Quantum Algorithms - State of the Art

(for high-energy physics and/or gravitational waves?)

Ronald de Wolf

CWI

QuSoft

Universiteit van Amsterdam

Quantum computers

Quantum computers

Two important questions:

Quantum computers

Two important questions:

1. Can we build such a computer?

Quantum computers

Two important questions:

1. Can we build such a computer? We're not there yet. . .

Quantum computers

Two important questions:

1. Can we build such a computer? We're not there yet...
2. What can it do?

Quantum computers

Two important questions:

1. Can we build such a computer? We're not there yet...
2. What can it do? This talk: **quantum algorithms**

Quantum computers

Two important questions:

1. Can we build such a computer? We're not there yet...
2. What can it do? This talk: **quantum algorithms**

These work by interplay of superposition and interference:

Quantum computers

Two important questions:

1. Can we build such a computer? We're not there yet...
2. What can it do? This talk: **quantum algorithms**

These work by interplay of superposition and interference:

1. Start with all qubits in easily-preparable state (e.g. all $|0\rangle$)

Quantum computers

Two important questions:

1. Can we build such a computer? We're not there yet...
2. What can it do? This talk: **quantum algorithms**

These work by interplay of superposition and interference:

1. Start with all qubits in easily-preparable state (e.g. all $|0\rangle$)
2. Set up initial superposition and manipulate it with gates, so that computational paths leading to correct output interfere constructively, others interfere destructively

Quantum computers

Two important questions:

1. Can we build such a computer? We're not there yet...
2. What can it do? This talk: **quantum algorithms**

These work by interplay of superposition and interference:

1. Start with all qubits in easily-preparable state (e.g. all $|0\rangle$)
2. Set up initial superposition and manipulate it with gates, so that computational paths leading to correct output interfere constructively, others interfere destructively (computation is efficient if it uses few gates)

Quantum computers

Two important questions:

1. Can we build such a computer? We're not there yet. . .
2. What can it do? This talk: **quantum algorithms**

These work by interplay of superposition and interference:

1. Start with all qubits in easily-preparable state (e.g. all $|0\rangle$)
2. Set up initial superposition and manipulate it with gates, so that computational paths leading to correct output interfere constructively, others interfere destructively (computation is efficient if it uses few gates)
3. Measurement of final state then gives classical output

Quantum algorithms that might help HEP and/or GW

Quantum algorithms that might help HEP and/or GW

1. Extracting periodicity using the quantum Fourier transform
2. Searching through large spaces
3. Faster optimization
4. Simulating quantum systems

Quantum algorithms that might help HEP and/or GW

1. Extracting periodicity using the quantum Fourier transform
2. Searching through large spaces
3. Faster optimization
4. Simulating quantum systems

I will go over these 4 items at a fairly high level, hoping that something triggers a click with the computational needs of high-energy physics and/or gravitational-wave research

1. Quantum Fourier transform

1. Quantum Fourier transform

- Fourier transforms are key to analysing periodic sequences

1. Quantum Fourier transform

- Fourier transforms are key to analysing periodic sequences in music

1. Quantum Fourier transform

- Fourier transforms are key to analysing periodic sequences in music, for image compression

1. Quantum Fourier transform

- Fourier transforms are key to analysing periodic sequences in music, for image compression, *for gravitational waves?*

1. Quantum Fourier transform

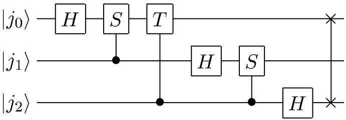
- Fourier transforms are key to analysing periodic sequences in music, for image compression, *for gravitational waves?*
- n-qubit quantum Fourier Transform: $|j\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi ijk}{2^n}} |k\rangle$

  where j and k are n-bit integers

1. Quantum Fourier transform

- Fourier transforms are key to analysing periodic sequences in music, for image compression, *for gravitational waves?*
- n-qubit quantum Fourier Transform: $|j\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi ijk}{2^n}} |k\rangle$

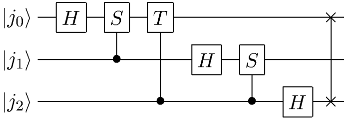
  where j and k are n-bit integers
- 2^n-dimensional unitary, can be implemented with $O(n^2)$ gates



1. Quantum Fourier transform

- Fourier transforms are key to analysing periodic sequences in music, for image compression, *for gravitational waves?*
- n-qubit quantum Fourier Transform: $|j\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi ijk}{2^n}} |k\rangle$

  where j and k are n-bit integers
- 2^n-dimensional unitary, can be implemented with $O(n^2)$ gates

