Quantum speedup of backtracking and Monte Carlo algorithms

Ashley Montanaro

School of Mathematics,
University of Bristol

19 February 2016

arXiv:1504.06987 and arXiv:1509.02374

Proc. R. Soc. A 2015 471 20150301

University of BRISTOL

EPSRC
Engineering and Physical Sciences Research Council

Quantum speedup of classical algorithms

Which general classical algorithmic techniques can we speed up using a quantum computer?

Quantum speedup of classical algorithms

Which general classical algorithmic techniques can we speed up using a quantum computer?

Some examples:

- Unstructured search [Grover '96]

Quantum speedup of classical algorithms

Which general classical algorithmic techniques can we speed up using a quantum computer?

Some examples:

- Unstructured search [Grover '96]
- Probability amplification: boost the success probability of a randomised algorithm to 99% [Brassard et al. '02]

Quantum speedup of classical algorithms

Which general classical algorithmic techniques can we speed up using a quantum computer?

Some examples:

- Unstructured search [Grover '96]
- Probability amplification: boost the success probability of a randomised algorithm to 99% [Brassard et al. '02]
- Probability estimation: determine the success probability of a randomised algorithm up to 1% relative error [Brassard et al. '02]

Quantum speedup of classical algorithms

Which general classical algorithmic techniques can we speed up using a quantum computer?

Some examples:

- Unstructured search [Grover '96]
- Probability amplification: boost the success probability of a randomised algorithm to 99% [Brassard et al. '02]
- Probability estimation: determine the success probability of a randomised algorithm up to 1% relative error [Brassard et al. '02]
- Simulated annealing [Somma et al. '07]

Quantum speedup of classical algorithms

Which general classical algorithmic techniques can we speed up using a quantum computer?

Some examples:

- Unstructured search [Grover '96]
- Probability amplification: boost the success probability of a randomised algorithm to 99% [Brassard et al. '02]
- Probability estimation: determine the success probability of a randomised algorithm up to 1% relative error [Brassard et al. '02]
- Simulated annealing [Somma et al. '07]

In all of these cases, there are quantum algorithms which achieve quadratic speedups over the corresponding classical algorithm.

Today's talk

Two other standard classical algorithmic techniques which can be accelerated by quantum algorithms:

Today's talk

Two other standard classical algorithmic techniques which can be accelerated by quantum algorithms:

- Backtracking: a standard method for solving constraint satisfaction problems

Today's talk

Two other standard classical algorithmic techniques which can be accelerated by quantum algorithms:

- Backtracking: a standard method for solving constraint satisfaction problems
- Approximating the mean of a random variable with bounded variance: the core of Monte Carlo methods

Today's talk

Two other standard classical algorithmic techniques which can be accelerated by quantum algorithms:

- Backtracking: a standard method for solving constraint satisfaction problems
- Approximating the mean of a random variable with bounded variance: the core of Monte Carlo methods

In both cases, we obtain quadratic quantum speedups.

Today's talk

Two other standard classical algorithmic techniques which can be accelerated by quantum algorithms:

- Backtracking: a standard method for solving constraint satisfaction problems
- Approximating the mean of a random variable with bounded variance: the core of Monte Carlo methods

In both cases, we obtain quadratic quantum speedups.

The quantum algorithms use different techniques:

- The backtracking algorithm uses quantum walks, based on an algorithm of [Belovs '13].
- The mean-approximation algorithm uses amplitude amplification, based on ideas of [Heinrich '01].

Constraint satisfaction problems

Backtracking is a general approach to solve constraint satisfaction problems (CSPs).

Constraint satisfaction problems

Backtracking is a general approach to solve constraint satisfaction problems (CSPs).

- An instance of a CSP on n variables $x_1, \ldots, x_n$ is specified by a sequence of constraints, all of which must be satisfied by the variables.

Constraint satisfaction problems

Backtracking is a general approach to solve constraint satisfaction problems (CSPs).

- An instance of a CSP on n variables $x_1, \ldots, x_n$ is specified by a sequence of constraints, all of which must be satisfied by the variables.
- We might want to find one assignment to $x_1, \ldots, x_n$ that satisfies all the constraints, or list all such assignments.

Constraint satisfaction problems

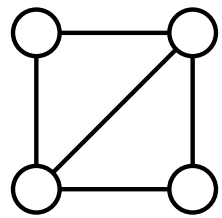
Backtracking is a general approach to solve constraint satisfaction problems (CSPs).

- An instance of a CSP on n variables $x_1, \ldots, x_n$ is specified by a sequence of constraints, all of which must be satisfied by the variables.
- We might want to find one assignment to $x_1, \ldots, x_n$ that satisfies all the constraints, or list all such assignments.
- For many CSPs, the best algorithms known for either task have exponential runtime in n.

Constraint satisfaction problems

Backtracking is a general approach to solve constraint satisfaction problems (CSPs).

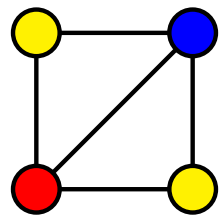
- An instance of a CSP on n variables $x_1, \ldots, x_n$ is specified by a sequence of constraints, all of which must be satisfied by the variables.
- We might want to find one assignment to $x_1, \ldots, x_n$ that satisfies all the constraints, or list all such assignments.
- For many CSPs, the best algorithms known for either task have exponential runtime in n.
- A simple example: graph 3-colouring.

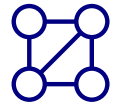


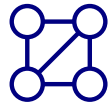
Constraint satisfaction problems

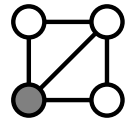
Backtracking is a general approach to solve constraint satisfaction problems (CSPs).

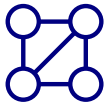
- An instance of a CSP on n variables $x_1, \ldots, x_n$ is specified by a sequence of constraints, all of which must be satisfied by the variables.
- We might want to find one assignment to $x_1, \ldots, x_n$ that satisfies all the constraints, or list all such assignments.
- For many CSPs, the best algorithms known for either task have exponential runtime in n.
- A simple example: graph 3-colouring.

