

Classical algorithm

The following natural algorithm solves this problem for any $\mathcal{A}$:

1. Produce k samples $v_1, \ldots, v_k$, each corresponding to the output of an independent execution of $\mathcal{A}$.
2. Output the average $\widetilde{\mu} = \frac{1}{k} \sum_{i=1}^{k} v_i$ of the samples as an approximation of μ.

Classical algorithm

The following natural algorithm solves this problem for any $\mathcal{A}$:

1. Produce k samples $v_1, \ldots, v_k$, each corresponding to the output of an independent execution of $\mathcal{A}$.
2. Output the average $\widetilde{\mu} = \frac{1}{k} \sum_{i=1}^{k} v_i$ of the samples as an approximation of μ.

Assuming that the variance of $v(\mathcal{A})$ is at most σ^2,

$$\Pr[|\widetilde{\mu} - \mu| \geqslant \epsilon] \leqslant \frac{\sigma^2}{k\epsilon^2}.$$

Classical algorithm

The following natural algorithm solves this problem for any $\mathcal{A}$:

1. Produce k samples $v_1, \ldots, v_k$, each corresponding to the output of an independent execution of $\mathcal{A}$.
2. Output the average $\widetilde{\mu} = \frac{1}{k} \sum_{i=1}^{k} v_i$ of the samples as an approximation of μ.

Assuming that the variance of $v(\mathcal{A})$ is at most σ^2,

$$\Pr[|\widetilde{\mu} - \mu| \geqslant \epsilon] \leqslant \frac{\sigma^2}{k\epsilon^2}.$$

So we can take $k = O(\sigma^2/\epsilon^2)$ to estimate μ up to additive error ϵ with, say, 99% success probability.

Classical algorithm

The following natural algorithm solves this problem for any $\mathcal{A}$:

1. Produce k samples $v_1, \ldots, v_k$, each corresponding to the output of an independent execution of $\mathcal{A}$.
2. Output the average $\widetilde{\mu} = \frac{1}{k} \sum_{i=1}^{k} v_i$ of the samples as an approximation of μ.

Assuming that the variance of $v(\mathcal{A})$ is at most σ^2,

$$\Pr[|\widetilde{\mu} - \mu| \geqslant \epsilon] \leqslant \frac{\sigma^2}{k\epsilon^2}.$$

So we can take $k = O(\sigma^2/\epsilon^2)$ to estimate μ up to additive error ϵ with, say, 99% success probability.

This scaling is optimal for classical algorithms [Dagum et al. '00].

Classical algorithm

The following natural algorithm solves this problem for any $\mathcal{A}$:

1. Produce k samples $v_1, \ldots, v_k$, each corresponding to the output of an independent execution of $\mathcal{A}$.
2. Output the average $\widetilde{\mu} = \frac{1}{k} \sum_{i=1}^{k} v_i$ of the samples as an approximation of μ.

Assuming that the variance of $v(\mathcal{A})$ is at most σ^2,

$$\Pr[|\widetilde{\mu} - \mu| \geqslant \epsilon] \leqslant \frac{\sigma^2}{k\epsilon^2}.$$

So we can take $k = O(\sigma^2/\epsilon^2)$ to estimate μ up to additive error ϵ with, say, 99% success probability.

This scaling is optimal for classical algorithms [Dagum et al. '00].

- To estimate π up to 4 decimal places with success probability 0.5, we would need $> 10^9$ trials!

Quantum speedup

With a quantum computer, we can do better:

Theorem [AM '15]

There is a quantum algorithm which estimates μ up to additive error ϵ with 99% success probability and

$$\widetilde{O}(\sigma/\epsilon)$$

uses of $\mathcal{A}$ (and $\mathcal{A}^{-1}$).

Quantum speedup

With a quantum computer, we can do better:

Theorem [AM '15]

There is a quantum algorithm which estimates μ up to additive error ϵ with 99% success probability and

$$\widetilde{O}(\sigma/\epsilon)$$

uses of $\mathcal{A}$ (and $\mathcal{A}^{-1}$).

- The $\widetilde{O}$ notation hides polylog factors: more precisely, the complexity is $O((\sigma/\epsilon)\log^{3/2}(\sigma/\epsilon)\log\log(\sigma/\epsilon))$.