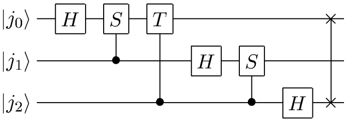


Can reduce to $O(n \log n)$ gates if we allow very small error

1. Quantum Fourier transform

- Fourier transforms are key to analysing periodic sequences in music, for image compression, *for gravitational waves?*
- n-qubit quantum Fourier Transform: $|j\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi ijk}{2^n}} |k\rangle$

  where j and k are n-bit integers
- 2^n-dimensional unitary, can be implemented with $O(n^2)$ gates

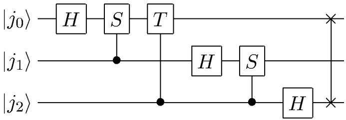


  Can reduce to $O(n \log n)$ gates if we allow very small error
- Key component in many quantum algorithms.

1. Quantum Fourier transform

- Fourier transforms are key to analysing periodic sequences in music, for image compression, *for gravitational waves?*
- n-qubit quantum Fourier Transform: $|j\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi ijk}{2^n}} |k\rangle$

  where j and k are n-bit integers
- 2^n-dimensional unitary, can be implemented with $O(n^2)$ gates



  Can reduce to $O(n \log n)$ gates if we allow very small error
- Key component in many quantum algorithms. Difficulty is how to "load" your periodic sequence as amplitudes of a state

Famous example: Shor's factoring algorithm (1994)

Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

1. Reduction to period-finding (uses classical number theory)

Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

1. Reduction to period-finding (uses classical number theory): choose random integer $c < N$, define $f : \mathbb{N} \to \{0, \ldots, N-1\}$

$$f(x) = c^x \bmod N$$

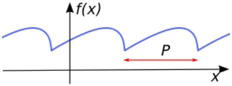
Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

1. Reduction to period-finding (uses classical number theory): choose random integer $c < N$, define $f : \mathbb{N} \to \{0, \ldots, N-1\}$

$$f(x) = c^x \bmod N$$

This is a periodic function



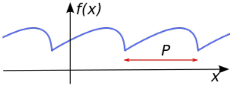
Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

1. Reduction to period-finding (uses classical number theory):
   choose random integer $c < N$, define $f : \mathbb{N} \to \{0, \ldots, N-1\}$

$$f(x) = c^x \bmod N$$

This is a periodic function



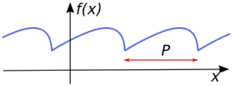
Claim: if you can find period P, then you can factor N

Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

1. Reduction to period-finding (uses classical number theory): choose random integer $c < N$, define $f : \mathbb{N} \to \{0, \ldots, N-1\}$

$$f(x) = c^x \bmod N$$

This is a periodic function 

Claim: if you can find period P, then you can factor N

2. Quantum algorithm for finding the period:

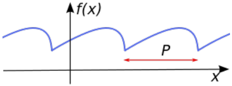
Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

1. Reduction to period-finding (uses classical number theory): choose random integer $c < N$, define $f : \mathbb{N} \to \{0, \ldots, N-1\}$

$$f(x) = c^x \bmod N$$

This is a periodic function



Claim: if you can find period P, then you can factor N

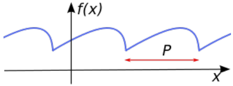
2. Quantum algorithm for finding the period:
   2.1 Generate superposition $\sum_x |x\rangle|0\rangle$

Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

1. Reduction to period-finding (uses classical number theory):
   choose random integer $c < N$, define $f : \mathbb{N} \to \{0, \ldots, N-1\}$

$$f(x) = c^x \bmod N$$

   This is a periodic function 

   Claim: if you can find period P, then you can factor N

2. Quantum algorithm for finding the period:
   2.1 Generate superposition $\sum_x |x\rangle|0\rangle$
   2.2 Compute f in superposition: $\sum_x |x\rangle|f(x)\rangle$

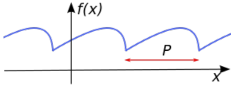
Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

1. Reduction to period-finding (uses classical number theory):
   choose random integer $c < N$, define $f : \mathbb{N} \to \{0, \ldots, N-1\}$

$$f(x) = c^x \bmod N$$

   This is a periodic function



   Claim: if you can find period P, then you can factor N

2. Quantum algorithm for finding the period:
   2.1 Generate superposition $\sum_x |x\rangle|0\rangle$
   2.2 Compute f in superposition: $\sum_x |x\rangle|f(x)\rangle$
   2.3 Measure 2nd register.

