PRINCIPLES FOR QUANTUM ALGORITHMS

Johannes Bausch

Quantum Information for Developers 2018

OVERVIEW

Goal: Get an intuitive understanding of quantum algorithm design.

Real goal: There is a *lot* of algorithms out there. Make use of them.

SESSION 1

- *Warmup*: Grover Search
- Brief Recap:
  - Complexity Classes
  - Basic Arithmetic
  - Qudits
- Szegedy Walks: better than flipping a coin
- Quantum Backtracking

SESSION 2

- *Warmup*: Let's predict (in retrospect) who wins the World Cup
- Gate Teleportation and Clifford Circuits: Magic!
- Repeat until Success
- How to load data into a quantum memory

BONUS

- Quantum Machine Learning
- Hamiltonian Simulation

SESSION 1

COMPLEXITY CLASSES

1. P
2. BPP
3. NP, MA

P

- PTIME, "poly-time"; $\mathcal{L} = \mathcal{L}_{YES} \dot{\cup} \mathcal{L}_{NO}$
- For a given input of size n, a classical Turing Machine can *decide* the problem in polynomial runtime.
- For circuits: family of Boolean circuits $C_n : n \in \mathbb{N}$, such that there exists a TM M which, on input 1^n outputs C_n, in poly-time.
- Example: unstructured search, computing digits of π

BPP

- Bounded-error poly-time; $\mathcal{L} = \mathcal{L}_{YES} \dot{\cup} \mathcal{L}_{NO}$
- Same as P, but you have coins

$$\begin{cases} \mathbb{P}(M(x) = 1) \geq \frac{2}{3} & x \in \mathcal{L}_{YES} \\ \mathbb{P}(M(x) = 1) \leq \frac{1}{3} & x \in \mathcal{L}_{YES} \end{cases}$$

BPP

- Bounded-error poly-time; $\mathcal{L} = \mathcal{L}_{YES} \dot{\cup} \mathcal{L}_{NO}$
- Same as P, but you have coins

$$\begin{cases} \mathbb{P}(M(x) = 1) \geq \frac{1}{2} + |x|^{-c} & x \in \mathcal{L}_{YES} \\ \mathbb{P}(M(x) = 1) \leq \frac{1}{2} - |x|^{-c} & x \in \mathcal{L}_{YES} \end{cases}$$

BPP

- Bounded-error poly-time; $\mathcal{L} = \mathcal{L}_{YES} \dot{\cup} \mathcal{L}_{NO}$
- Same as P, but you have coins

$$\begin{cases} \mathbb{P}(M(x) = 1) \geq 1 - 2^{-p(|x|)} & x \in \mathcal{L}_{YES} \\ \mathbb{P}(M(x) = 1) \leq 2^{-p(|x|)} & x \in \mathcal{L}_{YES} \end{cases}$$

PROBABILITY AMPLIFICATION

X_t outcome of run t, coin flip w/ prob $1/2 + q$.
$S_t := \sum_t X_t$.

Let $\mathbb{E}(S_t) := tq$, then $Var(S_t) = tq(1-q)$.

Chebyshev's inequality: Majority voting. Denote with A_t.

$$\mathbb{P}(A_t(x) = 1) = \mathbb{P}(S_t \geq t/2)$$

$$\vdots$$

$$= \frac{1}{t}\left(\frac{q^{-c}}{4} - 1\right)$$

PROBABILITY AMPLIFICATION

Chernoff bound:

$$\mathbb{P}(S_t \leq \lfloor t/2 \rfloor) \leq \exp\left[-\frac{t}{2q}\left(q - \frac{1}{2}\right)^2\right]$$

PROBABILITY AMPLIFICATION

Takehome Message:

> *The output probability of your randomized algorithm matters less than you think.*

But:

> *It does matter.*

NP

- "Non-deterministic poly-time"
- Any set of problems for which YES/NO can be *decided* with a P machine.
- Example: 3SAT, Knapsack, Subset Sum, Travelling Salesman, Hamiltonian Cycle

MA

- "Merlin-Arthur"
- Any set of problems for which YES/NO can be *decided* with a BPP machine
- Probabilities inherited from BPP
- Example: stoquastic k-SAT

BQP

- "Bounded-error quantum poly-time"
- For a given input of size n, a quantum Turing Machine can *decide* the problem in polynomial runtime.

BUT: QTM's ARE DIFFICULT.

- For circuits: family of quantum circuits $C_n : n \in \mathbb{N}$, such that there exists a **classical** TM M which, on input 1^n outputs C_n, in poly-time.
- Same acceptance/rejection bounds as BPP
- Example: Prime Factoring

QMA

- "Quantum Merlin-Arthur"
- Any set of problems for which YES/NO can be *decided* with a BQP machine
- Probabilities inherited from BQP
- Example: the local Hamiltonian problem

A FEW KNOWN RELATIONS

$\mathrm{P} \subset \mathrm{BPP} \subset \mathrm{BQP}$

$\mathrm{P} \subset \mathrm{NP} \subset \mathrm{MA} \subset \mathrm{QMA}$

$\mathrm{BPP} \subset \mathrm{MA}$

$\mathrm{BQP} \subset \mathrm{QMA}$

BASIC ARITHMETIC OPERATIONS

IMPORTANT GATES

H, X, Y, Z, T, S

CNOT, CCNOT (Toffoli)

Controlled-U

+

$$|a\rangle \xmapsto{+b} |a + b \mod 2^n\rangle$$

We use QFT.

$$|a\rangle \xrightarrow{\mathcal{F}} \frac{1}{\sqrt{2^n}} \sum_{t=0}^{2^n-1} \exp\left(\frac{at}{2^n}\right)|t\rangle$$

$$|a\rangle \xrightarrow{\mathcal{F}} \frac{1}{\sqrt{2^n}} |\phi_n(a)\rangle \otimes \ldots \otimes |\phi_2(a)\rangle \otimes |\phi_1(a)\rangle$$

where $|\phi_k(a)\rangle = (|0\rangle + \exp(a/2^k))|1\rangle)/\sqrt{2}$

Remember: $\exp(a/2^k) = 0.a_k \cdots a_2 a_1$

$$|\phi_k(a)\rangle = (|0\rangle + \exp(0.a_k \cdots a_2 a_1))|1\rangle)/\sqrt{2}$$

$$|b_1\rangle|b_2\rangle \cdots |b_n\rangle$$

Then

$$\begin{aligned}
|\phi_k(a)\rangle &= (|0\rangle + \exp(0.a_k \cdots a_2 a_1))|1\rangle)/\sqrt{2} \\
&\to (|0\rangle + \exp(0.a_k \cdots a_2 a_1 + 0.b_k))|1\rangle)/ \\
&\to (|0\rangle + \exp(0.a_k \cdots a_2 a_1 + 0.b_k b_{k-1}))|1 \\
&\vdots \\
&\to (|0\rangle + \exp(0.a_k \cdots a_2 a_1 + 0.b_k b_{k-1} \cdots
\end{aligned}$$

Gates work on qubits. But if you program in python, you don't think in bits.

THINK OF QUDITS

QUDIT LAND

- Take some number $a \in \mathbb{N}$
- Encode it in $n = \lceil \log_2 a \rceil$ many qubits $\in (\mathbb{C}^2)^{\otimes n}$.
- Treat it as one qudit in $\mathbb{C}^{2^n}$

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & \cdots \\ \vdots & & \ddots & \ddots & \\ 0 & & & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

COMPLEXITY OF BASIC ARITHMETIC OPERATIONS

Don't expect a speedup.