Quantum speedup

With a quantum computer, we can do better:

Theorem [AM '15]

There is a quantum algorithm which estimates μ up to additive error ϵ with 99% success probability and

$$\widetilde{O}(\sigma/\epsilon)$$

uses of $\mathcal{A}$ (and $\mathcal{A}^{-1}$).

- The $\widetilde{O}$ notation hides polylog factors: more precisely, the complexity is $O((\sigma/\epsilon)\log^{3/2}(\sigma/\epsilon)\log\log(\sigma/\epsilon))$.
- This complexity is optimal up to these polylog factors [Nayak and Wu '98].

Quantum speedup

With a quantum computer, we can do better:

Theorem [AM '15]

There is a quantum algorithm which estimates μ up to additive error ϵ with 99% success probability and

$$\widetilde{O}(\sigma/\epsilon)$$

uses of $\mathcal{A}$ (and $\mathcal{A}^{-1}$).

- The $\widetilde{O}$ notation hides polylog factors: more precisely, the complexity is $O((\sigma/\epsilon)\log^{3/2}(\sigma/\epsilon)\log\log(\sigma/\epsilon))$.
- This complexity is optimal up to these polylog factors [Nayak and Wu '98].

The underlying algorithm $\mathcal{A}$ can now be quantum itself.

Related work

This problem connects to several previous works, e.g.:

- Approximating the mean of an arbitrary bounded function (with range $[0, 1]$), with respect to the uniform distribution. Quantum complexity: $O(1/\epsilon)$ [Heinrich '01], [Brassard et al. '11].

Related work

This problem connects to several previous works, e.g.:

- Approximating the mean of an arbitrary bounded function (with range $[0, 1]$), with respect to the uniform distribution. Quantum complexity: $O(1/\epsilon)$ [Heinrich '01], [Brassard et al. '11].
- Estimating the expected value $\mathrm{tr}(A\rho)$ of certain observables A which are bounded [Wocjan et al. '09], or whose tails decay quickly [Knill et al. '07].

Related work

This problem connects to several previous works, e.g.:

- Approximating the mean of an arbitrary bounded function (with range $[0, 1]$), with respect to the uniform distribution. Quantum complexity: $O(1/\epsilon)$ [Heinrich '01], [Brassard et al. '11].
- Estimating the expected value $\mathrm{tr}(A\rho)$ of certain observables A which are bounded [Wocjan et al. '09], or whose tails decay quickly [Knill et al. '07].
- Approximating the mean, with respect to the uniform distribution, of functions with bounded L^2 norm [Heinrich '01]

Related work

This problem connects to several previous works, e.g.:

- Approximating the mean of an arbitrary bounded function (with range $[0, 1]$), with respect to the uniform distribution. Quantum complexity: $O(1/\epsilon)$ [Heinrich '01], [Brassard et al. '11].
- Estimating the expected value $\mathrm{tr}(A\rho)$ of certain observables A which are bounded [Wocjan et al. '09], or whose tails decay quickly [Knill et al. '07].
- Approximating the mean, with respect to the uniform distribution, of functions with bounded L^2 norm [Heinrich '01]

Here we generalise these by approximating the mean output value of arbitrary quantum algorithms, given only a bound on the variance.

Ideas behind the algorithm

The algorithm combines and extends ideas of [Heinrich '01], [Brassard et al. '11], [Wocjan et al. '09].

Ideas behind the algorithm

The algorithm combines and extends ideas of [Heinrich '01], [Brassard et al. '11], [Wocjan et al. '09].

First, in the special case where $v(\mathcal{A}) \in [0, 1]$:

- Assume $\mathcal{A}$ is a quantum algorithm which finishes with a computational basis measurement, and then associates each outcome x with output $\phi(x) \in [0, 1]$.

Ideas behind the algorithm

The algorithm combines and extends ideas of [Heinrich '01], [Brassard et al. '11], [Wocjan et al. '09].

First, in the special case where $v(\mathcal{A}) \in [0,1]$:

- Assume $\mathcal{A}$ is a quantum algorithm which finishes with a computational basis measurement, and then associates each outcome x with output $\phi(x) \in [0,1]$.
- Then we replace the end of $\mathcal{A}$ with the map

$$|x\rangle|0\rangle \mapsto |x\rangle(\sqrt{1-\phi(x)}|0\rangle + \sqrt{\phi(x)}|1\rangle).$$

- Now the probability of measuring 1 on the last qubit is precisely μ.