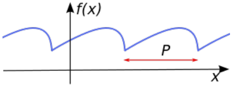
Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

1. Reduction to period-finding (uses classical number theory):
   choose random integer $c < N$, define $f : \mathbb{N} \to \{0, \ldots, N-1\}$

$$f(x) = c^x \bmod N$$

   This is a periodic function



   Claim: if you can find period P, then you can factor N

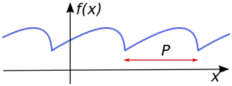
2. Quantum algorithm for finding the period:
   - 2.1 Generate superposition $\sum_x |x\rangle|0\rangle$
   - 2.2 Compute f in superposition: $\sum_x |x\rangle|f(x)\rangle$
   - 2.3 Measure 2nd register. If you see $f(s)$, then 1st register is now in superposition $|s\rangle + |s+P\rangle + |s+2P\rangle + |s+3P\rangle + \cdots$

Famous example: Shor's factoring algorithm (1994)

Goal: efficiently find the prime factors of given integer N

1. Reduction to period-finding (uses classical number theory):
   choose random integer $c < N$, define $f : \mathbb{N} \to \{0, \ldots, N-1\}$

$$f(x) = c^x \bmod N$$

   This is a periodic function 

   Claim: if you can find period P, then you can factor N

2. Quantum algorithm for finding the period:
   2.1 Generate superposition $\sum_x |x\rangle|0\rangle$
   2.2 Compute f in superposition: $\sum_x |x\rangle|f(x)\rangle$
   2.3 Measure 2nd register. If you see $f(s)$, then 1st register is now in superposition $|s\rangle + |s+P\rangle + |s+2P\rangle + |s+3P\rangle + \cdots$
   2.4 Do quantum Fourier transform and measure:
   the result gives information about the period P

2. Grover's quantum search algorithm (1996)

2. Grover's quantum search algorithm (1996)

- If you have an unordered search space with N possible locations, then on average you'll have to inspect $N/2$ of them before you found what you were looking for

2. Grover's quantum search algorithm (1996)

- If you have an unordered search space with N possible locations, then on average you'll have to inspect $N/2$ of them before you found what you were looking for
- Grover's quantum algorithm solves this search problem in $O(\sqrt{N})$ steps

2. Grover's quantum search algorithm (1996)

- If you have an unordered search space with N possible locations, then on average you'll have to inspect $N/2$ of them before you found what you were looking for
- Grover's quantum algorithm solves this search problem in $O(\sqrt{N})$ steps
- Grover finds needle in a haystack much faster than classical search

2. Grover's quantum search algorithm (1996)

- If you have an unordered search space with N possible locations, then on average you'll have to inspect $N/2$ of them before you found what you were looking for
- Grover's quantum algorithm solves this search problem in $O(\sqrt{N})$ steps
- Grover finds needle in a haystack much faster than classical search

- Many applications: basically anything where search appears as a subproblem, and also many estimation problems (e.g., Monte Carlo)

Example application: finding patterns in long strings

Example application: finding patterns in long strings

- Some version of the following might be of interest for analyzing the (massive) data coming out of HEP experiments:

  Given a string $x_1, \ldots, x_N$ of N letters or numbers,
  see if a particular length-M pattern appears somewhere

Example application: finding patterns in long strings

- Some version of the following might be of interest for analyzing the (massive) data coming out of HEP experiments:

  Given a string $x_1, \ldots, x_N$ of N letters or numbers,
  see if a particular length-M pattern appears somewhere

$$\overbrace{a\ n\ o\ t\ h\ e\ r\ \underbrace{w\ o\ l\ f}_{M}\ f\ o\ u\ n\ d\ i\ n\ h\ o\ l\ l\ a\ n\ d}^{N}$$

Example application: finding patterns in long strings

- Some version of the following might be of interest for analyzing the (massive) data coming out of HEP experiments:

  Given a string $x_1, \ldots, x_N$ of N letters or numbers,
  see if a particular length-M pattern appears somewhere

$$\overbrace{a\ n\ o\ t\ h\ e\ r\ \underbrace{w\ o\ l\ f}_{M}\ f\ o\ u\ n\ d\ i\ n\ h\ o\ l\ l\ a\ n\ d}^{N}$$

- Trivial algorithm takes time $O(NM)$

Example application: finding patterns in long strings

- Some version of the following might be of interest for analyzing the (massive) data coming out of HEP experiments:

  Given a string $x_1, \ldots, x_N$ of N letters or numbers, see if a particular length-M pattern appears somewhere

$$\overbrace{a\ n\ o\ t\ h\ e\ r\ \underbrace{w\ o\ l\ f}_{M}\ f\ o\ u\ n\ d\ i\ n\ h\ o\ l\ l\ a\ n\ d}^{N}$$

- Trivial algorithm takes time $O(NM)$
- Knuth-Morris-Pratt classical algorithm: time $O(N + M)$

