Quantum Algorithms for Optimization

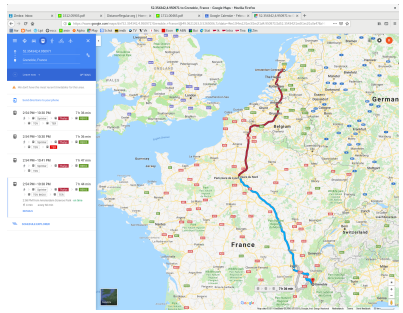
Ronald de Wolf

CWI QuSoft Universiteit van Amsterdam

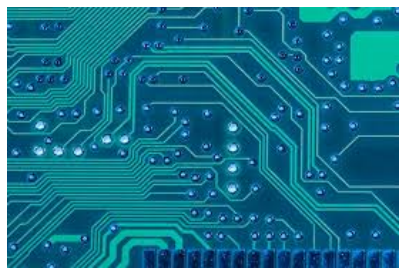
Optimization

General problem: $\min_{x \in K} f(x)$

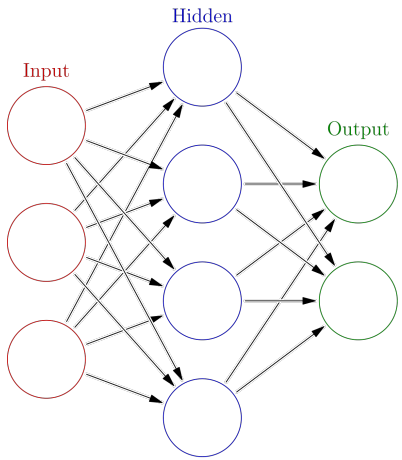
- Finding the shortest route on a given map



- Designing a more energy-efficient chip



- Training your neural network to detect cats



Discrete and continuous settings for optimization

Discrete optimization: variables are discrete (bits, integers)

- Shortest path algorithms
- Matching algorithms
- Max flow / Min cut in a network
- Often discrete optimization problems are NP-hard: Constraint-satisfaction, TSP, integer linear programs ...

Continuous optimization: variables are continuous (reals)

- Gradient descent
- Linear programs, semidefinite programs
- Non-convex optimization

Or a mix of these:

- LP/SDP-relaxation of discrete problems (max-cut in a graph)
- Graph sparsification to solve Laplacian linear system $Lx = b$

How can quantum computers help?

- Faster optimization is one of the main potential application areas of quantum computers (along with crypto & simulation)
- Several new quantum algorithms discovered in last 5-10 years
- Goal of this talk: **survey what we know**
- I'll focus on what I'm most familiar with: provable speed-ups.

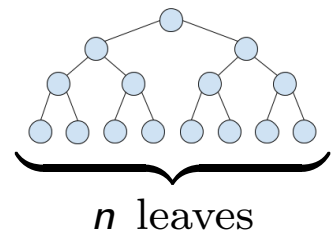
  Not on heuristics: adiabatic algorithm, annealing, variational algorithms like QAOA (Farhi-Goldstone-Gutmann'14):

  Low-depth quantum circuit parametrized by few parameters.
  Run it, measure the output, adjust parameters to improve.
  Hope to do something useful

Two big caveats for the provable speed-ups

1. Most optimization speed-ups $\leq$ quadratic. Is this any good?
   Compare quantum cost $C\sqrt{n}$ vs classical cost n:
   quantum beats classical for instance size $n > C^2$.
   If $C \sim 10^{10}$, then need huge $n > 10^{20}$ before get speed-up

2. If we are given classical data (eg, input graph, or constraint matrix) 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, accessible
   in superposition, $|i, 0\rangle \mapsto |i, x_i\rangle$
   Hard to implement with noise

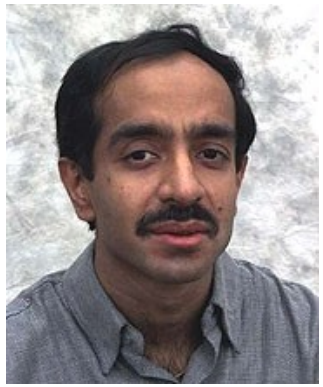


Does this mean these quantum optimization algorithms are useless?
No, but they're probably not for the near term