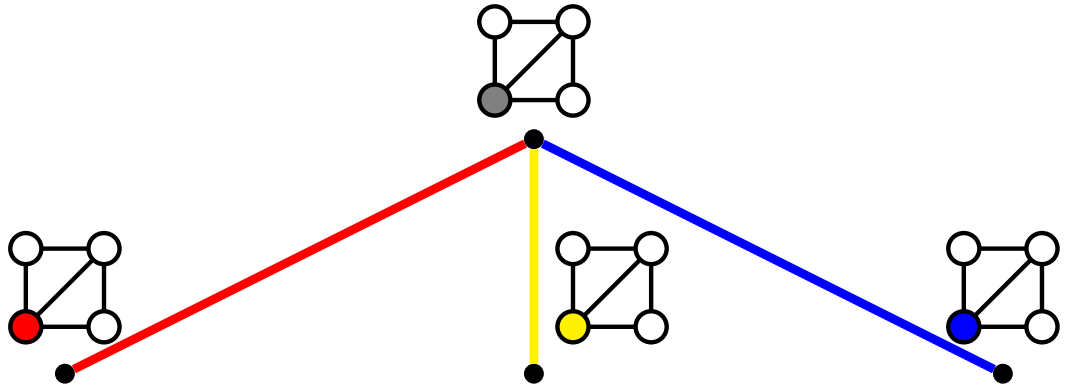


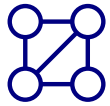
Colouring  by trial and error

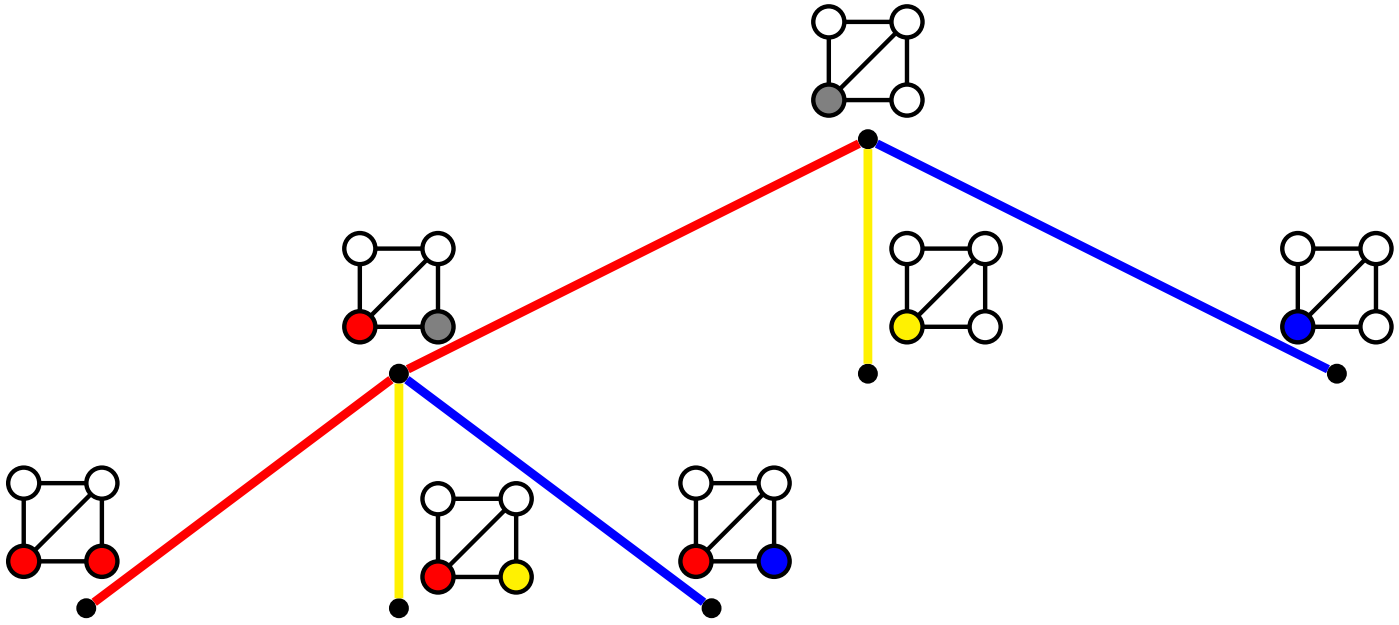
Colouring  by trial and error

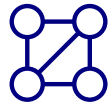


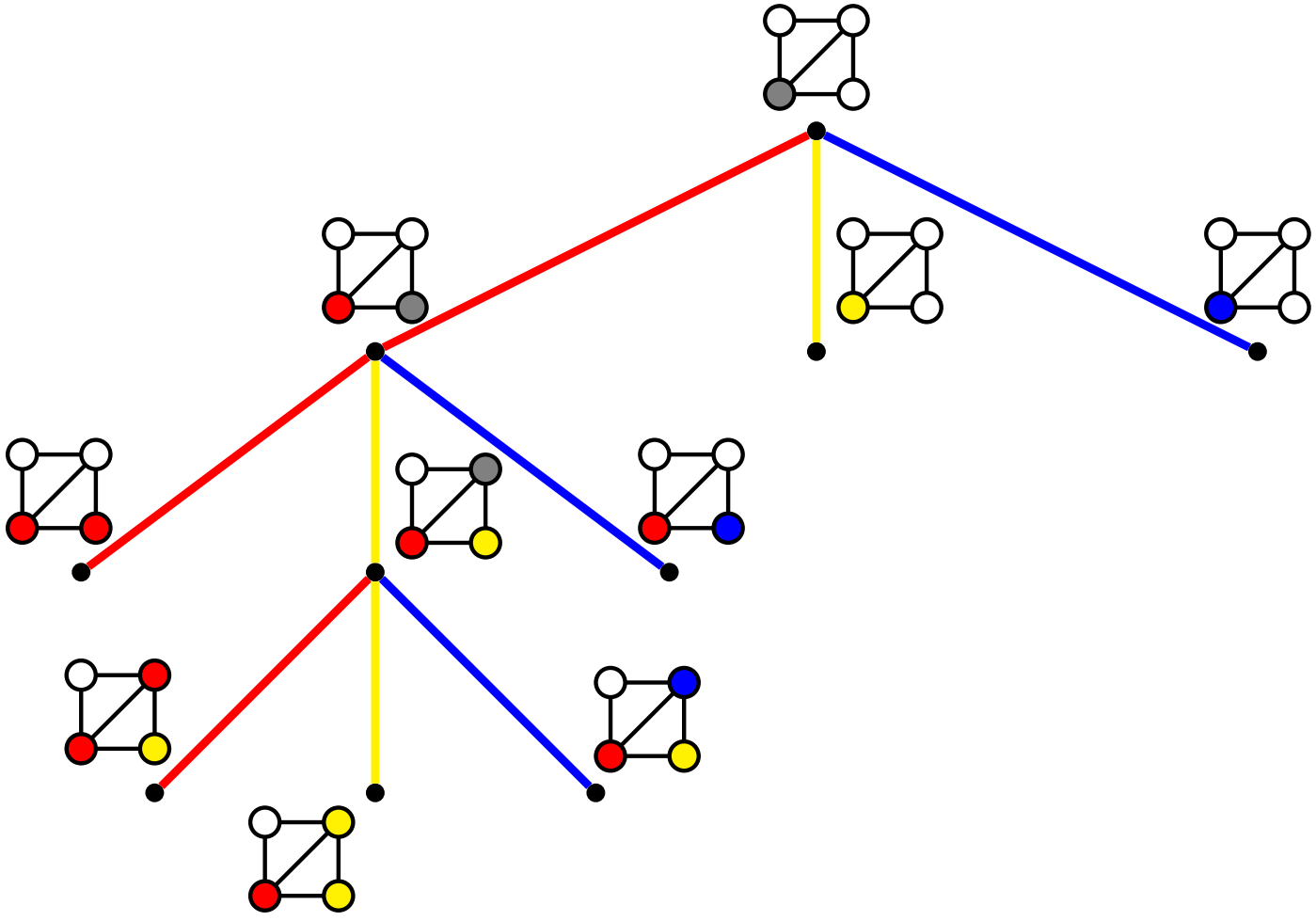
Colouring  by trial and error

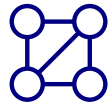


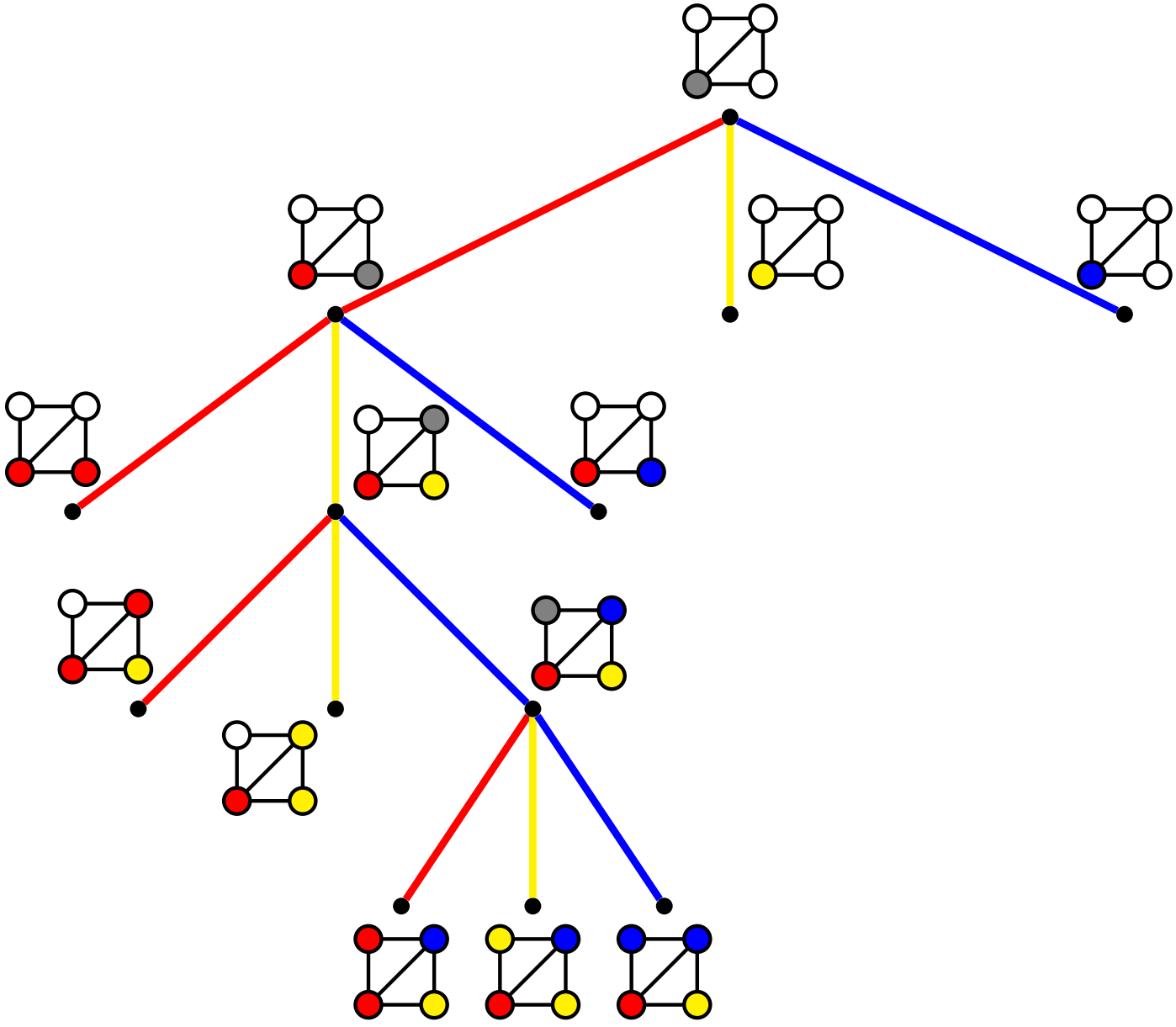
Colouring  by trial and error

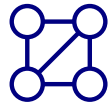


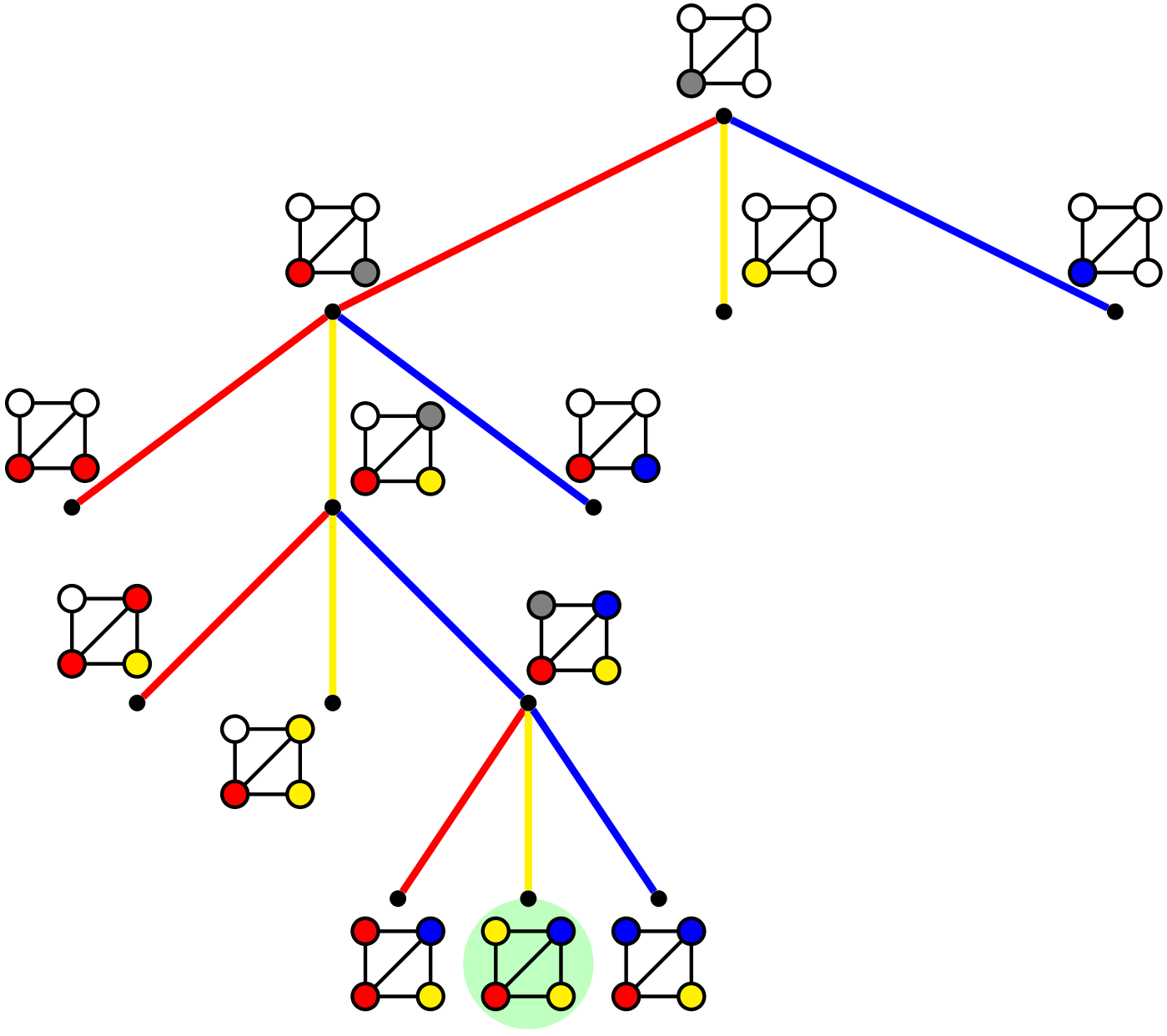
Colouring  by trial and error



Colouring  by trial and error



Colouring  by trial and error



General backtracking framework

This idea, known as backtracking, can be applied to any CSP, given the following assumptions:

- We have a problem on n variables, each picked from $[d] := \{0, \ldots, d-1\}$. Write $\mathcal{D} := ([d] \cup \{*\})^n$ for the set of partial assignments, where $*$ means "not assigned yet".

General backtracking framework

This idea, known as backtracking, can be applied to any CSP, given the following assumptions:

- We have a problem on n variables, each picked from $[d] := \{0, \ldots, d-1\}$. Write $\mathcal{D} := ([d] \cup \{*\})^n$ for the set of partial assignments, where $*$ means "not assigned yet".
- We have access to a predicate

$$P : \mathcal{D} \to \{\text{true}, \text{false}, \text{indeterminate}\}$$