Example application: finding patterns in long strings

- Some version of the following might be of interest for analyzing the (massive) data coming out of HEP experiments:
  Given a string $x_1, \ldots, x_N$ of N letters or numbers, see if a particular length-M pattern appears somewhere

$$\overbrace{a\ n\ o\ t\ h\ e\ r\ \underbrace{w\ o\ l\ f}_{M}\ f\ o\ u\ n\ d\ i\ n\ h\ o\ l\ l\ a\ n\ d}^{N}$$

- Trivial algorithm takes time $O(NM)$
- Knuth-Morris-Pratt classical algorithm: time $O(N+M)$
- Ramesh & Vinay'03: quantum algorithm with time $O(\sqrt{N}+\sqrt{M})$

Example application: finding patterns in long strings

- Some version of the following might be of interest for analyzing the (massive) data coming out of HEP experiments:

  Given a string $x_1, \ldots, x_N$ of N letters or numbers,
  see if a particular length-M pattern appears somewhere

$$\overbrace{a\ n\ o\ t\ h\ e\ r\ \underbrace{w\ o\ l\ f}_{M}\ f\ o\ u\ n\ d\ i\ n\ h\ o\ l\ l\ a\ n\ d}^{N}$$

- Trivial algorithm takes time $O(NM)$
- Knuth-Morris-Pratt classical algorithm: time $O(N+M)$
- Ramesh & Vinay'03: quantum algorithm with time $O(\sqrt{N}+\sqrt{M})$ using Grover as a subroutine

Example application: finding patterns in long strings

- Some version of the following might be of interest for analyzing the (massive) data coming out of HEP experiments:
  Given a string $x_1, \ldots, x_N$ of N letters or numbers, see if a particular length-M pattern appears somewhere

$$\overbrace{a\ n\ o\ t\ h\ e\ r\ \underbrace{w\ o\ l\ f}_{M}\ f\ o\ u\ n\ d\ i\ n\ h\ o\ l\ l\ a\ n\ d}^{N}$$

- Trivial algorithm takes time $O(NM)$
- Knuth-Morris-Pratt classical algorithm: time $O(N + M)$
- Ramesh & Vinay'03: quantum algorithm with time $O(\sqrt{N} + \sqrt{M})$ using Grover as a subroutine
- Montanaro'14: exponential speed-up on average in higher dimensions (Grover + algorithm for finding "hidden shifts")

Two big caveats

Two big caveats

1. Grover-based speedups are $\leq$ quadratic. Is this any good?

Two big caveats

1. Grover-based speedups are $\leq$ quadratic. Is this any good?
   Compare quantum cost $C\sqrt{N}$ vs classical cost cN:

Two big caveats

1. Grover-based speedups are $\leq$ quadratic. Is this any good?
   Compare quantum cost $C\sqrt{N}$ vs classical cost cN:
   quantum beats classical for instance-size $N > (C/c)^2$.

Two big caveats

1. Grover-based speedups are $\leq$ quadratic. Is this any good?
   Compare quantum cost $C\sqrt{N}$ vs classical cost cN:
   quantum beats classical for instance-size $N > (C/c)^2$.
   If $C/c \sim 10^{10}$, then need huge $N > 10^{20}$ before get speed-up

Two big caveats

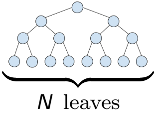
1. Grover-based speedups are $\leq$ quadratic. Is this any good?
   Compare quantum cost $C\sqrt{N}$ vs classical cost cN:
   quantum beats classical for instance-size $N > (C/c)^2$.
   If $C/c \sim 10^{10}$, then need huge $N > 10^{20}$ before get speed-up

2. If we are given classical data (eg, sequence of numbers, or input graph) we should be able to access this in superposition.

Two big caveats

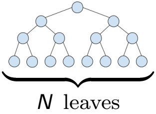
1. Grover-based speedups are $\leq$ quadratic. Is this any good?
   Compare quantum cost $C\sqrt{N}$ vs classical cost cN:
   quantum beats classical for instance-size $N > (C/c)^2$.
   If $C/c \sim 10^{10}$, then need huge $N > 10^{20}$ before get speed-up

2. If we are given classical data (eg, sequence of numbers, or input graph) we should be able to access this in superposition.
   Classical N-bit RAM is a piece of hardware of size $\sim N$ that can be accessed in $\sim \log N$ steps



Two big caveats

1. Grover-based speedups are $\leq$ quadratic. Is this any good?
   Compare quantum cost $C\sqrt{N}$ vs classical cost cN:
   quantum beats classical for instance-size $N > (C/c)^2$.
   If $C/c \sim 10^{10}$, then need huge $N > 10^{20}$ before get speed-up

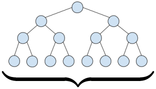
2. If we are given classical data (eg, sequence of numbers, or input graph) we should be able to access this in superposition.
   Classical N-bit RAM is a piece of hardware of size $\sim N$ that can be accessed in $\sim \log N$ steps
   

   Quantum RAM should be the same ($|i,0\rangle \mapsto |i,x_i\rangle$) accessible in superposition.