Discrete optimization

Quantum speed-up for discrete optimization

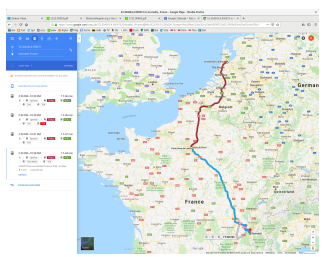
- These typically use Grover's quantum search as a blackbox within larger, often re-designed classical algorithm. Grover finds a solution in a size-n search space in time $O(\sqrt{n})$



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

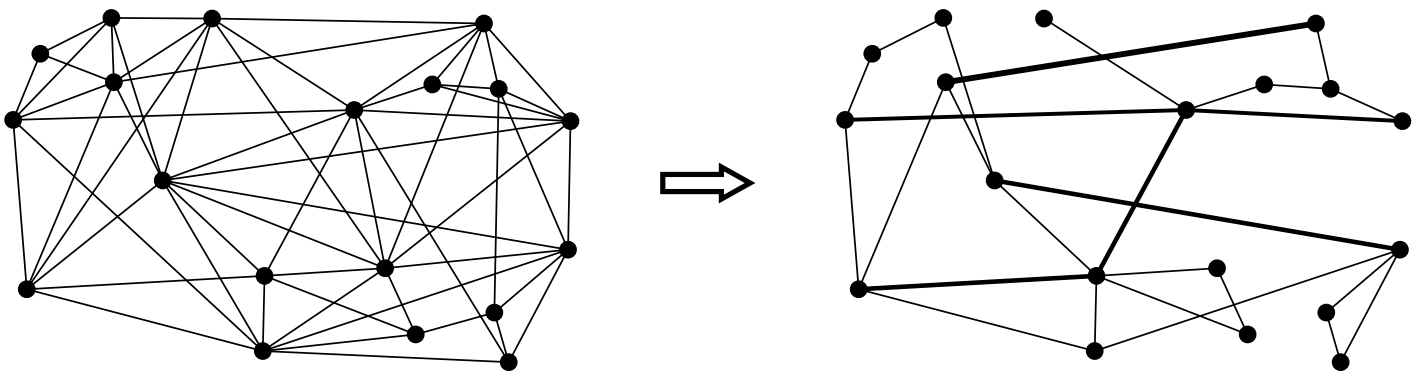
- Finding shortest path in an n-vertex graph

  classical complexity of $O(n^2)$ (Dijkstra'56)
  vs quantum complexity $O(n^{1.5})$ (DHHM'04)

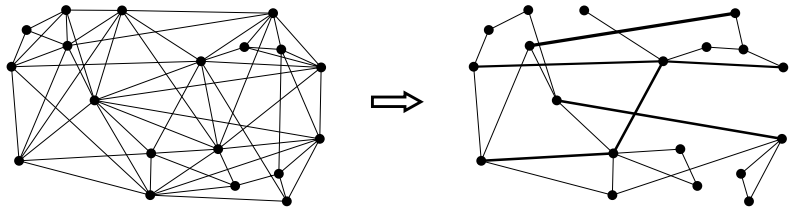


- Polynomial speed-ups for matching, other graph problems

Sparsification: less is more!



Graph sparsification



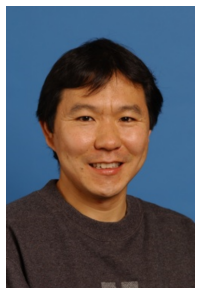
- Graph $G = (V, E, w)$ with $n = |V|$ vertices, $m = |E|$ edges, weight function $w : E \to \mathbb{R}_{\geq 0}$. Given as adjacency list
- Goal: sparsify (ie reduce m) while preserving many properties
- Laplacian $L_G = \sum_{e \in E} w(e) L_e$; $\quad L_{(i,j)} = (e_i - e_j)(e_i - e_j)^T$
- An ε-spectral sparsifier of G is graph $H = (V, E' \subseteq E, w')$ s.t.

$$\text{for all } x \in \mathbb{R}^n : x^T L_G x = (1 \pm \varepsilon) x^T L_H x$$

Example: expander graph is sparsification of complete graph

- $O(n/\varepsilon^2)$ edges suffice for H, we can find H in time $\tilde{O}(m)$
- Many applications, incl. approximate min cut and max cut, Laplacian systems.