Ideas behind the algorithm

The algorithm combines and extends ideas of [Heinrich '01], [Brassard et al. '11], [Wocjan et al. '09].

First, in the special case where $v(\mathcal{A}) \in [0,1]$:

- Assume $\mathcal{A}$ is a quantum algorithm which finishes with a computational basis measurement, and then associates each outcome x with output $\phi(x) \in [0,1]$.
- Then we replace the end of $\mathcal{A}$ with the map

$$|x\rangle|0\rangle \mapsto |x\rangle(\sqrt{1-\phi(x)}|0\rangle + \sqrt{\phi(x)}|1\rangle).$$

- Now the probability of measuring 1 on the last qubit is precisely μ.
- We can use amplitude estimation to approximate μ up to additive error ϵ, using $\mathcal{A}$ (and $\mathcal{A}^{-1}$) $O(1/\epsilon)$ times.

Ideas behind the algorithm

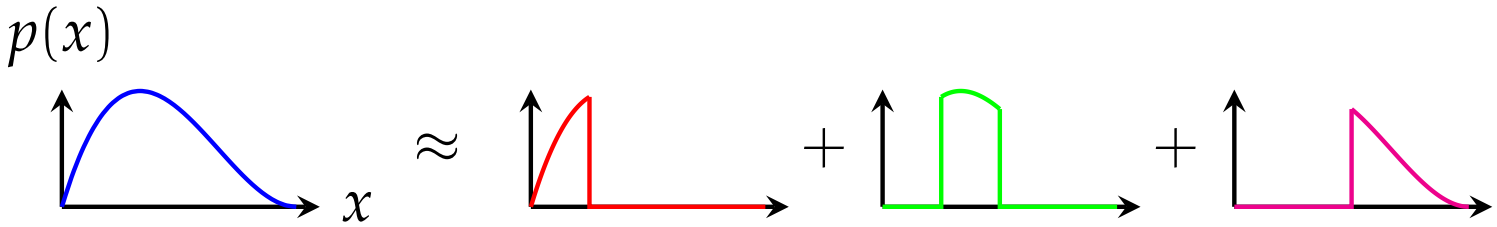
Next: the more general case where $v(\mathcal{A}) \geqslant 0$, $\mathbb{E}[v(\mathcal{A})^2] = O(1)$.

Ideas behind the algorithm

Next: the more general case where $v(\mathcal{A}) \geqslant 0$, $\mathbb{E}[v(\mathcal{A})^2] = O(1)$.

In this case (using ideas of [Heinrich '01]):

- Divide up the output values of $\mathcal{A}$ into blocks, such that in the t'th block $2^{t-1} \leqslant v(\mathcal{A}) \leqslant 2^t$.
- Use $\widetilde{O}(1/\epsilon)$ iterations of the previous algorithm to estimate the average output values in each of the first $O(\log 1/\epsilon)$ blocks, each divided by 2^t.
- Sum up the results (after rescaling them again).

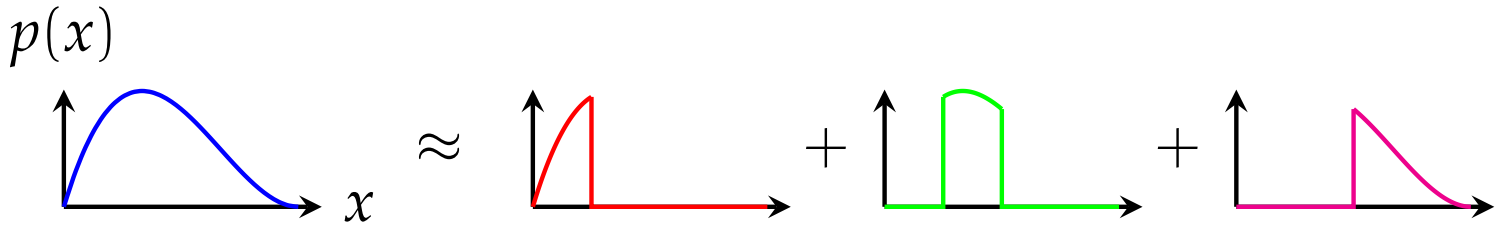


Ideas behind the algorithm

Next: the more general case where $v(\mathcal{A}) \geqslant 0$, $\mathbb{E}[v(\mathcal{A})^2] = O(1)$.