Two big caveats

1. Grover-based speedups are $\leq$ quadratic. Is this any good?
   Compare quantum cost $C\sqrt{N}$ vs classical cost cN:
   quantum beats classical for instance-size $N > (C/c)^2$.
   If $C/c \sim 10^{10}$, then need huge $N > 10^{20}$ before get speed-up

2. If we are given classical data (eg, sequence of numbers, or input graph) we should be able to access this in superposition.
   Classical N-bit RAM is a piece of hardware of size $\sim N$ that can be accessed in $\sim \log N$ steps

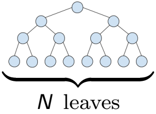
   

   N leaves

   Quantum RAM should be the same ($|i, 0\rangle \mapsto |i, x_i\rangle$) accessible in superposition. Hard to implement with noise.

Two big caveats

1. Grover-based speedups are $\leq$ quadratic. Is this any good?
   Compare quantum cost $C\sqrt{N}$ vs classical cost cN:
   quantum beats classical for instance-size $N > (C/c)^2$.
   If $C/c \sim 10^{10}$, then need huge $N > 10^{20}$ before get speed-up

2. If we are given classical data (eg, sequence of numbers, or input graph) we should be able to access this in superposition.
   Classical N-bit RAM is a piece of hardware of size $\sim N$ that can be accessed in $\sim \log N$ steps
   

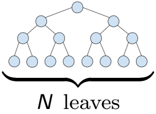
   Quantum RAM should be the same ($|i,0\rangle \mapsto |i,x_i\rangle$) accessible in superposition. Hard to implement with noise.

Data of HEP/GW experiments is probably too big to fit in a RAM anyway, so we'll need some sort of "streaming" model.

Two big caveats

1. Grover-based speedups are $\leq$ quadratic. Is this any good?
   Compare quantum cost $C\sqrt{N}$ vs classical cost cN:
   quantum beats classical for instance-size $N > (C/c)^2$.
   If $C/c \sim 10^{10}$, then need huge $N > 10^{20}$ before get speed-up

2. If we are given classical data (eg, sequence of numbers, or input graph) we should be able to access this in superposition.
   Classical N-bit RAM is a piece of hardware of size $\sim N$ that can be accessed in $\sim \log N$ steps
   Quantum RAM should be the same ($|i, 0\rangle \mapsto |i, x_i\rangle$) accessible in superposition. Hard to implement with noise.



Data of HEP/GW experiments is probably too big to fit in a RAM anyway, so we'll need some sort of "streaming" model.

Grover-based speedups are probably not for the near term

3. Optimization

3. Optimization

- Optimization is one of the main applications of computers in the real world: allocate resources to jobs, optimize designs, minimize energy use, machine learning (fit model to data) etc.

3. Optimization

- Optimization is one of the main applications of computers in the real world: allocate resources to jobs, optimize designs, minimize energy use, machine learning (fit model to data) etc.

$$\min_{x \in K} f(x)$$

3. Optimization

- Optimization is one of the main applications of computers in the real world: allocate resources to jobs, optimize designs, minimize energy use, machine learning (fit model to data) etc.

$$\min_{x \in K} f(x),$$

think of $x \in \mathbb{R}^n$ with some constraints

3. Optimization

- Optimization is one of the main applications of computers in the real world: allocate resources to jobs, optimize designs, minimize energy use, machine learning (fit model to data) etc.

$$\min_{x \in K} f(x),$$

  think of $x \in \mathbb{R}^n$ with some constraints

- Continuous optimization: variables are reals

3. Optimization

- Optimization is one of the main applications of computers in the real world: allocate resources to jobs, optimize designs, minimize energy use, machine learning (fit model to data) etc.

$$\min_{x \in K} f(x),$$

  think of $x \in \mathbb{R}^n$ with some constraints

- Continuous optimization: variables are reals

  Discrete optimization: variables are bits, integers

3. Optimization

- Optimization is one of the main applications of computers in the real world: allocate resources to jobs, optimize designs, minimize energy use, machine learning (fit model to data) etc.

$$\min_{x \in K} f(x),$$

think of $x \in \mathbb{R}^n$ with some constraints

- Continuous optimization: variables are reals

  Discrete optimization: variables are bits, integers

  Or a mix of these

3. Optimization

- Optimization is one of the main applications of computers in the real world: allocate resources to jobs, optimize designs, minimize energy use, machine learning (fit model to data) etc.