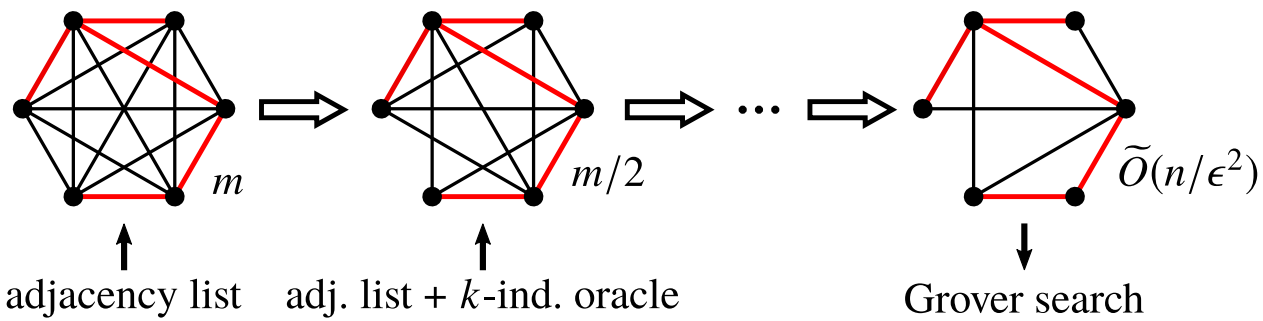
Gödel Prize 2015 for Spielman and Teng



Faster quantum algorithm for sparsification

- Apers-dW'20: quantum algorithm to find ε-spectral sparsifier H in sublinear time $\tilde{O}(\sqrt{mn}/\varepsilon)$ (this is optimal!)
- Similar speed-up for cut problems, Laplacian systems etc.
- Koutis-Xu'16 iterative sparsifier: each iteration identifies $\tilde{O}(n)$ important edges (by finding a few "spanners") and randomly removes half of the other edges; log iterations suffice
- Quantum speed-up using two tools:

  find spanners in time $\tilde{O}(\sqrt{mn})$ instead of classical $\tilde{O}(m)$, and find the final set of $\tilde{O}(n/\varepsilon^2)$ edges of H using Grover



Quantum speed-up for NP-hard optimization problems

Polynomial speed-ups are possible:

- Find a satisfying assignment to a formula ϕ on n Boolean variables $x_1, \ldots, x_n$ in $\sim \sqrt{2^n}$ steps using Grover
- If ϕ is a 3-SAT formula, then plain Grover is slower than classical Schöning algorithm, which takes time $\sim (4/3)^n$. Quadratic speed-up of Schöning via amplitude amplification

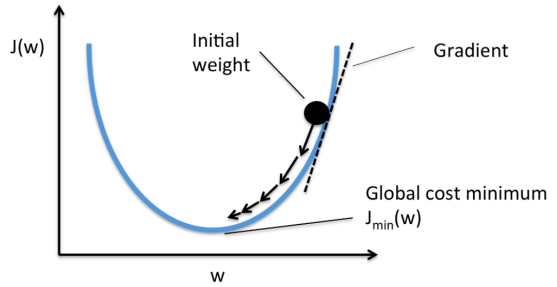
Two other methods for quantum speed-up:

- Montanaro'15: quadratic speed-up for backtracking
- ABIKPV'18: polynomial speed-ups for dynamic programming, incl. speeding up Held-Karp algorithm for TSP from 2^n to 1.7^n

We don't expect exponential quantum speed-up. Viewing SAT as unstructured search, quadratic speed-up is optimal (BBBV'93); we don't know how to use the structure...

Continuous optimization

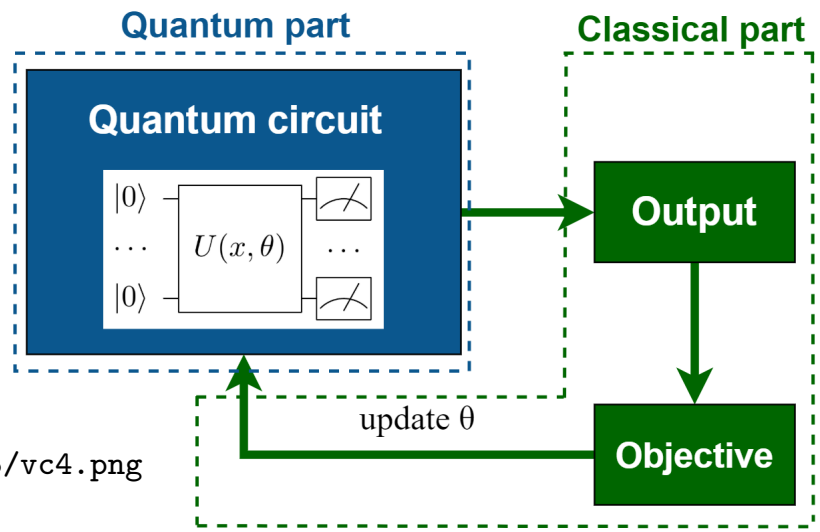
Quantum speed-ups for **continuous** optimization