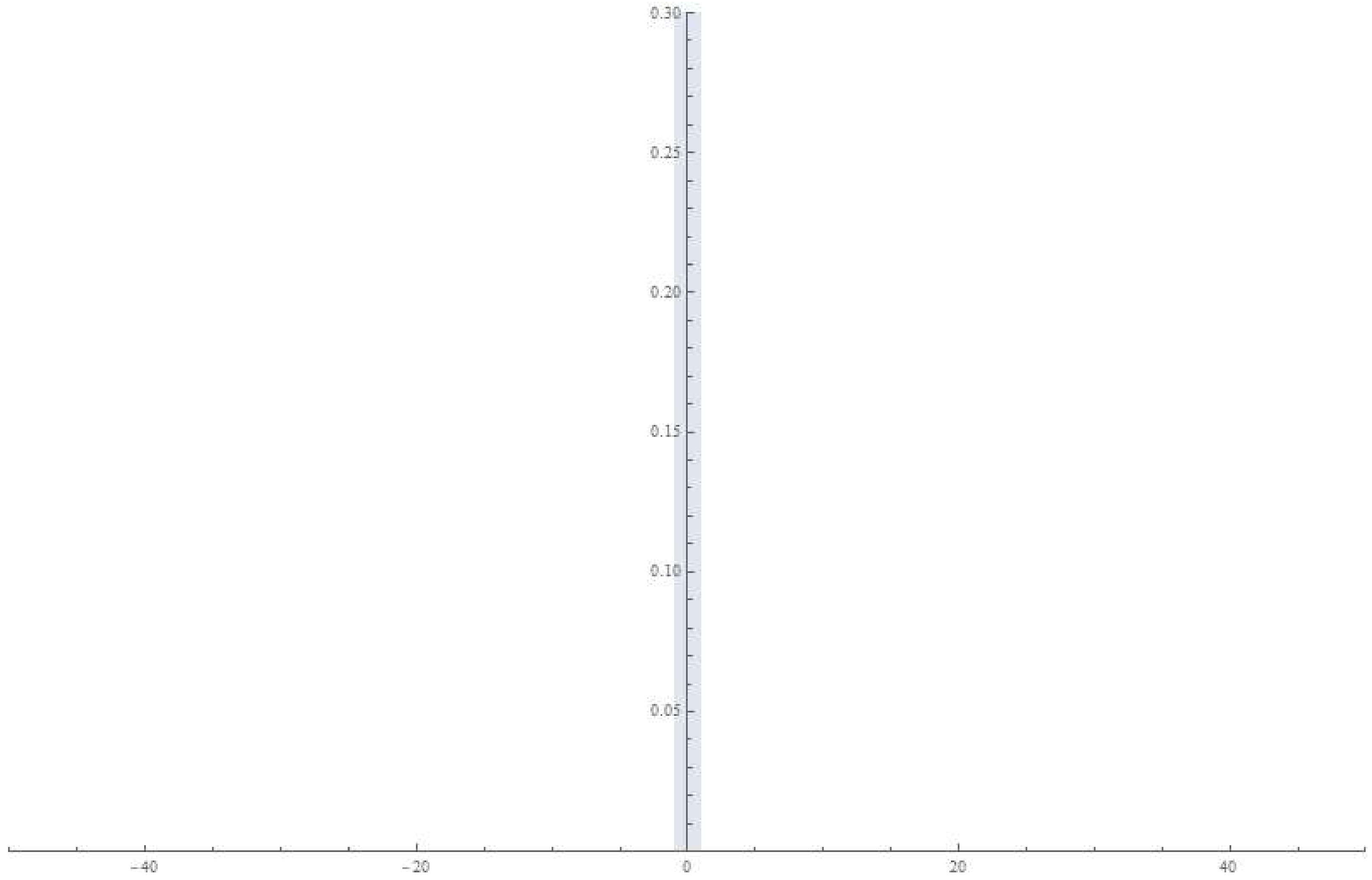
HIGH-LEVEL ALGORITHMS

https://math.nist.gov/quantum/zoo/

CLASSICAL RANDOM WALKS

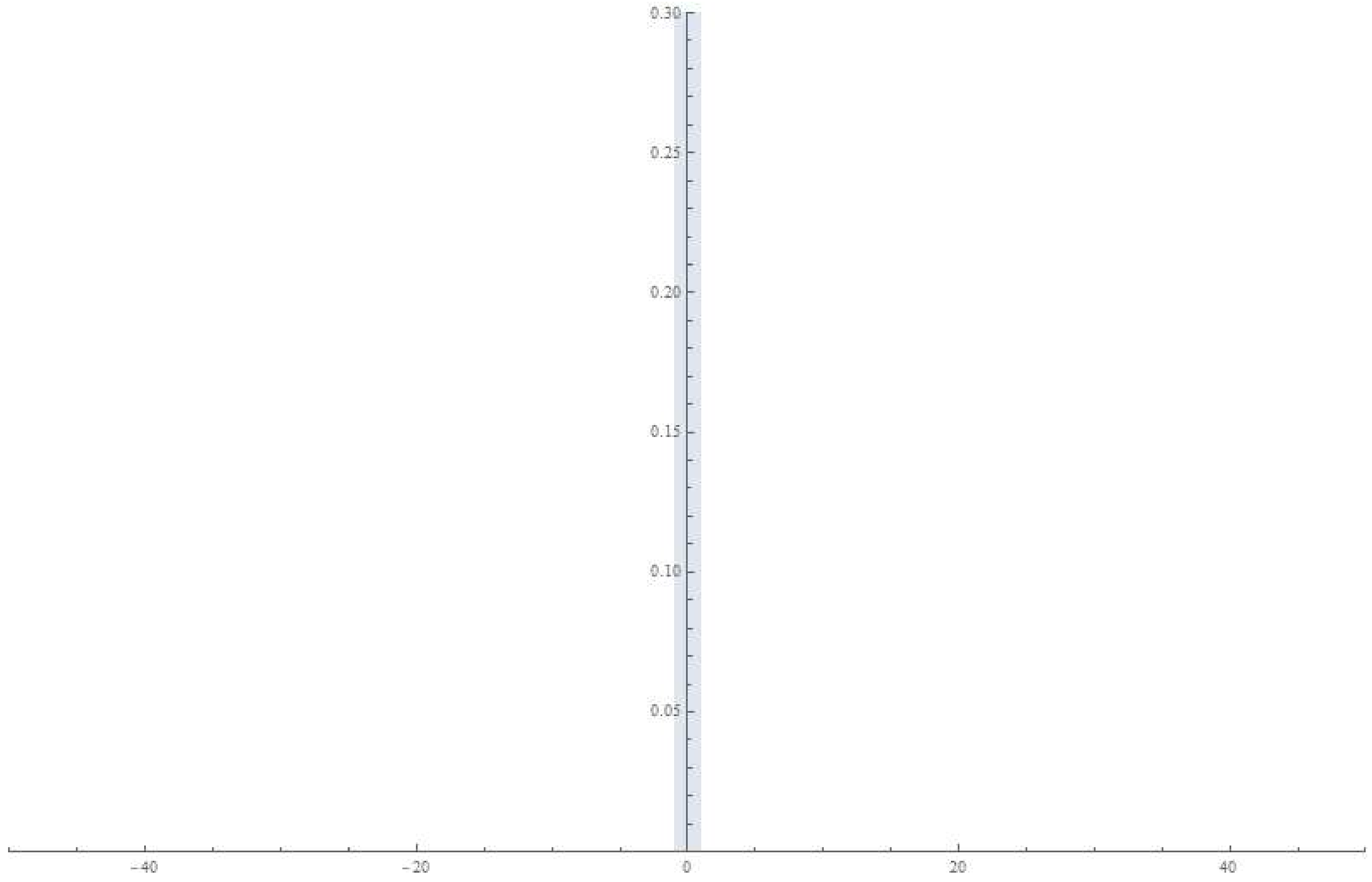
1. Markov chain: Graph $G = (V, E)$ with transition probabilities $p_e, e \in E$
2. We will usually assume ergodicity (non-patologic) and symmetry (undirected)
3. Classically: described by stochastic matrix M such that $Mx_t = x_{t+1}$

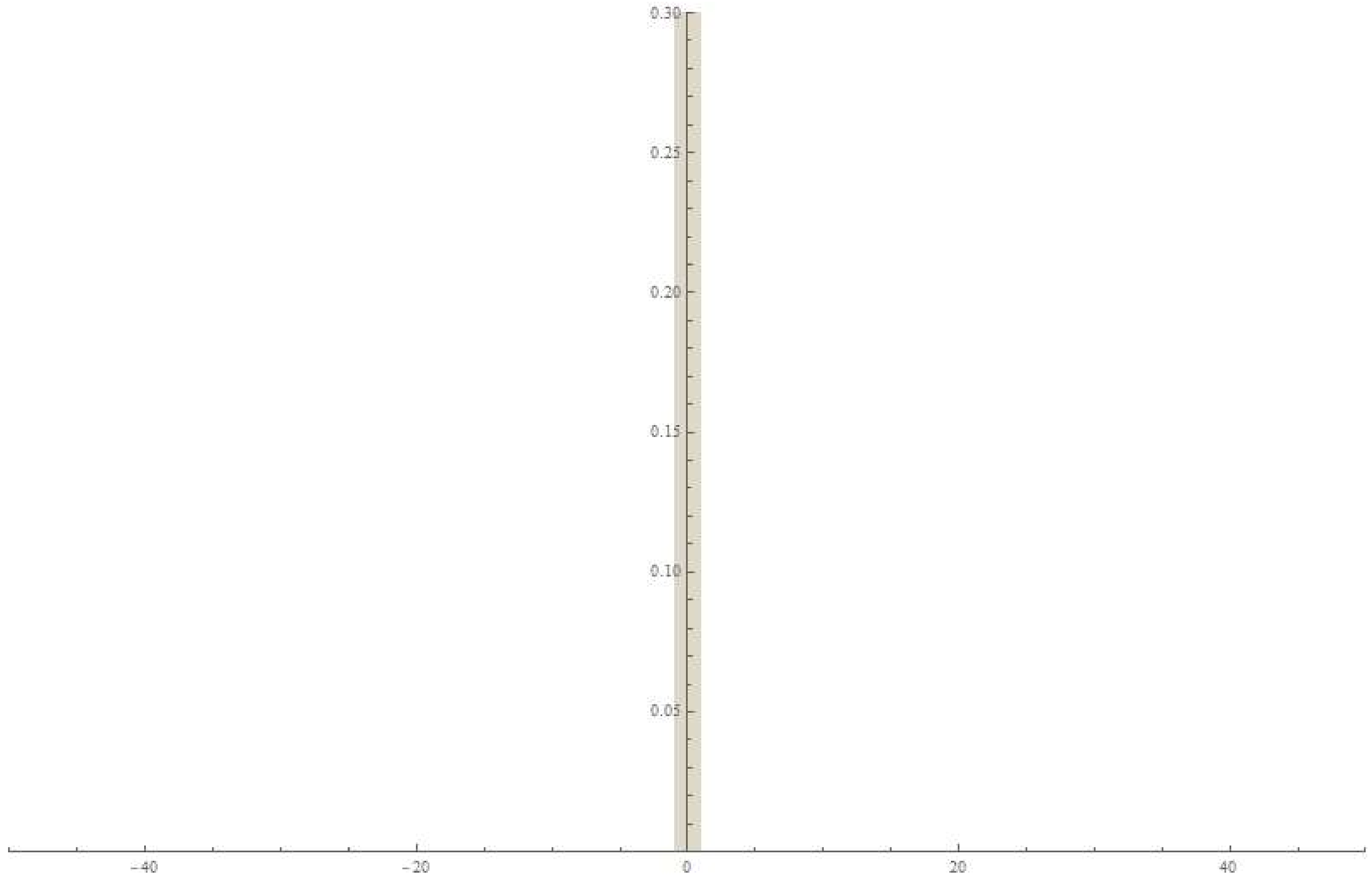
What would be a good quantum analogue of this?


0.30
0.25
0.20
0.15
0.10
0.05
-40
-20
0
20
40

QUANTUM WALKS

1. Start with a bipartite Hilbert space $\mathbb{C}^2 \otimes \mathbb{C}^L$, which is
   and location space, respectively
2. Quantum walk on a line (Aharonov): perform a coin fli
   shift:
   $$|0\rangle|l\rangle \xmapsto{H} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|l\rangle \xmapsto{S} \frac{1}{\sqrt{2}}|0\rangle|l-1\rangle +$$
3. Continuous time (Farhi)
4. Finding marked vertices in graph (Szegedy)
   - How do we make the walk *detect* marked elements?
   - What's the speedup?


0.30
0.25
0.20
0.15
0.10
0.05
-40
-20
0
20
40


0.30
0.25
0.20
0.15
0.10
0.05
-40
-20
0
20
40

SZEGEDY WALKS

Walk on a graph $G = (V, E)$ with transition probabilities p_{xy} to find some target.