  $$\min_{x \in K} f(x),$$

  think of $x \in \mathbb{R}^n$ with some constraints

- Continuous optimization: variables are reals

  Discrete optimization: variables are bits, integers

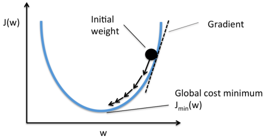
  Or a mix of these

- Quantum computers can help (sometimes)

Quantum speed-ups for **continuous** optimization

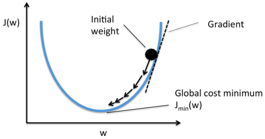
Quantum speed-ups for **continuous** optimization

- Gradient descent: iterative method to find local minimum of $f : \mathbb{R}^n \to \mathbb{R}$

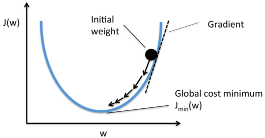


Quantum speed-ups for **continuous** optimization

- Gradient descent: iterative method to find local minimum of $f : \mathbb{R}^n \to \mathbb{R}$
  1. Start with $t = 0$, and some initial point $x^{(0)}$

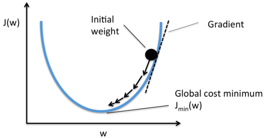


Quantum speed-ups for **continuous** optimization



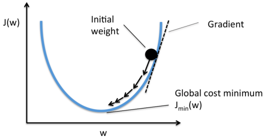
- Gradient descent: iterative method to find local minimum of $f : \mathbb{R}^n \to \mathbb{R}$
  1. Start with $t = 0$, and some initial point $x^{(0)}$
  2. Compute the gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$ at point $x^{(t)}$

Quantum speed-ups for **continuous** optimization



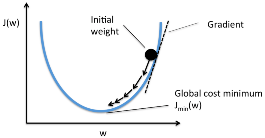
- Gradient descent: iterative method to find local minimum of $f : \mathbb{R}^n \to \mathbb{R}$
  1. Start with $t = 0$, and some initial point $x^{(0)}$
  2. Compute the gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$ at point $x^{(t)}$
  3. Move down for some stepsize η: $x^{(t+1)} \leftarrow x^{(t)} - \eta \cdot \nabla f(x^{(t)})$

Quantum speed-ups for **continuous** optimization



- Gradient descent: iterative method to find local minimum of $f : \mathbb{R}^n \to \mathbb{R}$
  1. Start with $t = 0$, and some initial point $x^{(0)}$
  2. Compute the gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$ at point $x^{(t)}$
  3. Move down for some stepsize η: $x^{(t+1)} \leftarrow x^{(t)} - \eta \cdot \nabla f(x^{(t)})$
  4. Set $t \leftarrow t + 1$, goto 2

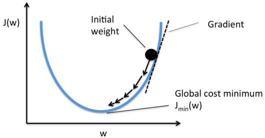
Quantum speed-ups for **continuous** optimization



- Gradient descent: iterative method to find local minimum of $f : \mathbb{R}^n \to \mathbb{R}$
  1. Start with $t = 0$, and some initial point $x^{(0)}$
  2. Compute the gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$ at point $x^{(t)}$
  3. Move down for some stepsize η: $x^{(t+1)} \leftarrow x^{(t)} - \eta \cdot \nabla f(x^{(t)})$
  4. Set $t \leftarrow t + 1$, goto 2

  QCs can sometimes compute the gradient more efficiently

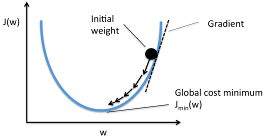
Quantum speed-ups for **continuous** optimization



- Gradient descent: iterative method to find local minimum of $f : \mathbb{R}^n \to \mathbb{R}$
  1. Start with $t = 0$, and some initial point $x^{(0)}$
  2. Compute the gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$ at point $x^{(t)}$
  3. Move down for some stepsize η: $x^{(t+1)} \leftarrow x^{(t)} - \eta \cdot \nabla f(x^{(t)})$
  4. Set $t \leftarrow t + 1$, goto 2

  QCs can sometimes compute the gradient more efficiently
- Polynomial speed-up for linear programs with n real variables, m linear constraints: $\max \; c^T x$ s.t. $Ax \leq b$

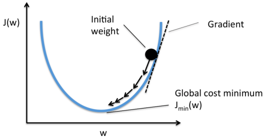
Quantum speed-ups for **continuous** optimization



- Gradient descent: iterative method to find local minimum of $f : \mathbb{R}^n \to \mathbb{R}$
  1. Start with $t = 0$, and some initial point $x^{(0)}$
  2. Compute the gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$ at point $x^{(t)}$
  3. Move down for some stepsize η: $x^{(t+1)} \leftarrow x^{(t)} - \eta \cdot \nabla f(x^{(t)})$
  4. Set $t \leftarrow t + 1$, goto 2