- 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

- Variational methods: use classical methods to optimize over some parametrized quantum circuits (heuristic)



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

Linear and semidefinite programs

- Linear program, with variable $x \in \mathbb{R}^n$:

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & a_j^T x \le b_j, \quad j = 1, \ldots, m \\ & x \ge 0 \end{array}$$

- Important tool in optimization since 1940s (simplex method), polynomial-time solvable by ellipsoid method (Khachiyan'79)
- Semidefinite program, with variable $X \in \mathbb{R}^{n \times n}$:

$$\begin{array}{ll} \max & \mathrm{Tr}(CX) \\ \text{s.t.} & \mathrm{Tr}(A_j X) \le b_j \\ & X \succeq 0 \\ & \mathrm{Tr}(X) \le R \end{array} \quad \Leftrightarrow \quad \begin{array}{ll} \min & b^T y + R y_0 \\ \text{s.t.} & \sum_{j=1}^m A_j y_j + I y_0 \succeq C \\ & y \ge 0 \end{array}$$

- Can be stronger than LP, e.g. for approximating maximal cut in graph (GW'94). OPT still efficiently approximable to $\pm\varepsilon$

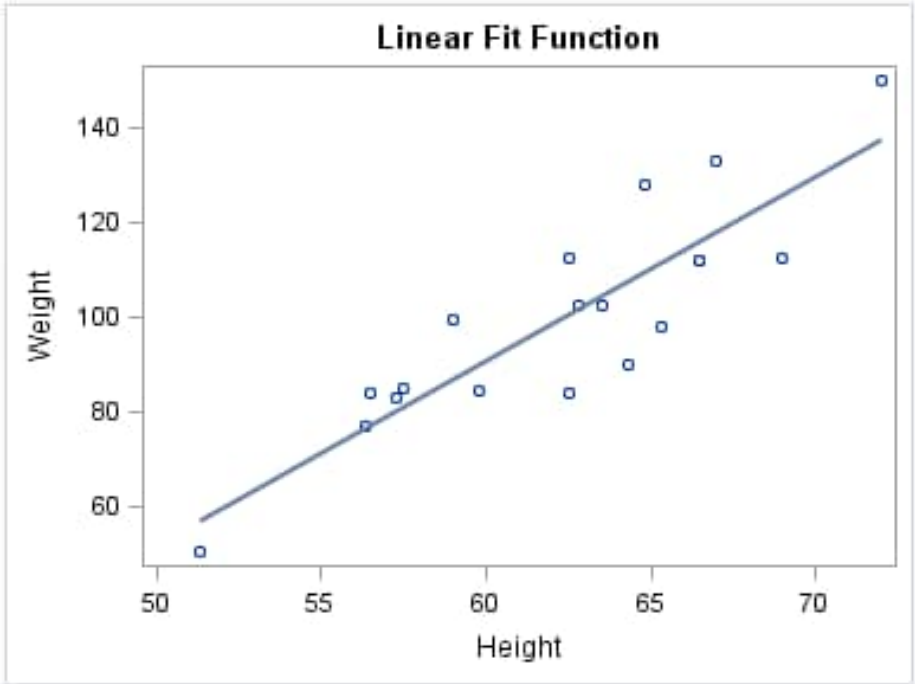
The Arora-Kale SDP-solver & its quantization

- For not-too-small ε, best classical solver is matrix multiplicative weights method (Arora-Kale'05)
- This iterative algorithm bounces back-and-forth between primal solutions $X^{(t)}$ (psd matrices with bounded trace) and solutions $y^{(t)}$ of a relaxation of the dual
- Converges to ε-optimal $X^{(T)}$ for $T = O(\log(n)/\varepsilon^2)$ iterations
- Brandão-Svore'16: treat $X^{(t)}$ as $\log(n)$-qubit quantum state; prepare it as a "Gibbs state", use it to approximate $\mathrm{Tr}(X^{(t)}A_j)$

  Use Grover to solve relaxed dual more efficiently
- This was improved by [vAGGdW'17], [BKLLWS'17] vApeldoorn-Gilyén'18: s-sparse SDPs in $\tilde{O}((\sqrt{m}/\varepsilon^4 + \sqrt{n}/\varepsilon^5)s)$
- For small ε, interior-point methods are best, at least in theory. Recent quantum speed-ups for this as well.