  which tells us the status of a partial assignment.

General backtracking framework

This idea, known as backtracking, can be applied to any CSP, given the following assumptions:

- We have a problem on n variables, each picked from $[d] := \{0, \ldots, d-1\}$. Write $\mathcal{D} := ([d] \cup \{*\})^n$ for the set of partial assignments, where $*$ means "not assigned yet".
- We have access to a predicate

$$P : \mathcal{D} \to \{\text{true}, \text{false}, \text{indeterminate}\}$$

  which tells us the status of a partial assignment.
- We have access to a heuristic

$$h : \mathcal{D} \to \{1, \ldots, n\}$$

  which determines which variable to choose next, for a given partial assignment.

Main result [AM '15]

Theorem

Let T be the number of vertices in the backtracking tree. Then there is a bounded-error quantum algorithm which evaluates P and h $O(\sqrt{T}n^{3/2}\log n)$ times each, and outputs x such that $P(x)$ is true, or "not found" if no such x exists.

Main result [AM '15]

Theorem

Let T be the number of vertices in the backtracking tree. Then there is a bounded-error quantum algorithm which evaluates P and h $O(\sqrt{T}n^{3/2}\log n)$ times each, and outputs x such that $P(x)$ is true, or "not found" if no such x exists.

If we are promised that there exists a unique x_0 such that $P(x_0)$ is true, this is improved to $O(\sqrt{Tn}\log^3 n)$.