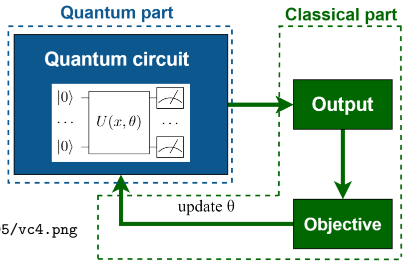
  QCs can sometimes compute the gradient more efficiently
- Polynomial speed-up for linear programs with n real variables, m linear constraints: $\max \; c^T x$ s.t. $Ax \leq b$
- Variational methods: use classical methods to optimize over some parametrized circuits

Quantum speed-ups for **continuous** optimization



- Gradient descent: iterative method to find local minimum of $f : \mathbb{R}^n \to \mathbb{R}$
  1. Start with $t = 0$, and some initial point $x^{(0)}$
  2. Compute the gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$ at point $x^{(t)}$
  3. Move down for some stepsize η: $x^{(t+1)} \leftarrow x^{(t)} - \eta \cdot \nabla f(x^{(t)})$
  4. Set $t \leftarrow t + 1$, goto 2

  QCs can sometimes compute the gradient more efficiently
- Polynomial speed-up for linear programs with n real variables, m linear constraints: $\max \; c^T x$ s.t. $Ax \leq b$
- Variational methods: use classical methods to optimize over some parametrized circuits



https://dkopczyk.quantee.co.uk/wp-content/uploads/2019/05/vc4.png

Quantum speed-ups for **discrete** optimization

Quantum speed-ups for **discrete** optimization

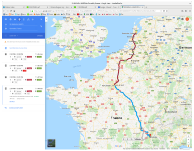
- Find the minimum of $f : \{1, \ldots, N\} \to \mathbb{R}$

Quantum speed-ups for **discrete** optimization

- Find the minimum of $f : \{1, \ldots, N\} \to \mathbb{R}$
  in $O(\sqrt{N})$ f-evaluations and other operations (Dürr-Høyer'96)

Quantum speed-ups for **discrete** optimization

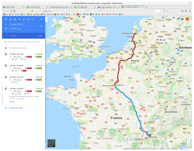
- Find the minimum of $f : \{1, \ldots, N\} \to \mathbb{R}$
  in $O(\sqrt{N})$ f-evaluations and other operations (Dürr-Høyer'96)
- Speed-up for finding shortest path in a graph



Quantum speed-ups for **discrete** optimization

- Find the minimum of $f : \{1, \ldots, N\} \to \mathbb{R}$
  in $O(\sqrt{N})$ f-evaluations and other operations (Dürr-Høyer'96)

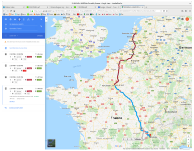
- Speed-up for finding shortest path in a graph

  for graph sparsification,
  many other graph problems

Quantum speed-ups for **discrete** optimization

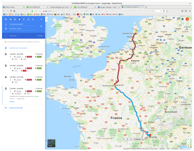
- Find the minimum of $f : \{1, \ldots, N\} \to \mathbb{R}$ in $O(\sqrt{N})$ f-evaluations and other operations (Dürr-Høyer'96)
- Speed-up for finding shortest path in a graph

  for graph sparsification,
  many other graph problems
- Many discrete optimization problems are NP-hard: Constraint-satisfaction, Traveling Salesman Problem, finding largest clique in a graph, integer linear programs, minimal energy of protein folding, of spin glasses . . .

Quantum speed-ups for **discrete** optimization

- Find the minimum of $f : \{1, \ldots, N\} \to \mathbb{R}$ in $O(\sqrt{N})$ f-evaluations and other operations (Dürr-Høyer'96)
- Speed-up for finding shortest path in a graph

  for graph sparsification,
  many other graph problems
- Many discrete optimization problems are NP-hard:
  Constraint-satisfaction, Traveling Salesman Problem,
  finding largest clique in a graph, integer linear programs,
  minimal energy of protein folding, of spin glasses ...

  We expect no more than quadratic quantum speed-up for NP-hard problems, so their complexity remains exponential

4. Simulating quantum systems

4. Simulating quantum systems

- Quantum computers can simulate the dynamics of a quantum system given by Hamiltonian H and initial state $|\psi(0)\rangle$

4. Simulating quantum systems

- Quantum computers can simulate the dynamics of a quantum system given by Hamiltonian H and initial state $|\psi(0)\rangle$:

$$|\psi(t)\rangle = \underbrace{e^{-iHt}}_{U} |\psi(0)\rangle$$

4. Simulating quantum systems

- Quantum computers can simulate the dynamics of a quantum system given by Hamiltonian H and initial state $|\psi(0)\rangle$:

$$|\psi(t)\rangle = \underbrace{e^{-iHt}}_{U} |\psi(0)\rangle$$

- Under reasonable assumptions on H (eg local), there is a quantum circuit for U with $O(t)$ gates and small error, so quantum computer can efficiently evolve state $|\psi(0)\rangle$ to $|\psi(t)\rangle$