Regularized linear regression

- Given N points $(x_1, y_1), \ldots, (x_N, y_N)$ with $x_i \in \mathbb{R}^d, y_i \in \mathbb{R}$, fit line through them: find coefficient-vector $\theta \in \mathbb{R}^d$ s.t. linear function $x_i^T \theta$ is a good predictor of y-variable



- Find θ to minimize least squares loss $L(\theta) = \sum_{i=1}^{N} (x_i^T \theta - y_i)^2$.

  Closed-form solution for the minimizer: $\theta^* = (X^T X)^+ X^T y$

- Problems: this tends to overfit and yield very dense θ-vectors

- Lasso adds "ℓ_1-regularizer": $\min L(\theta)$ subject to $\sum_{j=1}^{d} |\theta_j| \leq 1$

Quantum algorithm for Lasso

- Lasso: minimize $L(\theta)$ subject to $\sum_{j=1}^{d} |\theta_j| \leq 1$
- Finding the exact minimizer is a hard problem, so we typically try to find a vector θ whose loss is not much worse:

$$L(\theta) \leq L_{\min} + \varepsilon \quad \text{and} \quad \sum_{j=1}^{d} |\theta_j| \leq 1$$

- Best known classical algorithm runs in time $\tilde{O}(d/\varepsilon^2)$
- Chen & dW'21: gave a quantum algorithm that runs in time $\tilde{O}\left(\sqrt{d}/\varepsilon^2\right)$, by speeding up the Frank-Wolfe algorithm.
  In each iteration, FW selects largest entry of gradient $\nabla L(\theta)$ to determine where to move. We can quantumly speed up approximation of gradient-entries as well as maximum-finding
- We also proved $\sqrt{d}/\varepsilon^{1.5}$ lower bound for all quantum algorithms; the correct ε-dependence is still unknown!

Finding the principal eigenvectors of a matrix

- Given $d \times d$ matrix A. Its most important property is the largest eigenvalue λ_1 with associated "top eigenvector" v_1
- Efficiently computing v_1 is important in many applications: Google's Pagerank, Principal Component Analysis, all sorts of optimization algorithms...
- Can compute v_1 by diagonalizing A; this costs time $O(d^{2.37\ldots})$ in theory and $O(d^3)$ in practice. It also does too much...
- Power method is more efficient:
  1. Choose random unit vector $w = \sum_{i=1}^{d} a_i v_i$, $|a_i| \approx 1/\sqrt{d}$
  2. Compute $A^k w = \sum_i a_i \lambda_i^k v_i$.

  If λ_1 is bigger than the other eigenvalues, then $A^k w$ will converge quickly to a vector proportional to v_1
- So we can find v_1 with a few matrix-vector multiplications

Faster power method by faster matrix-vector multiplication

- In general, computing Aw on a quantum computer takes time $\sim d^2$ if you want to do it exactly. But power method still works if the matrix-vector products are computed with small and benign errors! (Moritz & Hardt'14)
- But even approximating Aw takes $\sim d^2$ steps classically
- Chen, Gilyén, dW'24 approximate Aw in time $\sim d^{1.5}$: approximately prepare Aw as a $\log(d)$-qubit quantum state, and then repeatedly measures copies of that state to estimate its vector $Aw/\|Aw\|$ of amplitudes
- This speeds up the power method to time roughly $d^{1.5}$ in the case of a constant eigenvalue gap $\lambda_1 - \lambda_2$
- PCA: approximate top-q eigenvectors in time $qd^{1.5}$

Summary

Faster optimization is one of the main potential applications of quantum computers to real-world problems, though probably not for the near term:

quantum time $C\sqrt{n}$ beats classical time n only for very large n;
QRAM issue: can we build quantum-accessible classical memory?

We have many quantum speed-ups, usually polynomial:

- Discrete: minimizing/maximizing over a finite set, faster shortest paths, graph sparsification, ...
- Continuous: gradient descent, linear/semidefinite programs, linear regression, principal component analysis, ...

Also many interesting open problems ...