In both cases the algorithm uses $\mathrm{poly}(n)$ space and $\mathrm{poly}(n)$ auxiliary quantum gates per use of P and h.

Main result [AM '15]

Theorem

Let T be the number of vertices in the backtracking tree. Then there is a bounded-error quantum algorithm which evaluates P and h $O(\sqrt{T}n^{3/2}\log n)$ times each, and outputs x such that $P(x)$ is true, or "not found" if no such x exists.

If we are promised that there exists a unique x_0 such that $P(x_0)$ is true, this is improved to $O(\sqrt{Tn}\log^3 n)$.

In both cases the algorithm uses $\mathrm{poly}(n)$ space and $\mathrm{poly}(n)$ auxiliary quantum gates per use of P and h.

- The algorithm can be modified to find all solutions by striking out previously seen solutions.

Main result [AM '15]

Theorem

Let T be the number of vertices in the backtracking tree. Then there is a bounded-error quantum algorithm which evaluates P and h $O(\sqrt{T}n^{3/2}\log n)$ times each, and outputs x such that $P(x)$ is true, or "not found" if no such x exists.

If we are promised that there exists a unique x_0 such that $P(x_0)$ is true, this is improved to $O(\sqrt{Tn}\log^3 n)$.

In both cases the algorithm uses $\mathrm{poly}(n)$ space and $\mathrm{poly}(n)$ auxiliary quantum gates per use of P and h.

- The algorithm can be modified to find all solutions by striking out previously seen solutions.
- We usually think of T as being exponentially large in n. In this regime, this is a near-quadratic separation.

Main result [AM '15]

Theorem

Let T be the number of vertices in the backtracking tree. Then there is a bounded-error quantum algorithm which evaluates P and h $O(\sqrt{T}n^{3/2}\log n)$ times each, and outputs x such that $P(x)$ is true, or "not found" if no such x exists.

If we are promised that there exists a unique x_0 such that $P(x_0)$ is true, this is improved to $O(\sqrt{Tn}\log^3 n)$.

In both cases the algorithm uses $\mathrm{poly}(n)$ space and $\mathrm{poly}(n)$ auxiliary quantum gates per use of P and h.

- The algorithm can be modified to find all solutions by striking out previously seen solutions.
- We usually think of T as being exponentially large in n. In this regime, this is a near-quadratic separation.
- Note that the algorithm does not need to know T.

Previous work

Some previous works have developed quantum algorithms related to backtracking:

- [Cerf, Grover and Williams '00] developed a quantum algorithm for constraint satisfaction problems, based on a nested version of Grover search. This can be seen as a quantum version of one particular backtracking algorithm that runs quadratically faster.

Previous work

Some previous works have developed quantum algorithms related to backtracking:

- [Cerf, Grover and Williams '00] developed a quantum algorithm for constraint satisfaction problems, based on a nested version of Grover search. This can be seen as a quantum version of one particular backtracking algorithm that runs quadratically faster.

- [Farhi and Gutmann '98] used continuous-time quantum walks to find solutions in backtracking trees. They showed that, for some trees, the quantum walk can find a solution exponentially faster than a classical random walk.

Previous work

Some previous works have developed quantum algorithms related to backtracking:

- [Cerf, Grover and Williams '00] developed a quantum algorithm for constraint satisfaction problems, based on a nested version of Grover search. This can be seen as a quantum version of one particular backtracking algorithm that runs quadratically faster.

- [Farhi and Gutmann '98] used continuous-time quantum walks to find solutions in backtracking trees. They showed that, for some trees, the quantum walk can find a solution exponentially faster than a classical random walk.

By contrast, the algorithm presented here achieves a (nearly) quadratic separation for all trees.

Search in the backtracking tree

Idea: Use quantum search to find a solution ("marked vertex") in the tree produced by the backtracking algorithm.

Search in the backtracking tree

Idea: Use quantum search to find a solution ("marked vertex") in the tree produced by the backtracking algorithm.

Many works have studied quantum search in various graphs, e.g. [Szegedy '04], [Aaronson and Ambainis '05], [Magniez et al. '11] ...

Search in the backtracking tree

Idea: Use quantum search to find a solution ("marked vertex") in the tree produced by the backtracking algorithm.

Many works have studied quantum search in various graphs, e.g. [Szegedy '04], [Aaronson and Ambainis '05], [Magniez et al. '11] ...

But here there are some difficulties:

- The graph is not known in advance, and is determined by the backtracking algorithm.

Search in the backtracking tree

Idea: Use quantum search to find a solution ("marked vertex") in the tree produced by the backtracking algorithm.

Many works have studied quantum search in various graphs, e.g. [Szegedy '04], [Aaronson and Ambainis '05], [Magniez et al. '11] ...

But here there are some difficulties:

- The graph is not known in advance, and is determined by the backtracking algorithm.
- We start at the root of the tree, not in the stationary distribution of a random walk on the graph.

Search in the backtracking tree

Idea: Use quantum search to find a solution ("marked vertex") in the tree produced by the backtracking algorithm.

Many works have studied quantum search in various graphs, e.g. [Szegedy '04], [Aaronson and Ambainis '05], [Magniez et al. '11] ...

But here there are some difficulties:

- The graph is not known in advance, and is determined by the backtracking algorithm.
- We start at the root of the tree, not in the stationary distribution of a random walk on the graph.

These can be overcome using work of [Belovs '13] relating quantum walks to effective resistance in an electrical network.

Search by quantum walk (sketch)

We apply phase estimation to a quantum walk starting at the root, with precision $O(1/\sqrt{Tn})$, where n is an upper bound on the depth of the tree, and output "solution exists" if the eigenvalue is 1, and "no solution" otherwise.

Search by quantum walk (sketch)

We apply phase estimation to a quantum walk starting at the root, with precision $O(1/\sqrt{Tn})$, where n is an upper bound on the depth of the tree, and output "solution exists" if the eigenvalue is 1, and "no solution" otherwise.

Claim (special case of [Belovs '13])

This procedure succeeds with probability $O(1)$.

Search by quantum walk (sketch)

We apply phase estimation to a quantum walk starting at the root, with precision $O(1/\sqrt{Tn})$, where n is an upper bound on the depth of the tree, and output "solution exists" if the eigenvalue is 1, and "no solution" otherwise.

Claim (special case of [Belovs '13])

This procedure succeeds with probability $O(1)$.

- So we can detect the existence of a solution with $O(\sqrt{Tn})$ quantum walk steps.

Search by quantum walk (sketch)

We apply phase estimation to a quantum walk starting at the root, with precision $O(1/\sqrt{Tn})$, where n is an upper bound on the depth of the tree, and output "solution exists" if the eigenvalue is 1, and "no solution" otherwise.

Claim (special case of [Belovs '13])

This procedure succeeds with probability $O(1)$.

- So we can detect the existence of a solution with $O(\sqrt{Tn})$ quantum walk steps.
- Each quantum walk step can be implemented with $O(1)$ uses of P and h.

Search by quantum walk (sketch)

We apply phase estimation to a quantum walk starting at the root, with precision $O(1/\sqrt{Tn})$, where n is an upper bound on the depth of the tree, and output "solution exists" if the eigenvalue is 1, and "no solution" otherwise.

Claim (special case of [Belovs '13])

This procedure succeeds with probability $O(1)$.