4. Simulating quantum systems

- Quantum computers can simulate the dynamics of a quantum system given by Hamiltonian H and initial state $|\psi(0)\rangle$:

$$|\psi(t)\rangle = \underbrace{e^{-iHt}}_{U} |\psi(0)\rangle$$

- Under reasonable assumptions on H (eg local), there is a quantum circuit for U with $O(t)$ gates and small error, so quantum computer can efficiently evolve state $|\psi(0)\rangle$ to $|\psi(t)\rangle$
- Simulating dynamics replaces lab experiments by a computer; may help material science, quantum chemistry, drug design...

4. Simulating quantum systems

- Quantum computers can simulate the dynamics of a quantum system given by Hamiltonian H and initial state $|\psi(0)\rangle$:

$$|\psi(t)\rangle = \underbrace{e^{-iHt}}_{U} |\psi(0)\rangle$$

- Under reasonable assumptions on H (eg local), there is a quantum circuit for U with $O(t)$ gates and small error, so quantum computer can efficiently evolve state $|\psi(0)\rangle$ to $|\psi(t)\rangle$
- Simulating dynamics replaces lab experiments by a computer; may help material science, quantum chemistry, drug design. . .
- This is plain-vanilla quantum mechanics, but you can also efficiently simulate dynamics of some quantum field theories (eg Jordan, Lee, Preskill'11-'14; see survey Preskill'18)

4. Simulating quantum systems

- Quantum computers can simulate the dynamics of a quantum system given by Hamiltonian H and initial state $|\psi(0)\rangle$:

$$|\psi(t)\rangle = \underbrace{e^{-iHt}}_{U}|\psi(0)\rangle$$

- Under reasonable assumptions on H (eg local), there is a quantum circuit for U with $O(t)$ gates and small error, so quantum computer can efficiently evolve state $|\psi(0)\rangle$ to $|\psi(t)\rangle$
- Simulating dynamics replaces lab experiments by a computer; may help material science, quantum chemistry, drug design...
- This is plain-vanilla quantum mechanics, but you can also efficiently simulate dynamics of some quantum field theories (eg Jordan, Lee, Preskill'11-'14; see survey Preskill'18)
- NB: Finding ground state energy of H is *much* harder problem

What quantum computers **cannot** do

What quantum computers **cannot** do

- A classical computer can simulate a quantum computer with exponential slowdown.

What quantum computers **cannot** do

- A classical computer can simulate a quantum computer with exponential slowdown.

  This implies that the set of *computable* problems doesn't change: "Church-Turing thesis" remains intact

What quantum computers **cannot** do

- A classical computer can simulate a quantum computer with exponential slowdown.

  This implies that the set of *computable* problems doesn't change: "Church-Turing thesis" remains intact

- For many problems we can show that quantum computers give no significant speed-up

What quantum computers **cannot** do

- A classical computer can simulate a quantum computer with exponential slowdown.

  This implies that the set of *computable* problems doesn't change: "Church-Turing thesis" remains intact

- For many problems we can show that quantum computers give no significant speed-up

  or at most a quadratic speed-up (Grover is provably optimal)

What quantum computers **cannot** do

- A classical computer can simulate a quantum computer with exponential slowdown.

  This implies that the set of *computable* problems doesn't change: "Church-Turing thesis" remains intact

- For many problems we can show that quantum computers give no significant speed-up

  or at most a quadratic speed-up (Grover is provably optimal)

- Conjecture: quantum computers can't efficiently solve NP-hard problems

Summary

Summary

- Quantum algorithms work by combining superposition and interference effects

Summary

- Quantum algorithms work by
  combining superposition and interference effects
- This is weaker (and more subtle) than classical parallelism, but much faster than classical computers for some computational problems

Summary

- Quantum algorithms work by
  combining superposition and interference effects
- This is weaker (and more subtle) than classical parallelism, but much faster than classical computers for some computational problems
- I tried to survey the state of the art in quantum algorithms that might be useful for problems in high-energy physics and/or gravitational waves

Summary

- Quantum algorithms work by combining superposition and interference effects
- This is weaker (and more subtle) than classical parallelism, but much faster than classical computers for some computational problems
- I tried to survey the state of the art in quantum algorithms that might be useful for problems in high-energy physics and/or gravitational waves:
  1. Extracting periodicity using the quantum Fourier transform
  2. Searching through large spaces
  3. Faster optimization
  4. Simulating quantum systems