In this case (using ideas of [Heinrich '01]):

- Divide up the output values of $\mathcal{A}$ into blocks, such that in the t'th block $2^{t-1} \leqslant v(\mathcal{A}) \leqslant 2^t$.
- Use $\widetilde{O}(1/\epsilon)$ iterations of the previous algorithm to estimate the average output values in each of the first $O(\log 1/\epsilon)$ blocks, each divided by 2^t.
- Sum up the results (after rescaling them again).



The constraint that $\mathbb{E}[v(\mathcal{A})^2] = O(1)$ implies that the overall error is at most ϵ.

Ideas behind the algorithm

The final step is to change the dependence on $\mathbb{E}[v(\mathcal{A})^2]$ to a dependence on

$$\mathsf{Var}(v(\mathcal{A})) = \mathbb{E}[(v(\mathcal{A}) - \mu)^2] \leqslant \sigma^2.$$

Ideas behind the algorithm

The final step is to change the dependence on $\mathbb{E}[v(\mathcal{A})^2]$ to a dependence on

$$\mathsf{Var}(v(\mathcal{A})) = \mathbb{E}[(v(\mathcal{A}) - \mu)^2] \leqslant \sigma^2.$$

- Run $\mathcal{A}$ once and use the output $\widetilde{m}$ as a guess for μ. $|\widetilde{m} - \mu| = O(\sigma)$ with high probability.

Ideas behind the algorithm

The final step is to change the dependence on $\mathbb{E}[v(\mathcal{A})^2]$ to a dependence on

$$\mathsf{Var}(v(\mathcal{A})) = \mathbb{E}[(v(\mathcal{A}) - \mu)^2] \leqslant \sigma^2.$$

- Run $\mathcal{A}$ once and use the output $\widetilde{m}$ as a guess for μ. $|\widetilde{m} - \mu| = O(\sigma)$ with high probability.
- Apply the previous algorithm, with accuracy $O(\epsilon/\sigma)$, to the subroutine produced by subtracting $\widetilde{m}$ and dividing by σ.
- Estimate the positive and negative parts separately.

Ideas behind the algorithm

The final step is to change the dependence on $\mathbb{E}[v(\mathcal{A})^2]$ to a dependence on

$$\mathsf{Var}(v(\mathcal{A})) = \mathbb{E}[(v(\mathcal{A}) - \mu)^2] \leqslant \sigma^2.$$

- Run $\mathcal{A}$ once and use the output $\widetilde{m}$ as a guess for μ. $|\widetilde{m} - \mu| = O(\sigma)$ with high probability.
- Apply the previous algorithm, with accuracy $O(\epsilon/\sigma)$, to the subroutine produced by subtracting $\widetilde{m}$ and dividing by σ.
- Estimate the positive and negative parts separately.

A similar idea works to estimate μ up to relative error ϵ: if $\sigma^2/\mu^2 \leqslant B$, we can estimate μ up to additive error $\epsilon\,\mathbb{E}[v(\mathcal{A})]$ with $\widetilde{O}(B/\epsilon)$ uses of $\mathcal{A}$.

Application: partition functions

Consider a (classical) physical system which has state space Ω, and a Hamiltonian $H : \Omega \to \mathbb{R}$ specifying the energy of each configuration $x \in \Omega$. Assume that H takes integer values in the set $\{0, \ldots, n\}$.

Application: partition functions

Consider a (classical) physical system which has state space Ω, and a Hamiltonian $H : \Omega \to \mathbb{R}$ specifying the energy of each configuration $x \in \Omega$. Assume that H takes integer values in the set $\{0, \ldots, n\}$.

We want to compute the partition function

$$Z(\beta) = \sum_{x \in \Omega} e^{-\beta H(x)}$$

for some inverse temperature β.

Application: partition functions

Consider a (classical) physical system which has state space Ω, and a Hamiltonian $H : \Omega \to \mathbb{R}$ specifying the energy of each configuration $x \in \Omega$. Assume that H takes integer values in the set $\{0, \ldots, n\}$.

We want to compute the partition function

$$Z(\beta) = \sum_{x \in \Omega} e^{-\beta H(x)}$$

for some inverse temperature β.