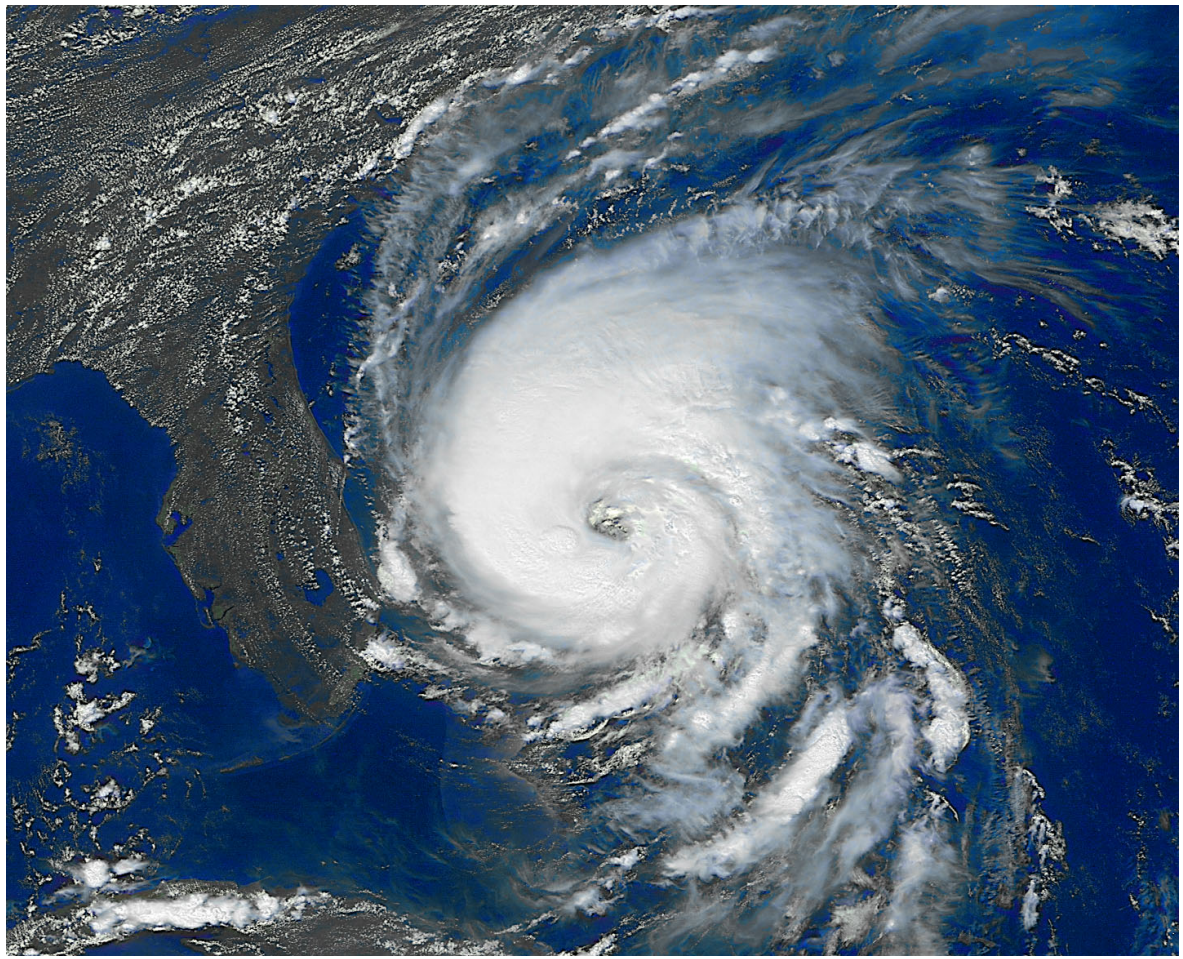
- So we can detect the existence of a solution with $O(\sqrt{Tn})$ quantum walk steps.
- Each quantum walk step can be implemented with $O(1)$ uses of P and h.
- We can also find a solution using binary search with a small overhead.