- Let $\mathcal{H} = \mathbb{C}^n \otimes \mathbb{C}^n$ represent *two* copies of the graph.
- computational basis $|x, y\rangle : x, y \in V$
- Define

$$|\Psi_x\rangle := |x\rangle \otimes \sum_y \sqrt{p_{x,y}}|y\rangle$$

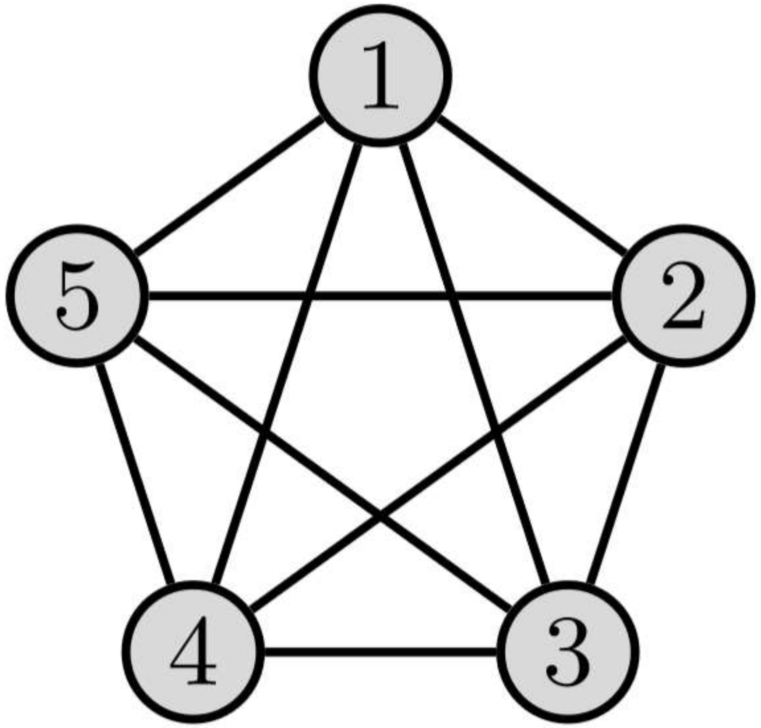
$$|\Phi_y\rangle := \sum_x \sqrt{p_{xy}}|x\rangle \otimes |y\rangle$$

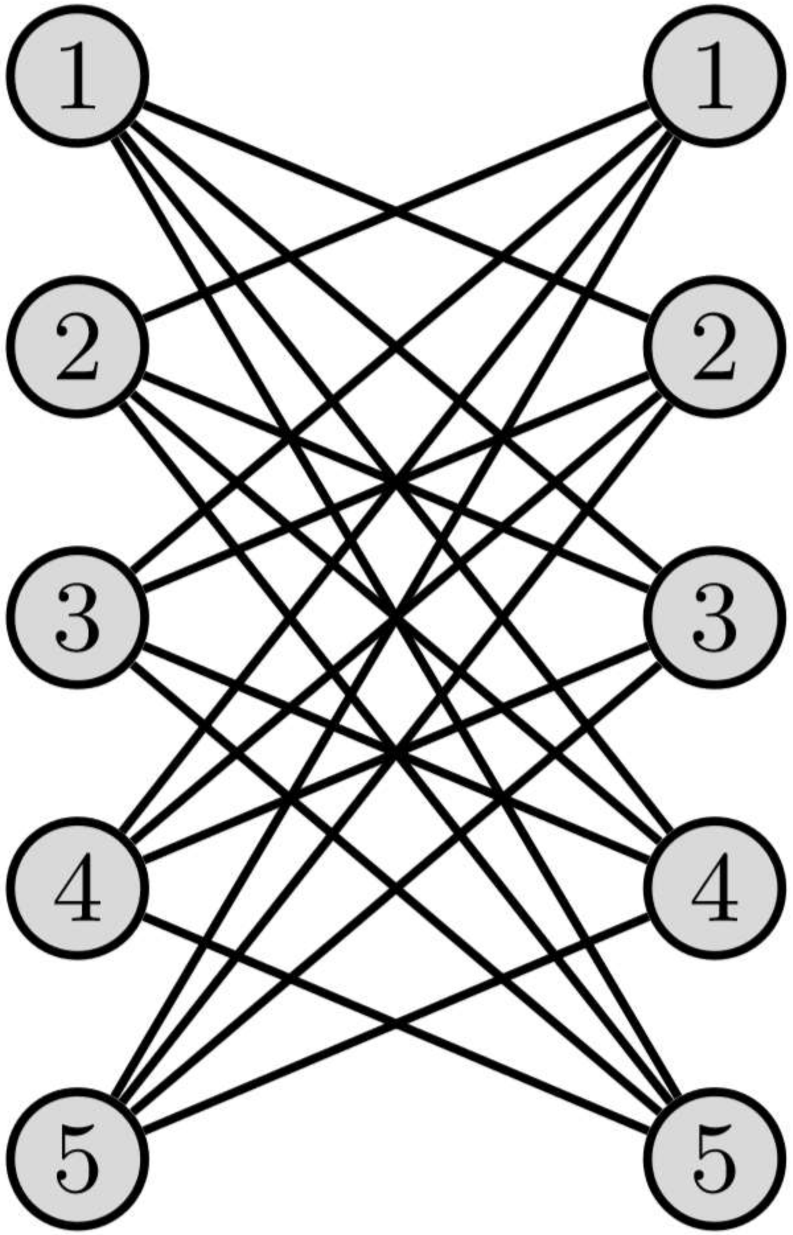
Iterate the reflections

$$R_A := 2\sum_{x\in E} |\Psi_x\rangle\langle\Psi_x| - 1_n$$

$$R_B := 2\sum_{y\in E} |\Phi_y\rangle\langle\Phi_y| - 1_n$$

1. Walk on the *edges* of the graph: each map maps an edge $|x, y\rangle$ to a superposition of edges (Santos).
2. How can we make it stop at a marked vertex $x \in M$? We have to redefine the transition probabilities


1
2
3
4
5


1
2
3
4
5
1
2
3
4
5

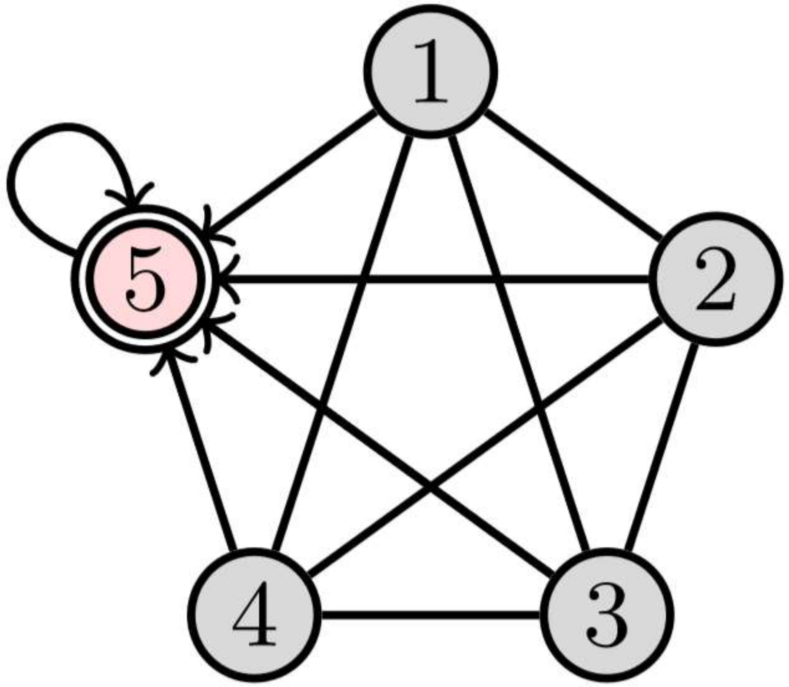
ABSORBING RANDOM WALK

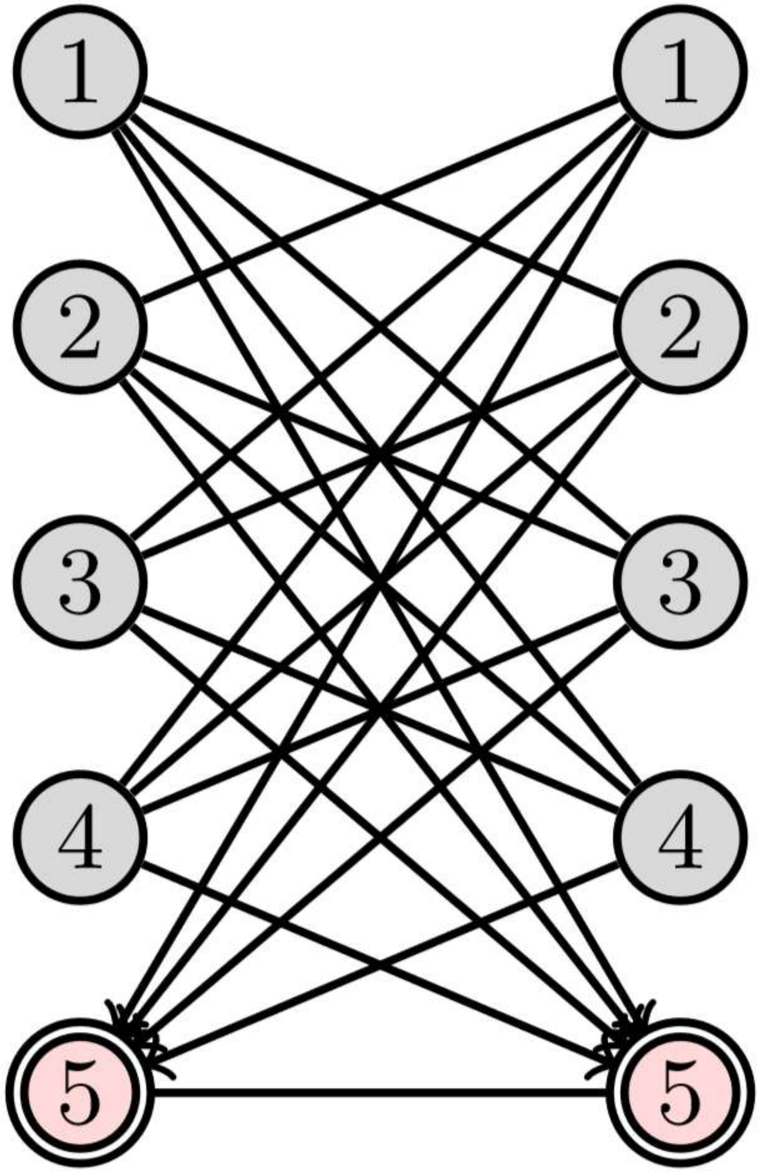
If p_{xy} is original transition probability, and M a set of marked vertices, then define

$$p'_{xy} := \begin{cases} p_{xy} & x \notin M \\ \delta_{xy} & x \in M \end{cases}$$

Initial state is stationary distribution of graph

$$|\psi(0)\rangle = \frac{1}{\sqrt{n}} \sum_{xy} \sqrt{p_{xy}} |x, y\rangle$$


1
2
3
4
5


1
2
3
4
5
1
2
3
4
5

ABSORBING RANDOM WALK

One can show that

1. The weight does not stay at the marked vertex—since the evolution is unitary
2. If there are $m = |M|$ out of $n = |V|$ marked vertices, then a marked element will be measured
   - within $t = O(\sqrt{n/m})$ steps, and
   - with probability $\geq 1/2 + O(\sqrt{m/n})$

Great—but if we already know the solution in order to modify the original transition probabilities p_{xy} for all elements in $x \in M$, why do we need to look for them at all?