Encapsulates some interesting problems:

- Physics: The Ising and Potts models
- Computer science: counting k-colourings of graphs, counting matchings (monomer-dimer coverings), ...

Application: partition functions

- $A := |\Omega|$ can be exponentially large and $Z(\beta)$ can be hard to compute; e.g. #P-hard. So we resort to randomised methods for approximating $Z(\beta)$.

- We want to approximate $Z(\beta)$ up to relative error ϵ, i.e. output $\widetilde{Z}$ such that

$$|\widetilde{Z} - Z(\beta)| \leqslant \epsilon\, Z(\beta).$$

- Assume we can exactly compute $Z(0) = A$ efficiently.

Application: partition functions

- $A := |\Omega|$ can be exponentially large and $Z(\beta)$ can be hard to compute; e.g. #P-hard. So we resort to randomised methods for approximating $Z(\beta)$.

- We want to approximate $Z(\beta)$ up to relative error ϵ, i.e. output $\widetilde{Z}$ such that

$$|\widetilde{Z} - Z(\beta)| \leqslant \epsilon\, Z(\beta).$$

- Assume we can exactly compute $Z(0) = A$ efficiently.

- One approach: multi-stage Markov chain Monte Carlo (e.g. [Valleau and Card '72, Stefankovič et al. '09]).

Multiple-stage Markov chain Monte Carlo

The basic framework of these methods:

- Let a cooling schedule be a sequence of inverse temperatures $0 = \beta_0 < \beta_1 < \cdots < \beta_\ell = \beta$.
- Express $Z(\beta_\ell)$ as the telescoping product

$$Z(\beta_\ell) = Z(\beta_0) \frac{Z(\beta_1)}{Z(\beta_0)} \frac{Z(\beta_2)}{Z(\beta_1)} \cdots \frac{Z(\beta_\ell)}{Z(\beta_{\ell-1})}.$$

Multiple-stage Markov chain Monte Carlo

The basic framework of these methods:

- Let a cooling schedule be a sequence of inverse temperatures $0 = \beta_0 < \beta_1 < \cdots < \beta_\ell = \beta$.
- Express $Z(\beta_\ell)$ as the telescoping product

$$Z(\beta_\ell) = Z(\beta_0)\frac{Z(\beta_1)}{Z(\beta_0)}\frac{Z(\beta_2)}{Z(\beta_1)}\cdots\frac{Z(\beta_\ell)}{Z(\beta_{\ell-1})}.$$

- Define random variables Y_i where $\mathbb{E}[Y_i] = Z(\beta_{i+1})/Z(\beta_i)$, with respect to the distribution π_i defined by

$$\Pr[x] = \frac{1}{Z(\beta_i)}e^{-\beta_i H(x)},$$

the Gibbs distribution at inverse temperature β_i.

Multiple-stage Markov chain Monte Carlo

The basic framework of these methods:

- Let a cooling schedule be a sequence of inverse temperatures $0 = \beta_0 < \beta_1 < \cdots < \beta_\ell = \beta$.
- Express $Z(\beta_\ell)$ as the telescoping product

$$Z(\beta_\ell) = Z(\beta_0)\frac{Z(\beta_1)}{Z(\beta_0)}\frac{Z(\beta_2)}{Z(\beta_1)}\cdots\frac{Z(\beta_\ell)}{Z(\beta_{\ell-1})}.$$

- Define random variables Y_i where $\mathbb{E}[Y_i] = Z(\beta_{i+1})/Z(\beta_i)$, with respect to the distribution π_i defined by

$$\Pr[x] = \frac{1}{Z(\beta_i)}e^{-\beta_i H(x)},$$

  the Gibbs distribution at inverse temperature β_i.
- Estimate $\mathbb{E}[Y_i]$ by sampling from this distribution.

Sampling and estimating

This idea will be efficient if we can satisfy two constraints:

1. The (relative) variance of each random variable Y_i is low: $\mathbb{E}[Y_i^2]/\mathbb{E}[Y_i]^2 = O(1)$ for all i.

2. We can (approximately) sample efficiently from the Gibbs distributions π_i.

Sampling and estimating

This idea will be efficient if we can satisfy two constraints:

1. The (relative) variance of each random variable Y_i is low: $\mathbb{E}[Y_i^2]/\mathbb{E}[Y_i]^2 = O(1)$ for all i.
2. We can (approximately) sample efficiently from the Gibbs distributions π_i.