Part 2: Monte Carlo methods

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

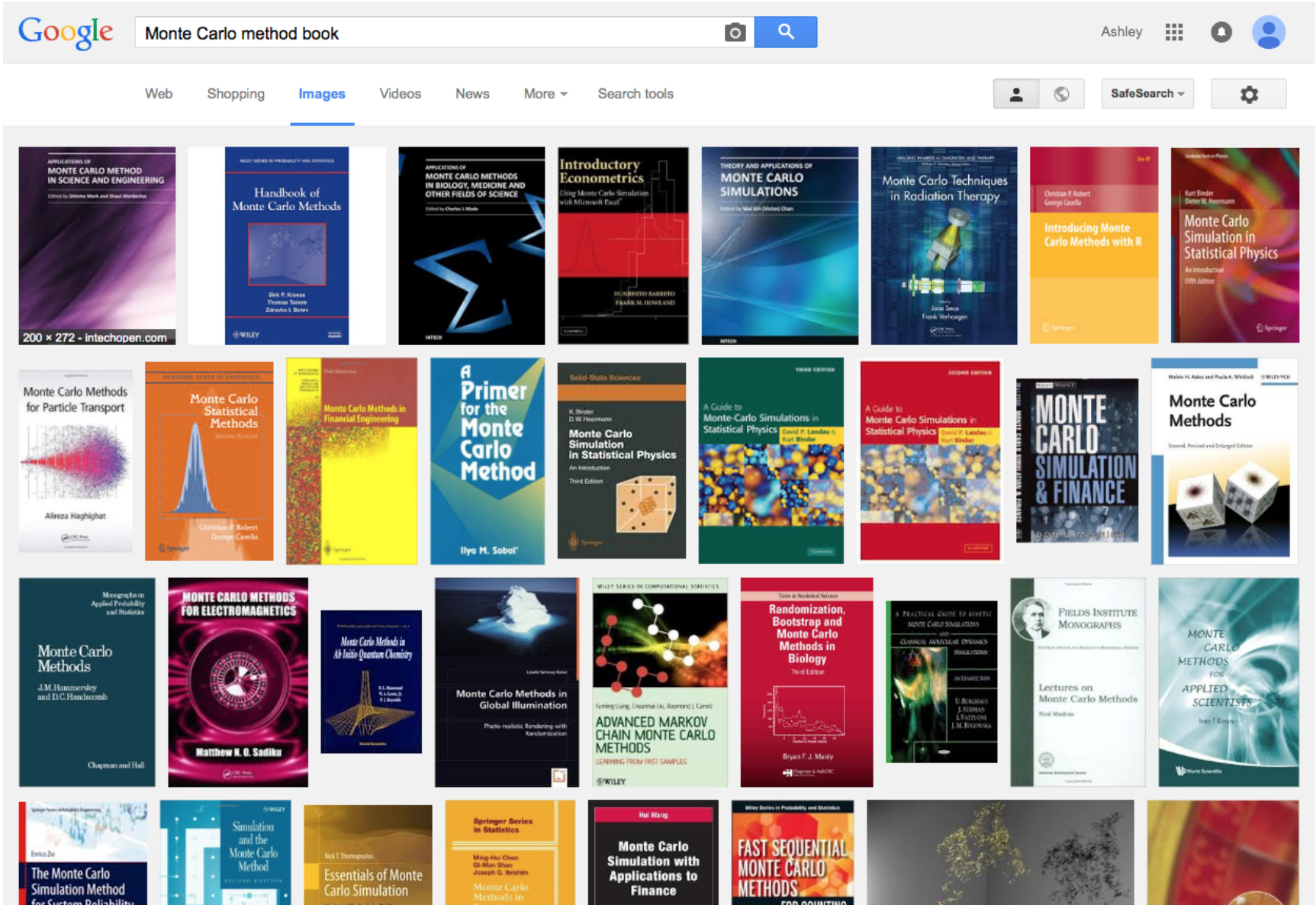
Part 2: Monte Carlo methods

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

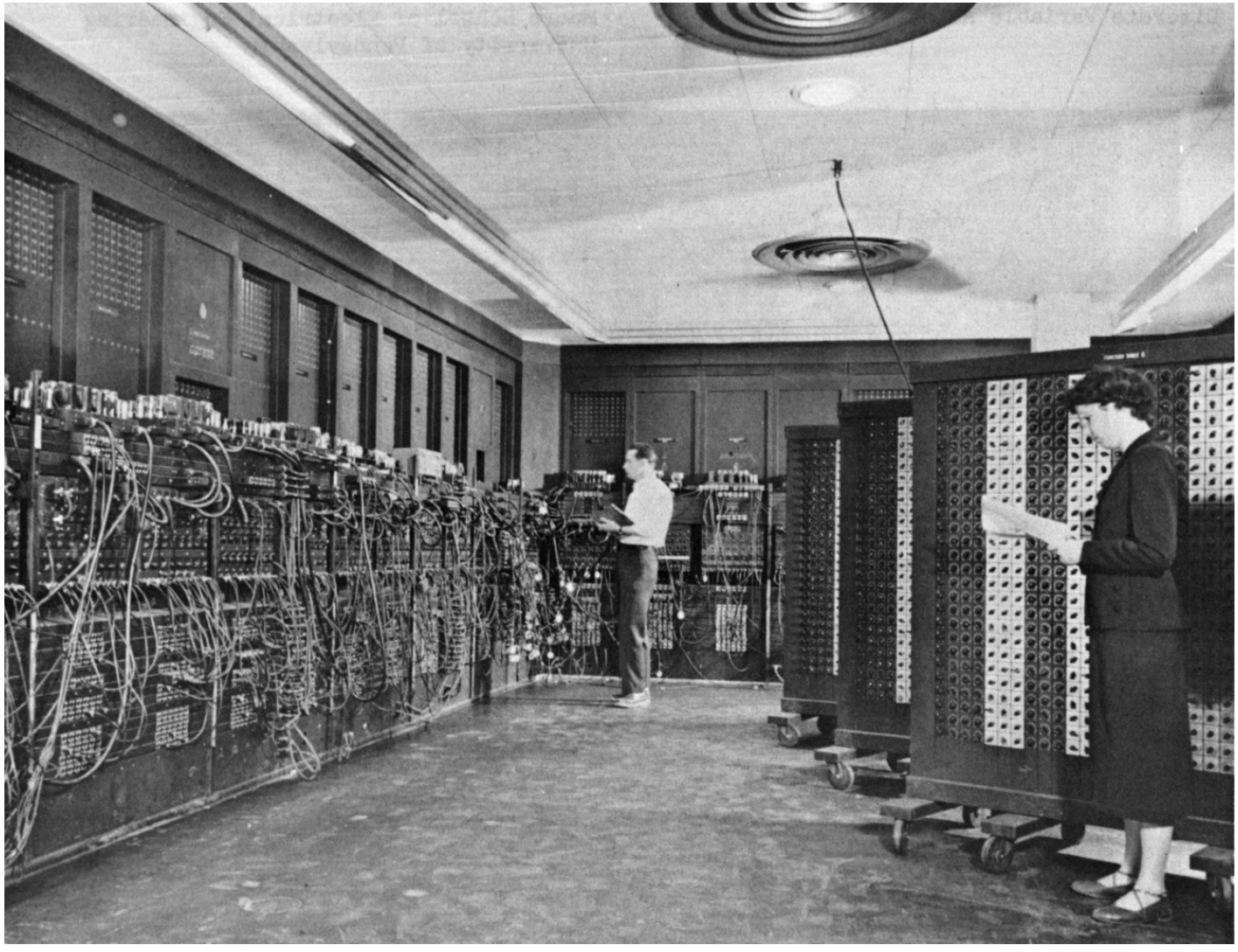


Pic: Wikipedia

These methods are used throughout science and engineering:



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



Pic: Wikipedia

Monte Carlo methods

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].

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, Ortiz and Somma '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, Ortiz and Somma '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, Ortiz and Somma '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]. A sketch of the argument:

- If we know that the output of $\mathcal{A}$ is bounded in $[0, 1]$, we can use amplitude estimation to approximate μ up to ϵ, using $\mathcal{A}$ (and $\mathcal{A}^{-1}$) $O(1/\epsilon)$ times.

Ideas behind the algorithm

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

- If we know that the output of $\mathcal{A}$ is bounded in $[0, 1]$, we can use amplitude estimation to approximate μ up to ϵ, using $\mathcal{A}$ (and $\mathcal{A}^{-1}$) $O(1/\epsilon)$ times.
- So divide up the output values of $\mathcal{A}$ into blocks of exponentially increasing distance from μ.

Ideas behind the algorithm

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

- If we know that the output of $\mathcal{A}$ is bounded in $[0, 1]$, we can use amplitude estimation to approximate μ up to ϵ, using $\mathcal{A}$ (and $\mathcal{A}^{-1}$) $O(1/\epsilon)$ times.
- So divide up the output values of $\mathcal{A}$ into blocks of exponentially increasing distance from μ.
- Rescale and shift the values in each block to be bounded in $[0, 1]$. Then use amplitude estimation to estimate the average output value in each block.

Ideas behind the algorithm

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

- If we know that the output of $\mathcal{A}$ is bounded in $[0, 1]$, we can use amplitude estimation to approximate μ up to ϵ, using $\mathcal{A}$ (and $\mathcal{A}^{-1}$) $O(1/\epsilon)$ times.
- So divide up the output values of $\mathcal{A}$ into blocks of exponentially increasing distance from μ.
- Rescale and shift the values in each block to be bounded in $[0, 1]$. Then use amplitude estimation to estimate the average output value in each block.
- Sum up the results (after rescaling them again).

Ideas behind the algorithm

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

- If we know that the output of $\mathcal{A}$ is bounded in $[0, 1]$, we can use amplitude estimation to approximate μ up to ϵ, using $\mathcal{A}$ (and $\mathcal{A}^{-1}$) $O(1/\epsilon)$ times.
- So divide up the output values of $\mathcal{A}$ into blocks of exponentially increasing distance from μ.
- Rescale and shift the values in each block to be bounded in $[0, 1]$. Then use amplitude estimation to estimate the average output value in each block.
- Sum up the results (after rescaling them again).

This works because, if the variance of $\mathcal{A}$ is low, output values far from μ do not contribute much to μ, so can be estimated with lower precision.

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

- $|\Omega|$ 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

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

Application: partition functions

- $|\Omega|$ 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

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

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

Application: partition functions

- $|\Omega|$ 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

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

- A standard classical approach: multi-stage Markov chain Monte Carlo (e.g. [Valleau and Card '72, Stefankovič et al. '09]).
- We can apply the above quantum algorithm to speed up an approximation of expected values in this approach...

Application: partition functions

- $|\Omega|$ 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

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

- A standard classical approach: multi-stage Markov chain Monte Carlo (e.g. [Valleau and Card '72, Stefankovič et al. '09]).
- We can apply the above quantum algorithm to speed up an approximation of expected values in this approach...
- ...and we can also replace the classical Markov chains with quantum walks to get an additional improvement, based on techniques of [Wocjan and Abeyesinghe '08].

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 samples from the Gibbs distribution in time $\widetilde{O}(n)$.

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 samples from the Gibbs distribution in time $\widetilde{O}(n)$.
- 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