ORACLES

ORACLES AND SZEGEDY WALKS

1. Standard phase kickback: the oracle maps $|x\rangle \mapsto -|x\rangle$ iff $x \in M$
2. Equivalent operation: reflection around marked elements $R_M := 2\sum_{x\in M} |x\rangle\langle x| - 1_n$
3. Promote to operator on $\mathcal{H}$ $(= \mathbb{C}^n \otimes \mathbb{C}^n)$: $R := R \otimes 1_n$
4. Change walk to $R_A R_B R R_A R_B R \ldots$

ORACLES AND SZEGEDY WALKS

One can show that the following states span invariant subspaces under $U = R_A R_B R$:

$$|a, a\rangle = \frac{1}{\sqrt{(n-1)(n-2)}} \sum_{\substack{x,y \notin M \\ x \neq y}} |x, y\rangle$$

$$|a, b\rangle := \frac{1}{\sqrt{n-1}} \sum_{x \notin M, y \in M} |x, y\rangle$$

$$|b, a\rangle := \frac{1}{\sqrt{n-1}} \sum_{x \in M, y \notin M} |x, y\rangle$$

1. This means that the operator U is a rotation in three dimensions:
$$U = \begin{pmatrix} \cos^2\phi & \cos\phi\sin\phi & -\sin\phi \\ \sin\phi & -\cos\phi & 0 \\ \cos\phi\sin\phi & \sin^2\phi & \cos\phi \end{pmatrix}$$
2. The initial state is roughly $|a, a\rangle$
3. Repeated applications of U map it to $|b, a\rangle$
4. Measurement returns a marked element with probability 1
5. Number of timesteps $O(\sqrt{n/m})$.

So we can use Szegedy walks to find marked elements in a graph, even if the vertex is determined by a subroutine.

But what if we don't know the graph beforehand?

BACKTRACKING

BACKTRACKING

- Algorithm to solve constraint satisfaction problems
- Tree exploration, where vertices are partial solutions
- Early dropout: often better than brute force
- Examples: Sudoku solver, SAT solver

BACKTRACKING

$$f : \{1, \ldots, d\}^n \longrightarrow \{\text{true}, \text{false}\}$$

Algorithm:

```
# f can return True, False, or Indetermined
def backtrack(f, x : list = []):
    # early stopping
    if f(*x) == True:
        print("solution found:", x)
        return True
    if f(*x) == False:
        return False

    # next solution; use heuristic
    for i in range(d):
        if backtrack(f, [*x, i]):
            return True
    return False
```

BACKTRACKING

Query complexity:

brute force	d^n
Grover	$d^{n/2}$
backtracking	T
quantum backtracking	$O(\sqrt{Tn}\log(1/\delta))$

---

1. $0 < \delta < 1$ failure probability
2. to also *find* a solution, extra $n \log n$.
3. uses $poly(n)$ space

QUANTUM BACKTRACKING

Ingredients:

1. Quantum Phase Estimation, to differentiate when a unitary U has eigenvalue 1: e.g. for
$$U = \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{\mathrm{i}\alpha} \end{pmatrix}$$
2. If for two states, $\| |\psi_1\rangle - |\psi_2\rangle \| = \epsilon$, then if measured in the computational basis, the total variation distance is $\leq \epsilon$

QUANTUM BACKTRACKING

We will look at the special case of **trees** where one starts at the **root** of the tree.

- T vertices labeled $r, 1, \ldots, T-1$, r being the root
- Distance from root $\leq n$; denote with $\ell(i)$
- A is the vertices with even ℓ, B with odd ℓ
- Write $x \to y$ if y is a child of x
- d_x is the degree of vertex x:

$$d_x = \begin{cases} |\{y : x \to y\}| + 1 & x \neq r \\ |\{y : x \to y\}| & \text{otherwise} \end{cases}$$

QUANTUM BACKTRACKING

- Label states $|r\rangle, |1\rangle, \ldots, |T-1\rangle$
- Define a diffusion operator D_x that only requires *local* knowledge of the tree:

  1. If x is marked, $D_x = 1_T$
  2. Otherwise, and if $x \neq r$, then $D_x = 1_T - 2|\psi_x\rangle\langle\psi_x|$ with
  $$|\psi_x\rangle = \frac{1}{\sqrt{d_x}}\left(|x\rangle + \sum_{y:x\to y}|y\rangle\right)$$
  3. $D_r = 1_T - 2|\psi_r\rangle\langle\psi_r|$ with
  $$|\psi_x\rangle = \frac{1}{\sqrt{1+nd_x}}\left(|x\rangle + \sqrt{n}\sum_{y:x\to y}|y\rangle\right)$$

- Let $R_A = \bigoplus_{x\in A} D_x$, $R_B = |r\rangle\langle r| + \bigoplus_{x\in B} D_x$

Why diffusion operator?

Think of a vertex x in the tree that $D_x = 1_T - 2|\psi_x\rangle\langle\psi_x|$ acts on. Then e.g.

$$|\psi_x\rangle\langle\psi_x| = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

is like the *adjacency matrix* of that graph segment; D_x is thus like a *Laplace operator*.

Algorithm:

1. Repeat K times:
    - apply QPE to $R_A R_B$
    - if eigenvalue is 1, accept, else reject
2. If the number of acceptance is $\geq 3K/8$, a marked vertex exists.

Why does this work?

1. If x is marked, $D_x = 1_T$
2. Otherwise, D_x diffuses the weight.
3. D_r, where r is the root element, *also* concentrates weight around r.

This means that if there is *no* marked element in the tree, there will be a single eigenvector with eigenvalue 1. This eigenvector is roughly $|r\rangle$.

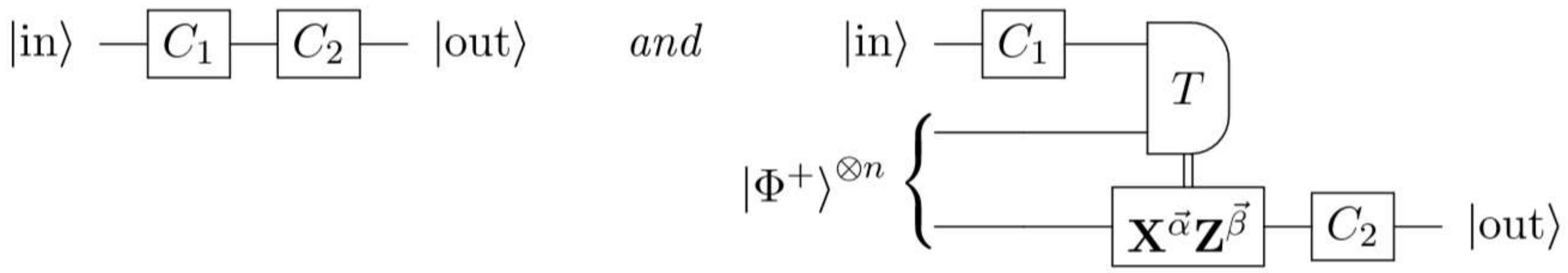
SESSION 2

WHO WON THE WORLD CUP?

https://bitbucket.org/rumschuettel/quantum-ranking

GATE TELEPORTATION

Quantum state teleportation can also be used to teleport operations around.

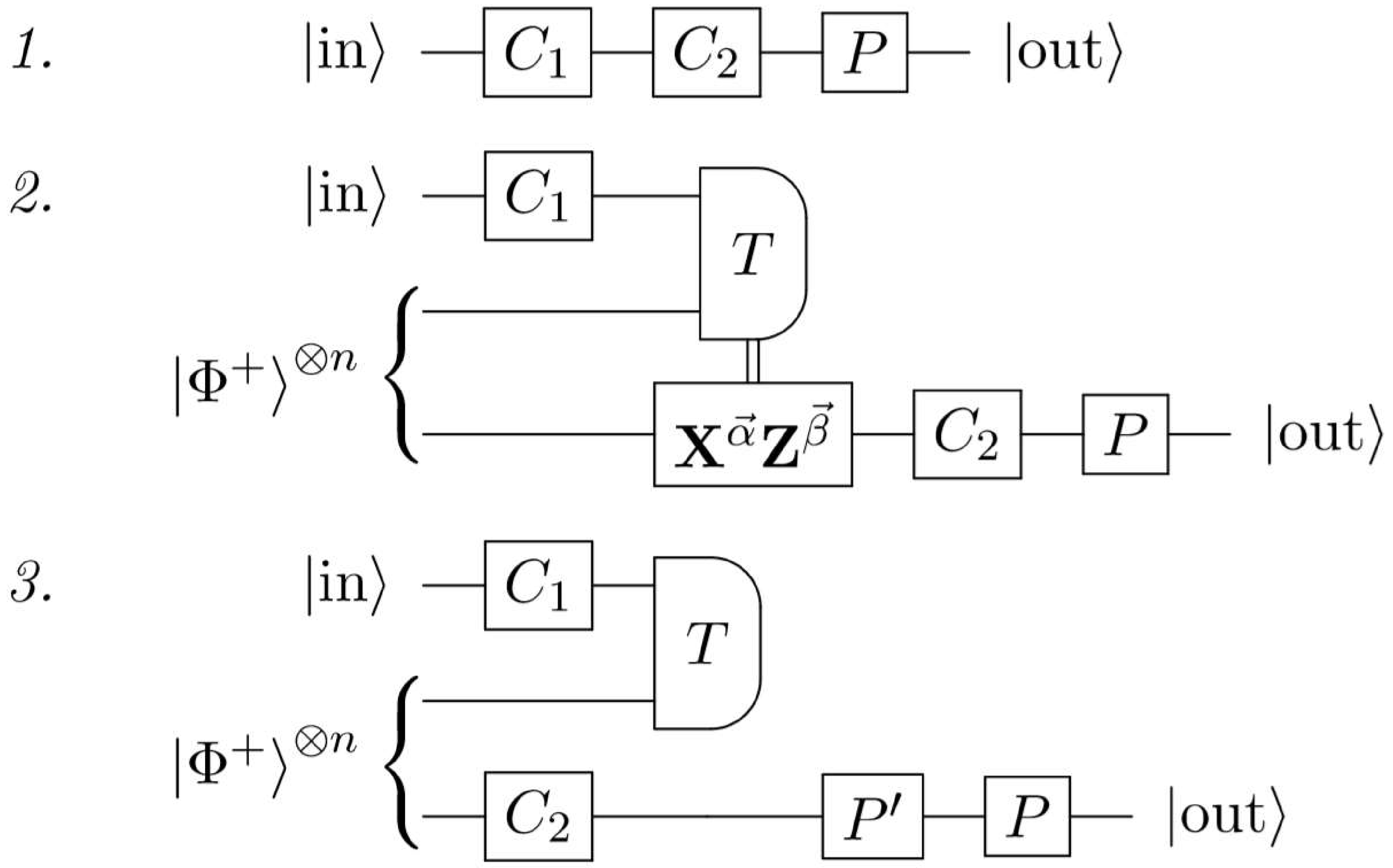


CLIFFORD CIRCUITS

1. Preparation of computational basis states, e.g. $|0\rangle$
2. gates: CNOT, H, S, Paulis (normalizers of the Pauli group)
3. Measurement in the computational basis.