Theorem **[Stefankovič et al. '09]**

For any partition function problem, there exists a cooling schedule satisfying the first constraint with $\ell = \widetilde{O}(\sqrt{\log A})$.

Such a cooling schedule is known as a Chebyshev cooling schedule.

Rapid mixing

To satisfy the second constraint, we can use a sequence of rapidly mixing Markov chains, each of which has a Gibbs distribution as its stationary distribution.

Rapid mixing

To satisfy the second constraint, we can use a sequence of rapidly mixing Markov chains, each of which has a Gibbs distribution as its stationary distribution.

Imagine we have a sequence of Markov chains M_i, each with stationary distribution π_i, and relaxation time at most τ. Then:

Theorem [Stefankovič et al. '09]

$Z(\beta)$ can be approximated up to relative error ϵ using $\widetilde{O}((\log A)\tau/\epsilon^2)$ steps of the Markov chains.

Rapid mixing

To satisfy the second constraint, we can use a sequence of rapidly mixing Markov chains, each of which has a Gibbs distribution as its stationary distribution.

Imagine we have a sequence of Markov chains M_i, each with stationary distribution π_i, and relaxation time at most τ. Then:

Theorem [Stefankovič et al. '09]

$Z(\beta)$ can be approximated up to relative error ϵ using $\widetilde{O}((\log A)\tau/\epsilon^2)$ steps of the Markov chains.

- In the quantum setting, we can apply our algorithm to accelerate the approximation of $\mathbb{E}[Y_i]$ (scaling goes from $O(1/\epsilon^2)$ to $\widetilde{O}(1/\epsilon)$)...
- ...and we can also replace the classical Markov chains with quantum walks to improve the dependence on τ.

Rapid mixing via quantum walks

- It turns out that the Chebyshev cooling schedule condition implies that quantum walks can be used to mix rapidly, using techniques of [Wocjan and Abeyesinghe '08].

Rapid mixing via quantum walks

- It turns out that the Chebyshev cooling schedule condition implies that quantum walks can be used to mix rapidly, using techniques of [Wocjan and Abeyesinghe '08].

- The mixing time improves from $O(\tau)$ to $O(\sqrt{\tau})$ and the final quantum complexity is $\widetilde{O}((\log A)(\sqrt{\tau}/\epsilon + \tau))$.

Rapid mixing via quantum walks

- It turns out that the Chebyshev cooling schedule condition implies that quantum walks can be used to mix rapidly, using techniques of [Wocjan and Abeyesinghe '08].

- The mixing time improves from $O(\tau)$ to $O(\sqrt{\tau})$ and the final quantum complexity is $\widetilde{O}((\log A)(\sqrt{\tau}/\epsilon + \tau))$.

Note 1: A similar idea was proposed by [Wocjan et al. '09]. However, that work needed $Z(\beta_{i+1})/Z(\beta_i) = \Omega(1)$, which would require $\ell = \Omega(\log A)$.

Rapid mixing via quantum walks

- It turns out that the Chebyshev cooling schedule condition implies that quantum walks can be used to mix rapidly, using techniques of [Wocjan and Abeyesinghe '08].

- The mixing time improves from $O(\tau)$ to $O(\sqrt{\tau})$ and the final quantum complexity is $\widetilde{O}((\log A)(\sqrt{\tau}/\epsilon + \tau))$.

Note 1: A similar idea was proposed by [Wocjan et al. '09]. However, that work needed $Z(\beta_{i+1})/Z(\beta_i) = \Omega(1)$, which would require $\ell = \Omega(\log A)$.

Note 2: The $\widetilde{O}((\log A)\tau)$ part of the bound is the complexity of computing the Chebyshev cooling schedule itself.

Example: The ferromagnetic Ising model

We are given as input a graph $G = (V, E)$ with n vertices. We consider the Ising Hamiltonian

$$H(z) = -\sum_{(u,v)\in E} z_u z_v.$$

for $z \in \{\pm 1\}^n$. We want to approximate

$$Z(\beta) = \sum_{z\in\{\pm 1\}^n} e^{-\beta H(z)}.$$

Example: The ferromagnetic Ising model

We are given as input a graph $G = (V, E)$ with n vertices. We consider the Ising Hamiltonian

$$H(z) = -\sum_{(u,v)\in E} z_u z_v.$$

for $z \in \{\pm 1\}^n$. We want to approximate

$$Z(\beta) = \sum_{z\in\{\pm 1\}^n} e^{-\beta H(z)}.$$

- Assume that we have a classical Markov chain which rapidly samples from the Gibbs distribution ($\tau = \widetilde{O}(n)$).
- This holds for low enough β (depending on the graph G).