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

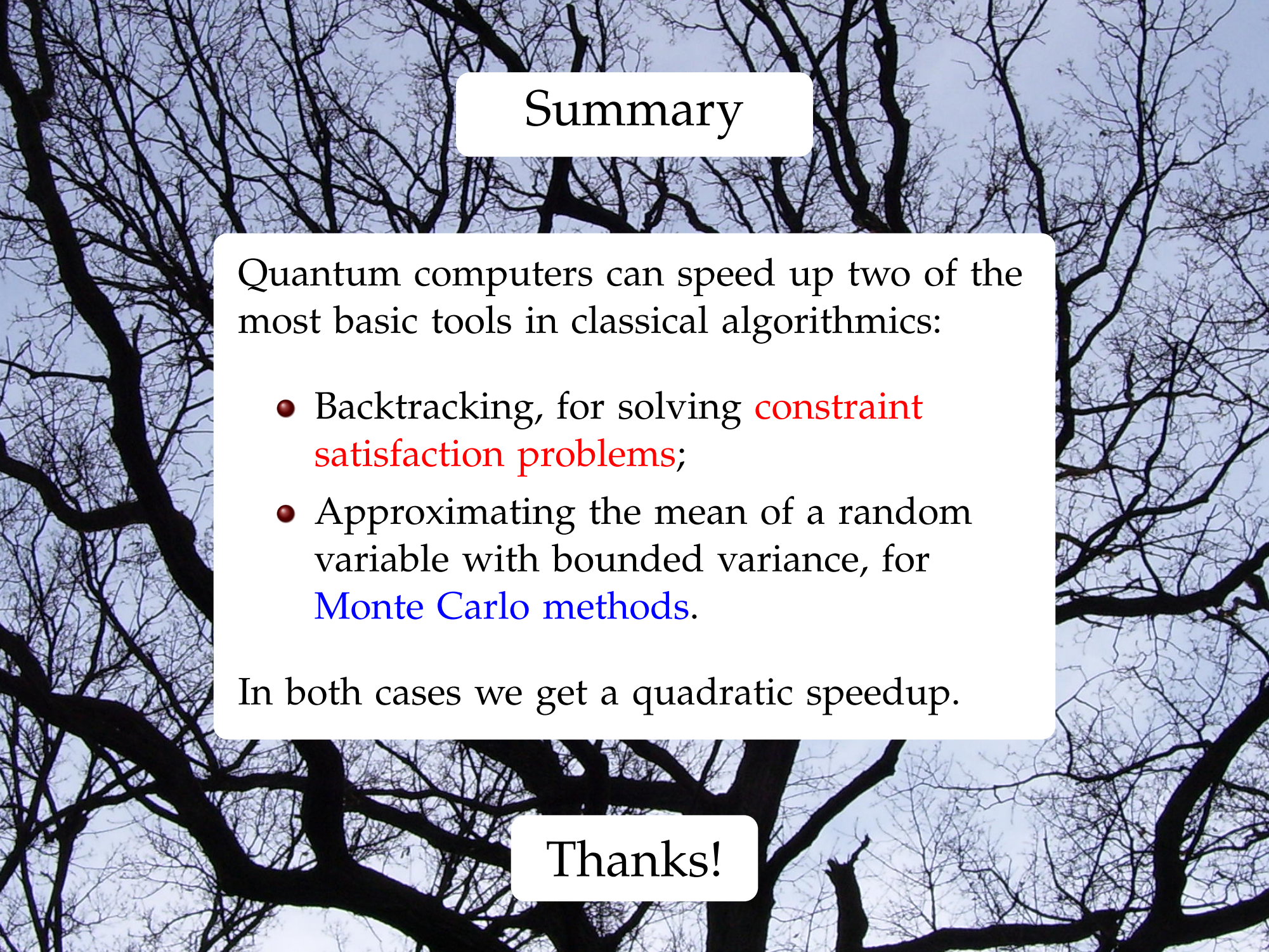
for $z \in \{\pm 1\}^n$. We want to approximate

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

- Assume that we have a classical Markov chain which samples from the Gibbs distribution in time $\widetilde{O}(n)$.
- 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: $\widetilde{O}(n^{3/2}/\epsilon + n^2)$



Summary

Quantum computers can speed up two of the most basic tools in classical algorithmics:

- Backtracking, for solving constraint satisfaction problems;
- Approximating the mean of a random variable with bounded variance, for Monte Carlo methods.

In both cases we get a quadratic speedup.

Thanks!

Quantum walk in a tree

The quantum walk operates on a T-dimensional Hilbert space spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \ldots, T-1\}\}$, where r is the root.

Quantum walk in a tree

The quantum walk operates on a T-dimensional Hilbert space spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \ldots, T-1\}\}$, where r is the root.

The walk starts in the state $|r\rangle$ and is based on a set of diffusion operators D_x, where D_x acts on the subspace $\mathcal{H}_x$ spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$:

Quantum walk in a tree

The quantum walk operates on a T-dimensional Hilbert space spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \ldots, T-1\}\}$, where r is the root.

The walk starts in the state $|r\rangle$ and is based on a set of diffusion operators D_x, where D_x acts on the subspace $\mathcal{H}_x$ spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$:

- If x is marked, then D_x is the identity.

Quantum walk in a tree

The quantum walk operates on a T-dimensional Hilbert space spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \ldots, T-1\}\}$, where r is the root.

The walk starts in the state $|r\rangle$ and is based on a set of diffusion operators D_x, where D_x acts on the subspace $\mathcal{H}_x$ spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$:

- If x is marked, then D_x is the identity.
- If x is not marked, and $x \neq r$, then $D_x = I - 2|\psi_x\rangle\langle\psi_x|$, where

$$|\psi_x\rangle \propto |x\rangle + \sum_{y, x \to y} |y\rangle.$$

Quantum walk in a tree

The quantum walk operates on a T-dimensional Hilbert space spanned by $\{|r\rangle\} \cup \{|x\rangle : x \in \{1, \ldots, T-1\}\}$, where r is the root.

The walk starts in the state $|r\rangle$ and is based on a set of diffusion operators D_x, where D_x acts on the subspace $\mathcal{H}_x$ spanned by $\{|x\rangle\} \cup \{|y\rangle : x \to y\}$:

- If x is marked, then D_x is the identity.
- If x is not marked, and $x \neq r$, then $D_x = I - 2|\psi_x\rangle\langle\psi_x|$, where
$$|\psi_x\rangle \propto |x\rangle + \sum_{y, x \to y} |y\rangle.$$
- $D_r = I - 2|\psi_r\rangle\langle\psi_r|$, where
$$|\psi_r\rangle \propto |r\rangle + \sqrt{n} \sum_{y, r \to y} |y\rangle.$$

Quantum walk in a tree

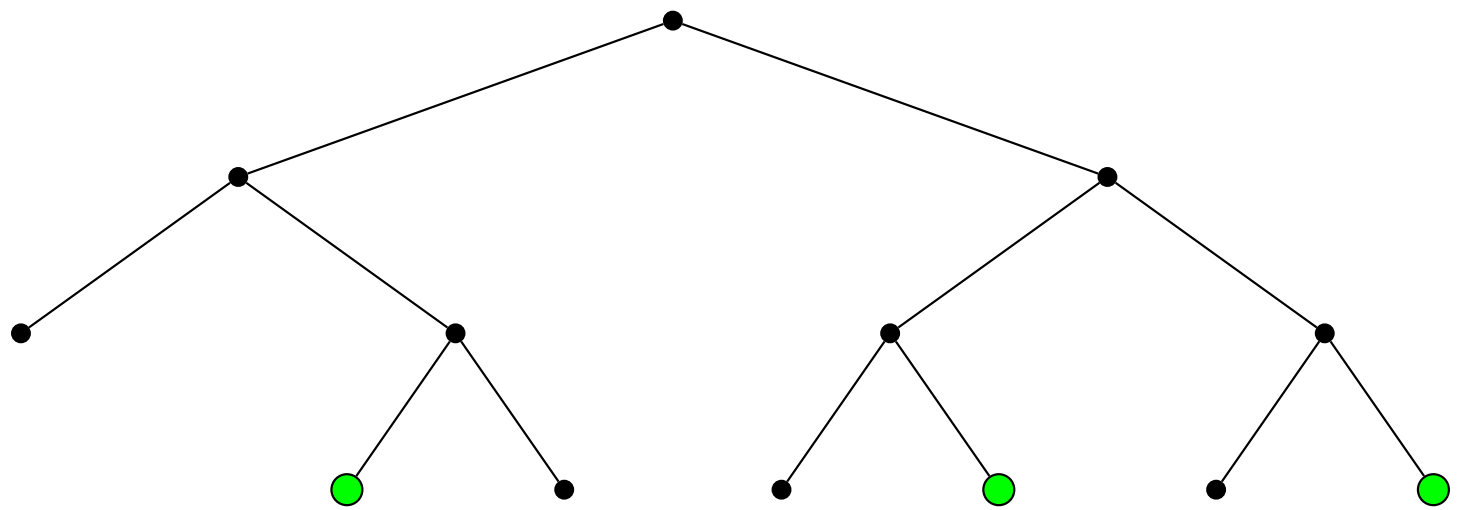
Let A and B be the sets of vertices an even and odd distance from the root, respectively.

Then a step of the walk consists of applying the operator $R_B R_A$, where $R_A = \bigoplus_{x \in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$.

Quantum walk in a tree

Let A and B be the sets of vertices an even and odd distance from the root, respectively.