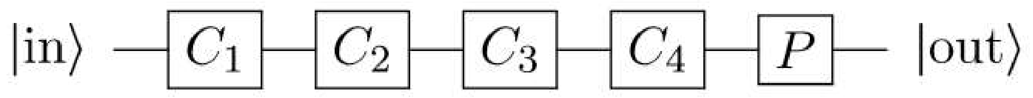
- Those are *not* yet universal
- In fact, they are classically simulable ([Gottesman-Knill])

SO WHY ARE THEY INTERESTING?

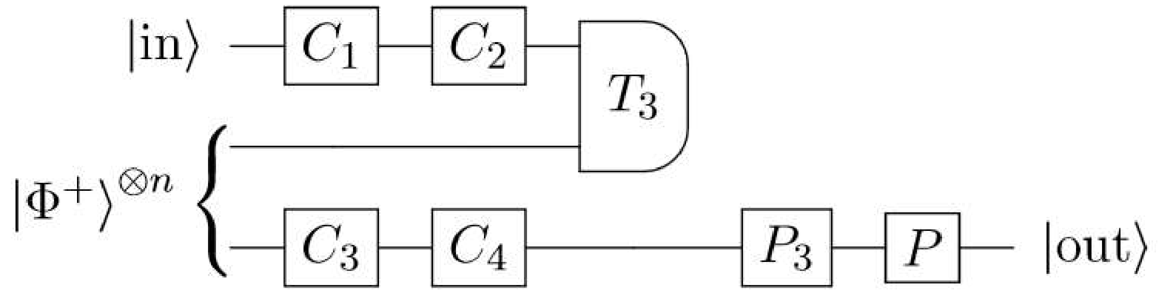


P, P' are depth-1 Pauli circuits.

1.

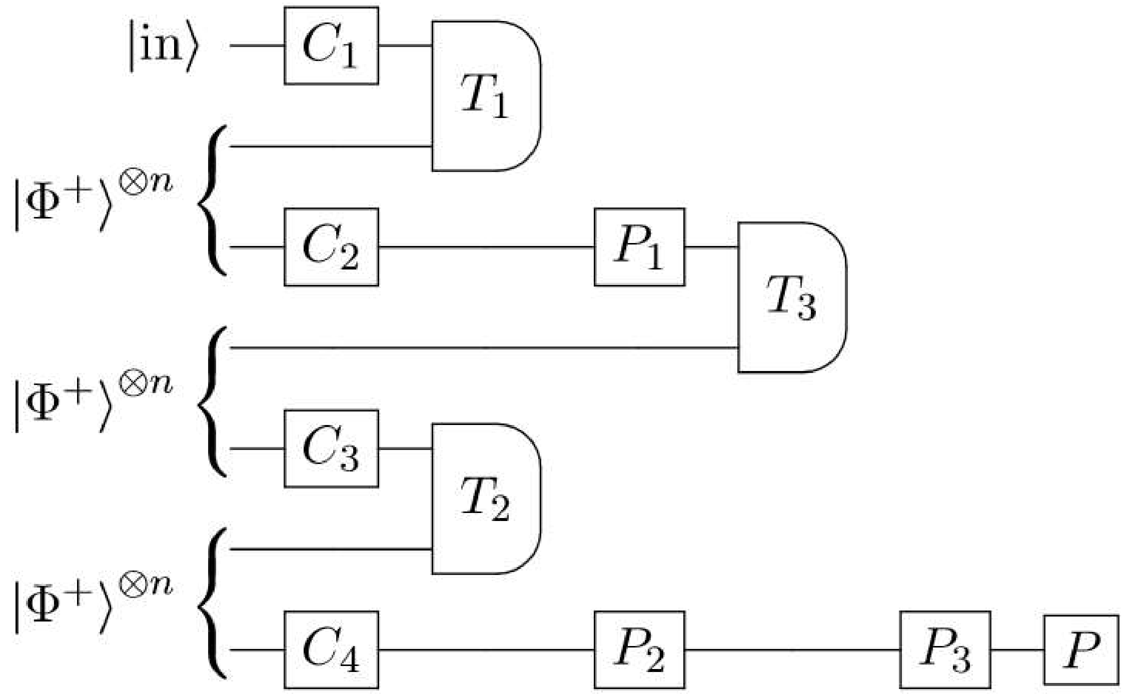


2.



where $P_3 = P_3(T_3, C_3, C_4)$.

3.


$|\mathrm{in}\rangle$
C_1
T_1
$|\Phi^+\rangle^{\otimes n}$
C_2
P_1
T_3
$|\Phi^+\rangle^{\otimes n}$
C_3
T_2
$|\Phi^+\rangle^{\otimes n}$
C_4
P_2
P_3
P

where $P_1 = P_1(T_1, C_2)$ *and* $P_2 = P_2(T_2, C_4)$.

Clifford circuits are a kind of sub-circuit that can be teleported in; a type of quantum speculative execution.

T

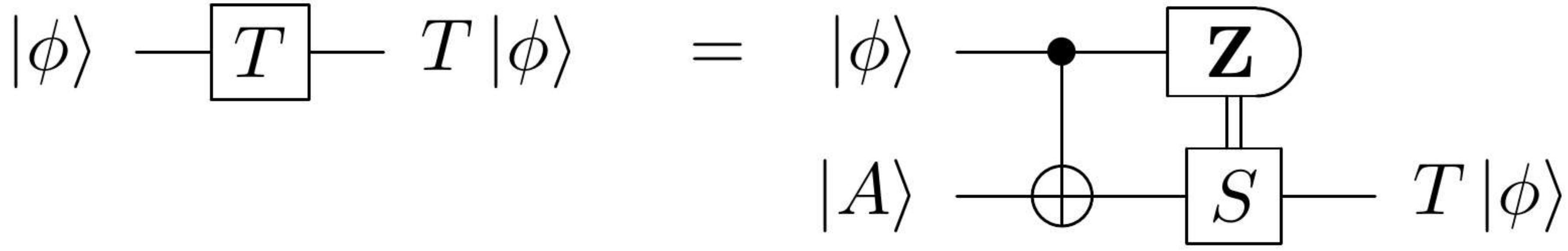
$$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

CLIFFORD + T IS UNIVERSAL.

MAGIC STATE INJECTION

Prepare the following state:

$$|A\rangle = (|0\rangle + \mathrm{e}^{\mathrm{i}\pi/4}|1\rangle)/\sqrt{2}$$



Like this, *any* quantum circuit can be decomposed into Clifford + Magic State injection.

MAGIC STATE INJECTION

=

REPEAT UNTIL SUCCESS (RUS)

REPEAT UNTIL SUCCESS

1. Prepare some states in some magic gate factory.
2. Your device can only perform a limited set of operations (e.g. measurements, and Pauli gates).
3. You attempt a gate; if it fails, apply recovery operation, and repeat.

HOW TO LOAD DATA INTO A QUANTUM MEMORY

Imagine you have a list of numbers that you want to load into your quantum device, e.g. to perform Grover search on it.

If that list is long, in time I have loaded the list I've already found the element, no?

YES, BUT...

QUANTUM DATA LOADING

REPRESENTING DATA

```
l : list = [m00, m01, m10, m11]

l[2]
# == m10

[ f(item) for item in l ]
# == [ f(m00), f(m01), f(m10), f(m11) ]
```

IN QUANTUM LAND

1. $|m\rangle = |00\rangle \otimes |m_{00}\rangle + |01\rangle \otimes |m_{01}\rangle + |10\rangle \otimes |m_{10}\rangle + |11\rangle \otimes$
2. read data: project onto corresponding address register, i.e. $(\langle 10| \otimes 1_{\text{mem}})|m\rangle = |m_{10}\rangle$
3. **BUT**: Let's exploit coherence for the function application! $(1_4 \otimes U_f$

How do we get this type of quantum memory of data?