Example: The ferromagnetic Ising model

We are given as input a graph $G = (V, E)$ with n vertices. We consider the Ising Hamiltonian

$$H(z) = -\sum_{(u,v)\in E} z_u z_v.$$

for $z \in \{\pm 1\}^n$. We want to approximate

$$Z(\beta) = \sum_{z\in\{\pm 1\}^n} e^{-\beta H(z)}.$$

- Assume that we have a classical Markov chain which rapidly samples from the Gibbs distribution ($\tau = \widetilde{O}(n)$).
- This holds for low enough β (depending on the graph G).

Then we have the following speedup:

- Best classical runtime known [Stefankovič et al. '09]: $\widetilde{O}(n^2/\epsilon^2)$
- Quantum runtime: $\widetilde{O}(n^{3/2}/\epsilon + n^2)$

Applications

There are also a number of combinatorial problems which can be expressed as partition function problems.

Applications

There are also a number of combinatorial problems which can be expressed as partition function problems.

Counting valid k-colourings of a graph G on n vertices:

- Assume, for example, that the degree of G is at most $k/2$.
- Best classical runtime known: $\widetilde{O}(n^2/\epsilon^2)$
- Quantum runtime: $\widetilde{O}(n^{3/2}/\epsilon + n^2)$

Applications

There are also a number of combinatorial problems which can be expressed as partition function problems.

Counting valid k-colourings of a graph G on n vertices:

- Assume, for example, that the degree of G is at most $k/2$.
- Best classical runtime known: $\widetilde{O}(n^2/\epsilon^2)$
- Quantum runtime: $\widetilde{O}(n^{3/2}/\epsilon + n^2)$

Counting matchings (monomer-dimer coverings) of a graph with n vertices and m edges:

- Best classical runtime known: $\widetilde{O}(n^2 m/\epsilon^2)$
- Quantum runtime: $\widetilde{O}(n^{3/2} m^{1/2}/\epsilon + n^2 m)$

Application: the total variation distance

- Imagine we can sample from probability distributions p and q on n elements.
- We would like to estimate the total variation distance

$$\|p - q\| = \frac{1}{2} \sum_x |p(x) - q(x)|$$

up to additive error ϵ.

Application: the total variation distance

- Imagine we can sample from probability distributions p and q on n elements.
- We would like to estimate the total variation distance

$$\|p - q\| = \frac{1}{2} \sum_x |p(x) - q(x)|$$

  up to additive error ϵ.
- Classically, this needs about $\Omega(n)$ samples [Valiant '11].
- Quantumly, we can do it using $O(\sqrt{n}/\epsilon^8)$ samples [Bravyi, Harrow and Hassidim '11].

Application: the total variation distance

- Imagine we can sample from probability distributions p and q on n elements.
- We would like to estimate the total variation distance

$$\|p - q\| = \frac{1}{2} \sum_x |p(x) - q(x)|$$

  up to additive error ϵ.
- Classically, this needs about $\Omega(n)$ samples [Valiant '11].
- Quantumly, we can do it using $O(\sqrt{n}/\epsilon^8)$ samples [Bravyi, Harrow and Hassidim '11].
- Using quantum mean estimation we improve this to $\widetilde{O}(\sqrt{n}/\epsilon^{3/2})$.

Application: the total variation distance

- We can write $\|p - q\| = \mathbb{E}_x[R(x)]$, where

$$R(x) = \frac{|p(x) - q(x)|}{p(x) + q(x)},$$

and x is drawn from the distribution $r = (p + q)/2$.

Application: the total variation distance

- We can write $\|p - q\| = \mathbb{E}_x[R(x)]$, where

$$R(x) = \frac{|p(x) - q(x)|}{p(x) + q(x)},$$

  and x is drawn from the distribution $r = (p + q)/2$.

- For each x, we can use amplitude estimation to estimate $R(x)$.