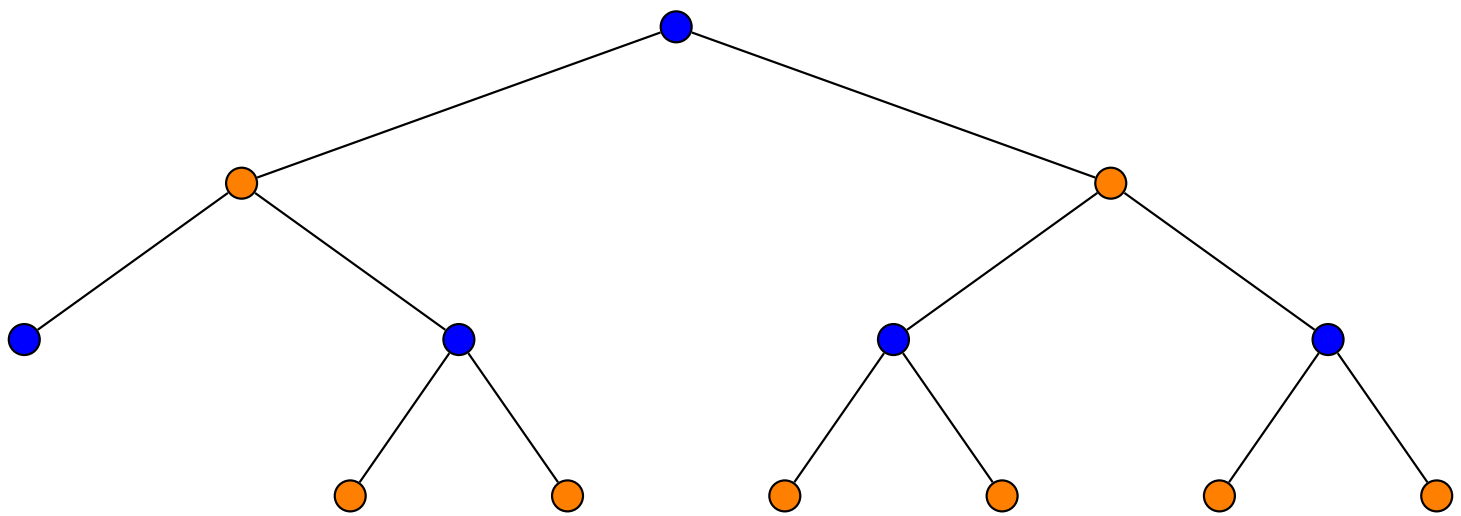
Then a step of the walk consists of applying the operator $R_B R_A$, where $R_A = \bigoplus_{x \in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$.



Quantum walk in a tree

Let A and B be the sets of vertices an even and odd distance from the root, respectively.

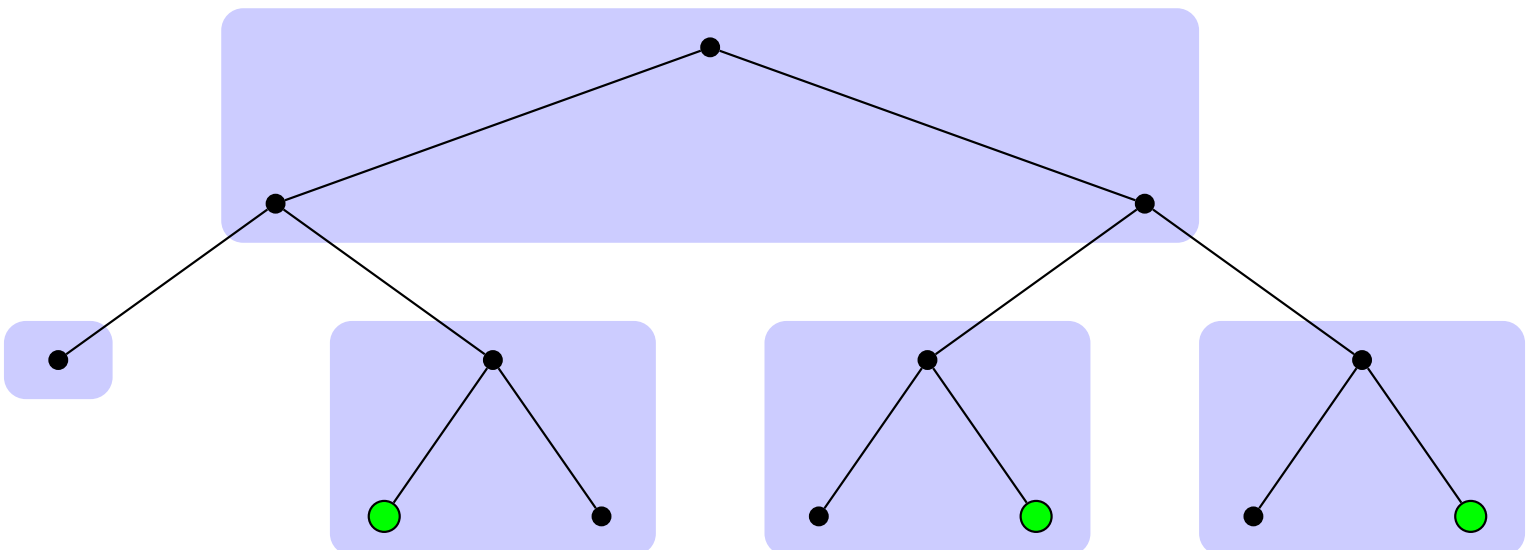
Then a step of the walk consists of applying the operator $R_B R_A$, where $R_A = \bigoplus_{x \in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$.



Quantum walk in a tree

Let A and B be the sets of vertices an even and odd distance from the root, respectively.

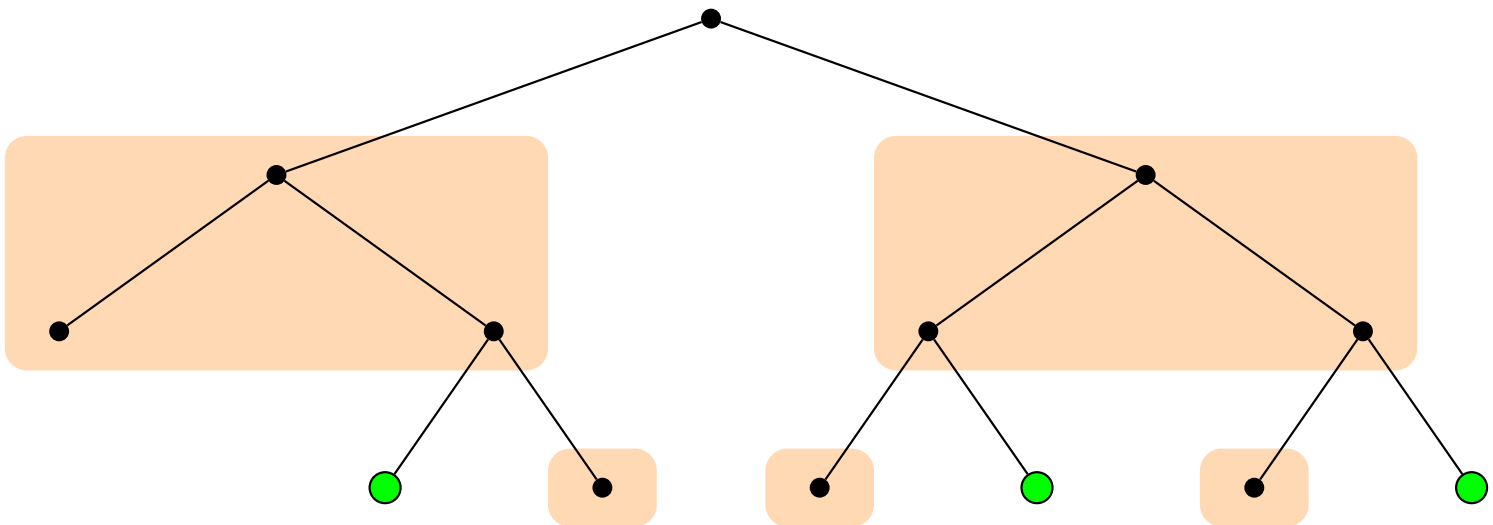
Then a step of the walk consists of applying the operator $R_B R_A$, where $R_A = \bigoplus_{x \in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$.



Quantum walk in a tree

Let A and B be the sets of vertices an even and odd distance from the root, respectively.

Then a step of the walk consists of applying the operator $R_B R_A$, where $R_A = \bigoplus_{x \in A} D_x$ and $R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_x$.



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: $\widetilde{O}(n^{3/2} m^{1/2}/\epsilon + n^2 m)$