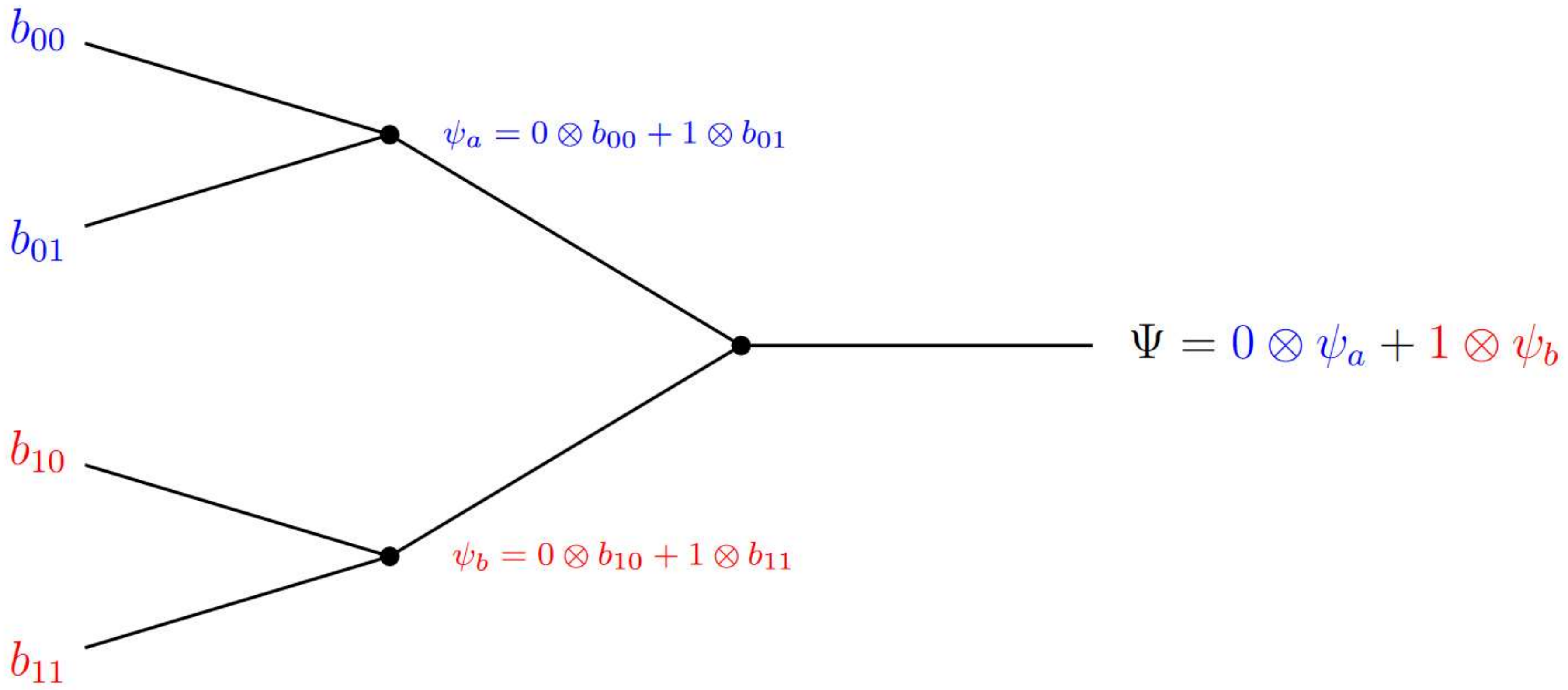
APPROACH A

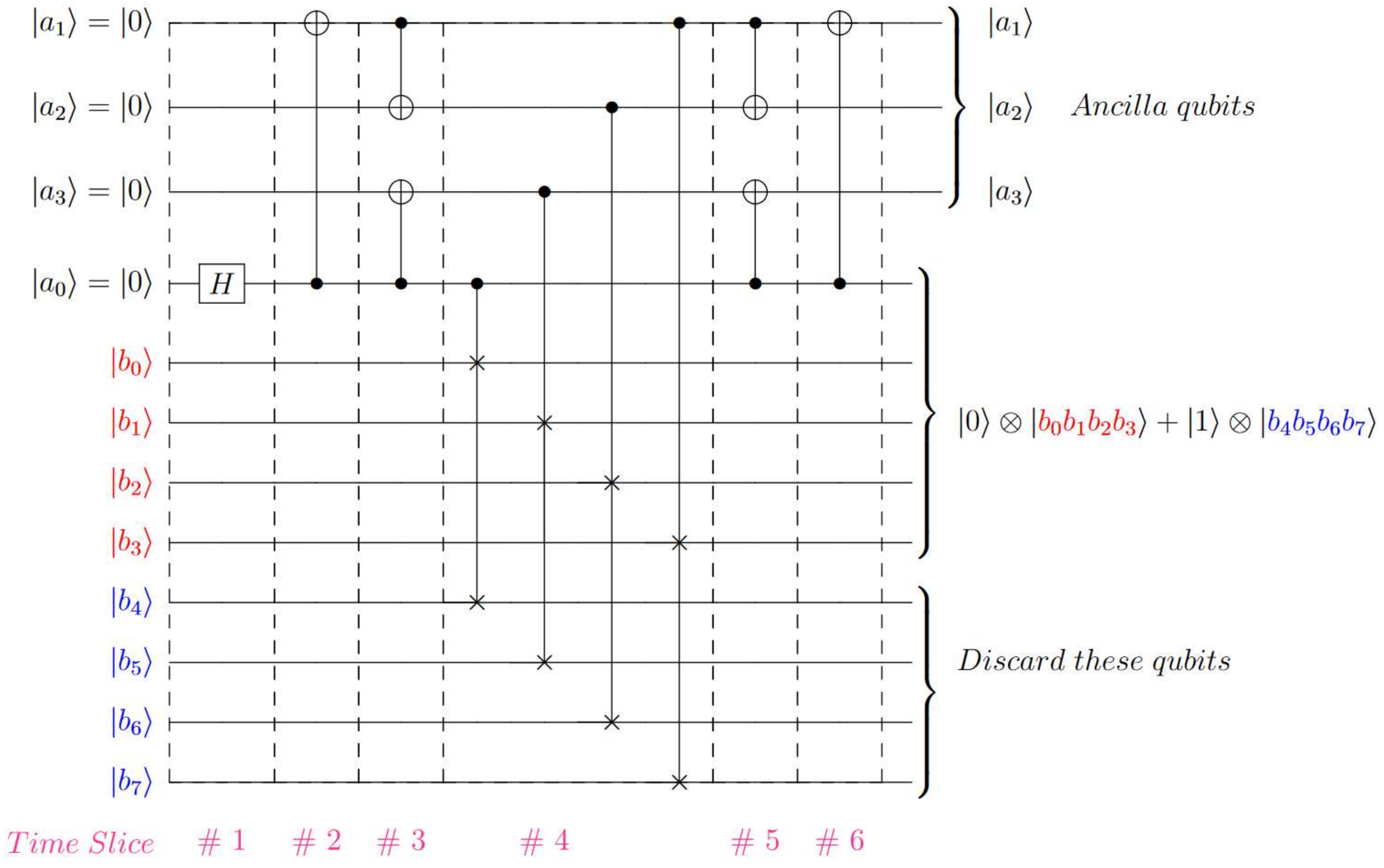
Load data serially.

APPROACH B

Load data in parallel.

PARALLEL DATA LOADING


b_{00}
b_{01}
b_{10}
b_{11}
$\psi_a = 0 \otimes b_{00} + 1 \otimes b_{01}$
$\psi_b = 0 \otimes b_{10} + 1 \otimes b_{11}$
$\Psi = 0 \otimes \psi_a + 1 \otimes \psi_b$



$|a_1\rangle = |0\rangle$
$|a_2\rangle = |0\rangle$
$|a_3\rangle = |0\rangle$
$|a_0\rangle = |0\rangle$
H
$|b_0\rangle$
$|b_1\rangle$
$|b_2\rangle$
$|b_3\rangle$
$|b_4\rangle$
$|b_5\rangle$
$|b_6\rangle$
$|b_7\rangle$
$|a_1\rangle$
$|a_2\rangle$
$|a_3\rangle$
Ancilla qubits
$|0\rangle \otimes |b_0b_1b_2b_3\rangle + |1\rangle \otimes |b_4b_5b_6b_7\rangle$
Discard these qubits
Time Slice
# 1
# 2
# 3
# 4
# 5
# 6

The gate depth of loading classical data into a quantum memory can be reduced exponentially.

But we still need to read the information once in first place.

QRAM is somewhat unrealistic.

QUANTUM MACHINE LEARNING

1. Use ML to learn something about quantum systems.
2. Use quantum algorithms to speed up classical neural nets.
3. Quantum neural nets.

NEURAL NETWORK STATES

If $|\psi\rangle = \sum_{i=1}^{2^n} \alpha_i |i\rangle$, we need exponentially many weights to represent the state. So do a *variational ansatz*:

1. Find a function $i \longmapsto f(i) \approx \alpha_i$.
2. Find a function which maintains some property of the state, e.g. entanglement entropy, fidelity wrt. some observable, ...

We know this from physics: a family of wavefunctions is used to minimize the energy wrt. some Hamiltonian.

NEURAL NETWORK STATES

$f(i)$ is a neural network, e.g. RBM, feed forward, recurrent, autoencoder, name your favourite.

I sense... competition.

1. Matrix Product States (MPS)
2. Projected Entangled Pair States (PEPS)
3. Tensor network states

NEURAL NETWORK STATES

Surprisingly good for a range of tasks.

1. Representing ground states of Hamiltonians.
2. Communication and error correction

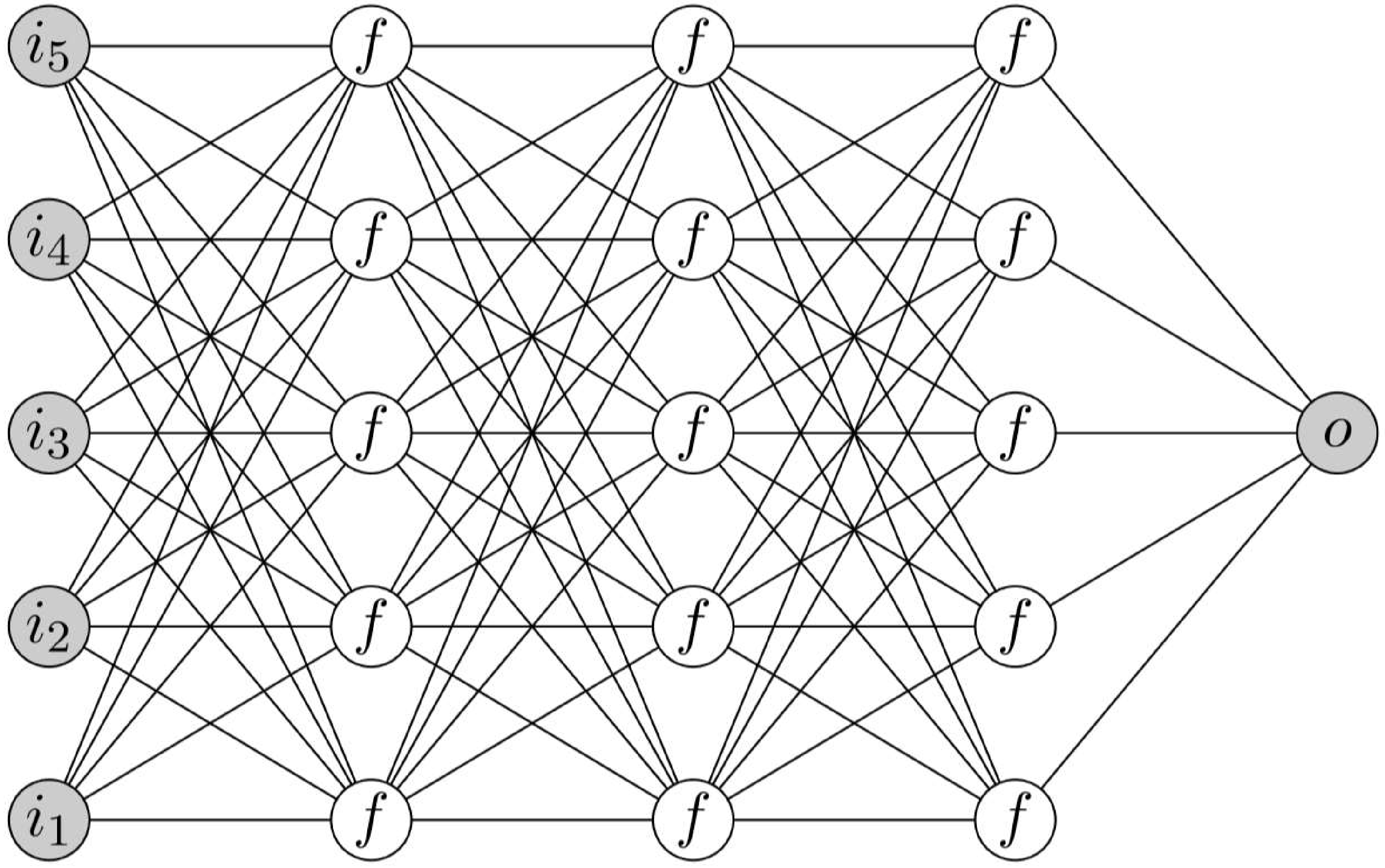
https://bitbucket.org/rumschuettel/coherent-information-optimizer