- It's sufficient to use $\widetilde{O}(\sqrt{n/\epsilon})$ iterations of amplitude estimation to approximate $\mathbb{E}_x[R(x)]$ up to additive error ϵ.

Application: the total variation distance

- We can write $\|p - q\| = \mathbb{E}_x[R(x)]$, where

$$R(x) = \frac{|p(x) - q(x)|}{p(x) + q(x)},$$

  and x is drawn from the distribution $r = (p + q)/2$.

- For each x, we can use amplitude estimation to estimate $R(x)$.

- It's sufficient to use $\widetilde{O}(\sqrt{n/\epsilon})$ iterations of amplitude estimation to approximate $\mathbb{E}_x[R(x)]$ up to additive error ϵ.

- Wrapping this within $O(1/\epsilon)$ iterations of the mean-estimation algorithm, we obtain an overall algorithm running in time $\widetilde{O}(\sqrt{n}/\epsilon^{3/2})$.

Summary

- There is a quantum algorithm which estimates μ up to additive error ϵ with $\widetilde{O}(\sigma/\epsilon)$ uses of $\mathcal{A}$.

Summary

- There is a quantum algorithm which estimates μ up to additive error ϵ with $\widetilde{O}(\sigma/\epsilon)$ uses of $\mathcal{A}$.
- We can use this to approximate partition functions more quickly than the best classical algorithms known.

Summary

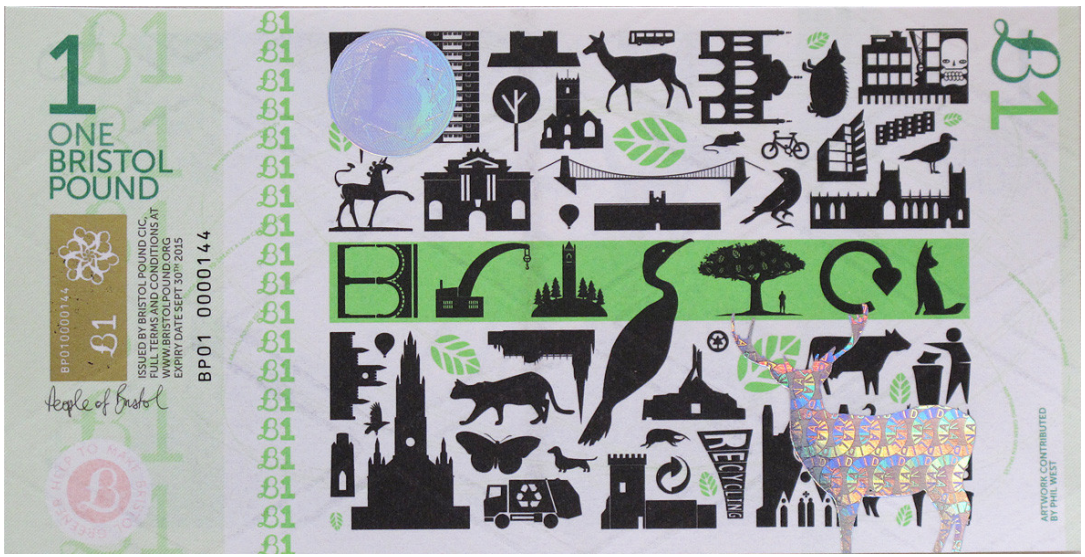
- There is a quantum algorithm which estimates μ up to additive error ϵ with $\widetilde{O}(\sigma/\epsilon)$ uses of $\mathcal{A}$.
- We can use this to approximate partition functions more quickly than the best classical algorithms known.
- Open problem: Is there a more efficient quantum algorithm for computing a Chebyshev cooling schedule?

Summary

- There is a quantum algorithm which estimates μ up to additive error ϵ with $\widetilde{O}(\sigma/\epsilon)$ uses of $\mathcal{A}$.
- We can use this to approximate partition functions more quickly than the best classical algorithms known.
- Open problem: Is there a more efficient quantum algorithm for computing a Chebyshev cooling schedule?

Summary

- There is a quantum algorithm which estimates μ up to additive error ϵ with $\widetilde{O}(\sigma/\epsilon)$ uses of $\mathcal{A}$.
- We can use this to approximate partition functions more quickly than the best classical algorithms known.
- Open problem: Is there a more efficient quantum algorithm for computing a Chebyshev cooling schedule?



Thanks!