SPEEDING UP LEARNING

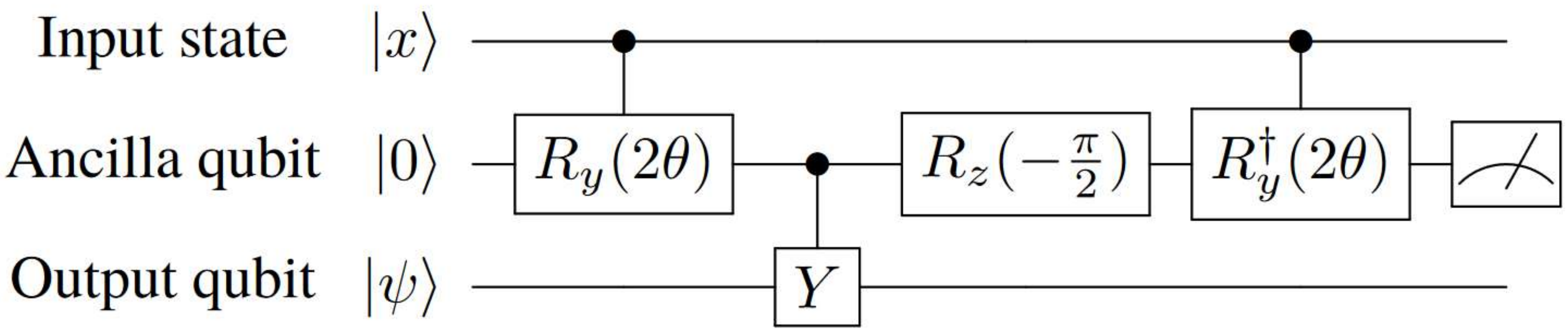
1. SVMs, principal component analysis: HHL
2. Use any quantum optimization algorithm:
   - Grover-type algorithms
   - Adiabatic evolution, annealing
   - Quantum gradient decent
3. Quantum approximate optimization algorithm (QAOA)

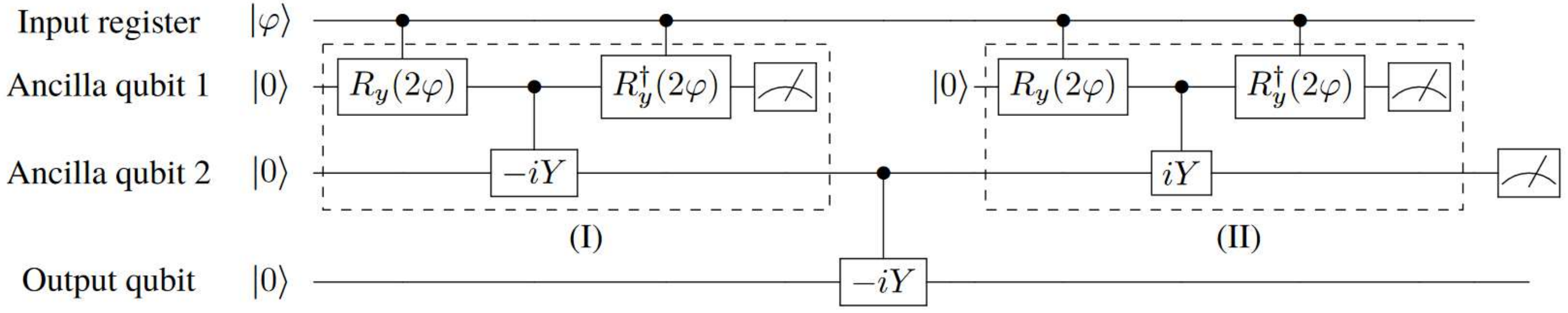
QUANTUM NEURAL NETWORK

If quantum mechanics is linear, how do we encode a non-linear activation function, like Sigmoid, or ELU?

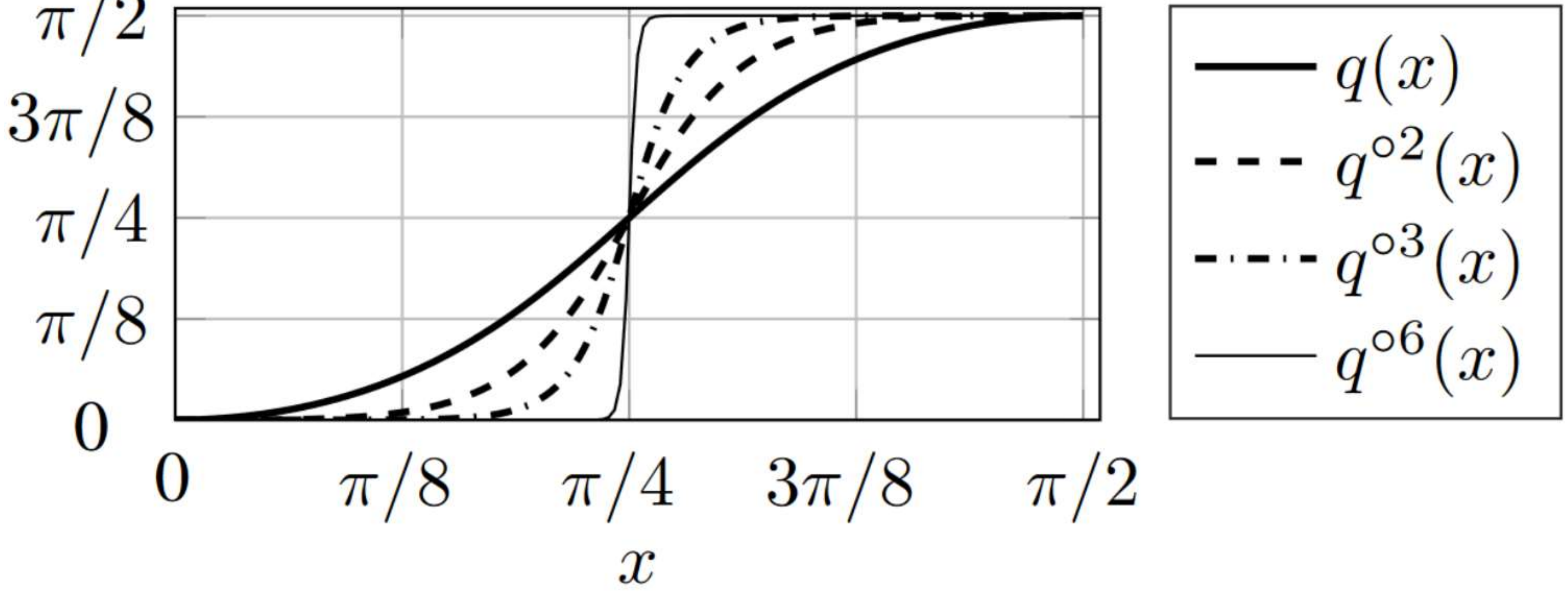

i_5
i_4
i_3
i_2
i_1
f
o

There is a quantum neuron:

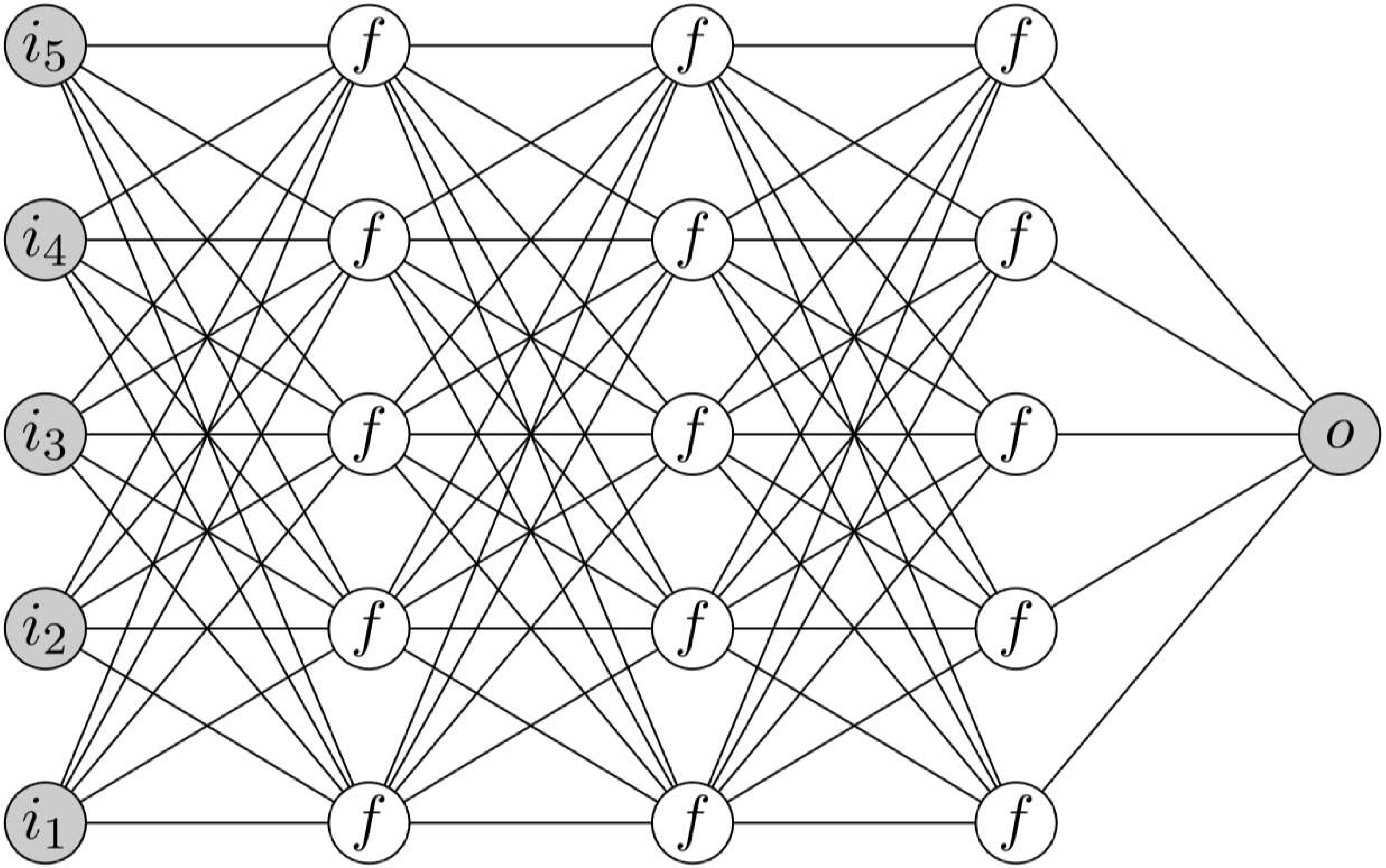




Input register
$|\varphi\rangle$
Ancilla qubit 1
$|0\rangle$
Ancilla qubit 2
$|0\rangle$
Output qubit
$|0\rangle$
$R_y(2\varphi)$
$-iY$
$R_y^\dagger(2\varphi)$
(I)
$-iY$
$|0\rangle$
$R_y(2\varphi)$
iY
$R_y^\dagger(2\varphi)$
(II)



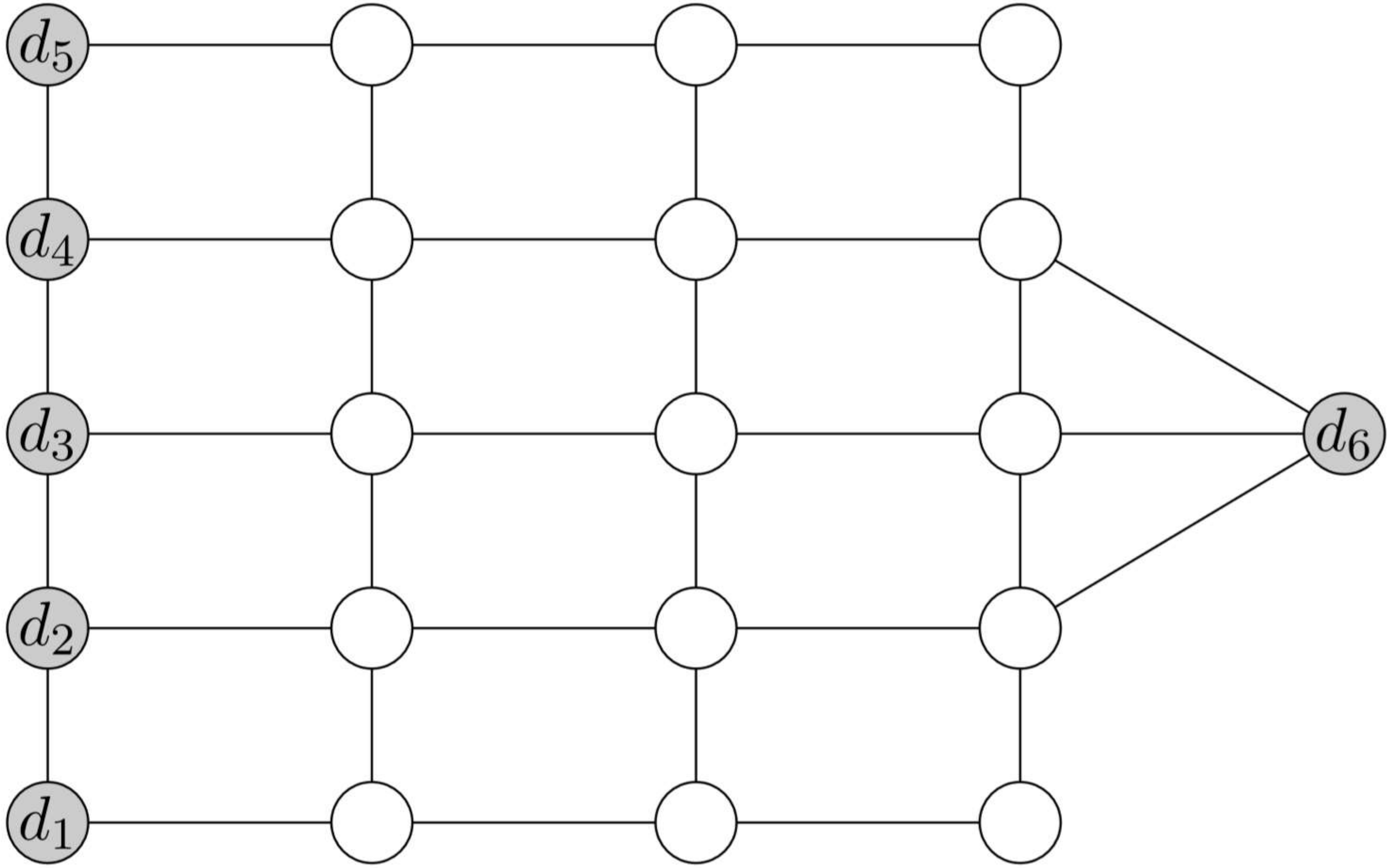
$\pi/2$
$3\pi/8$
$\pi/4$
$\pi/8$
0
0
$\pi/8$
$\pi/4$
$3\pi/8$
$\pi/2$
x
$q(x)$
$q^{\circ 2}(x)$
$q^{\circ 3}(x)$
$q^{\circ 6}(x)$



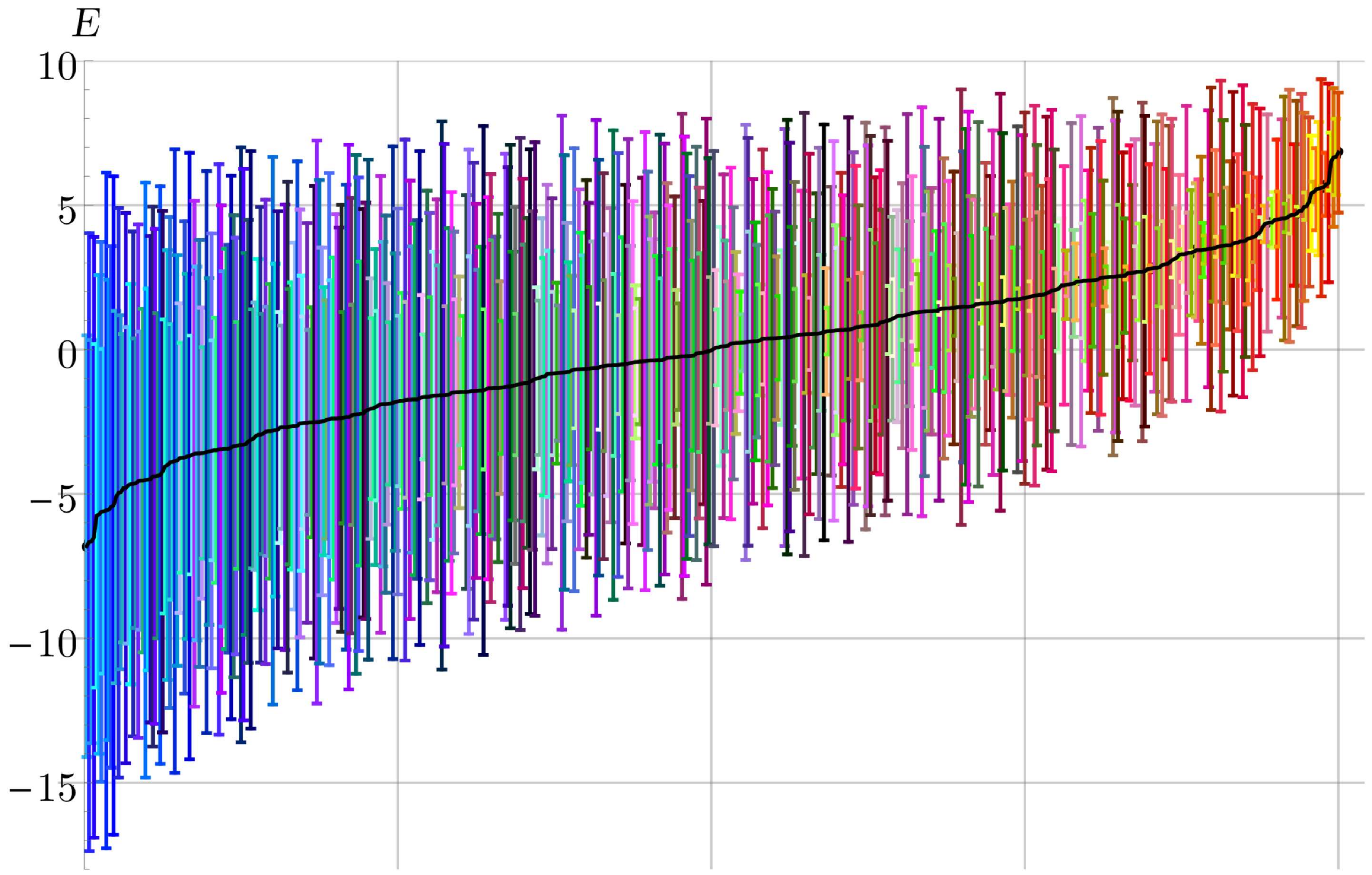
i_5
i_4
i_3
i_2
i_1
f
o

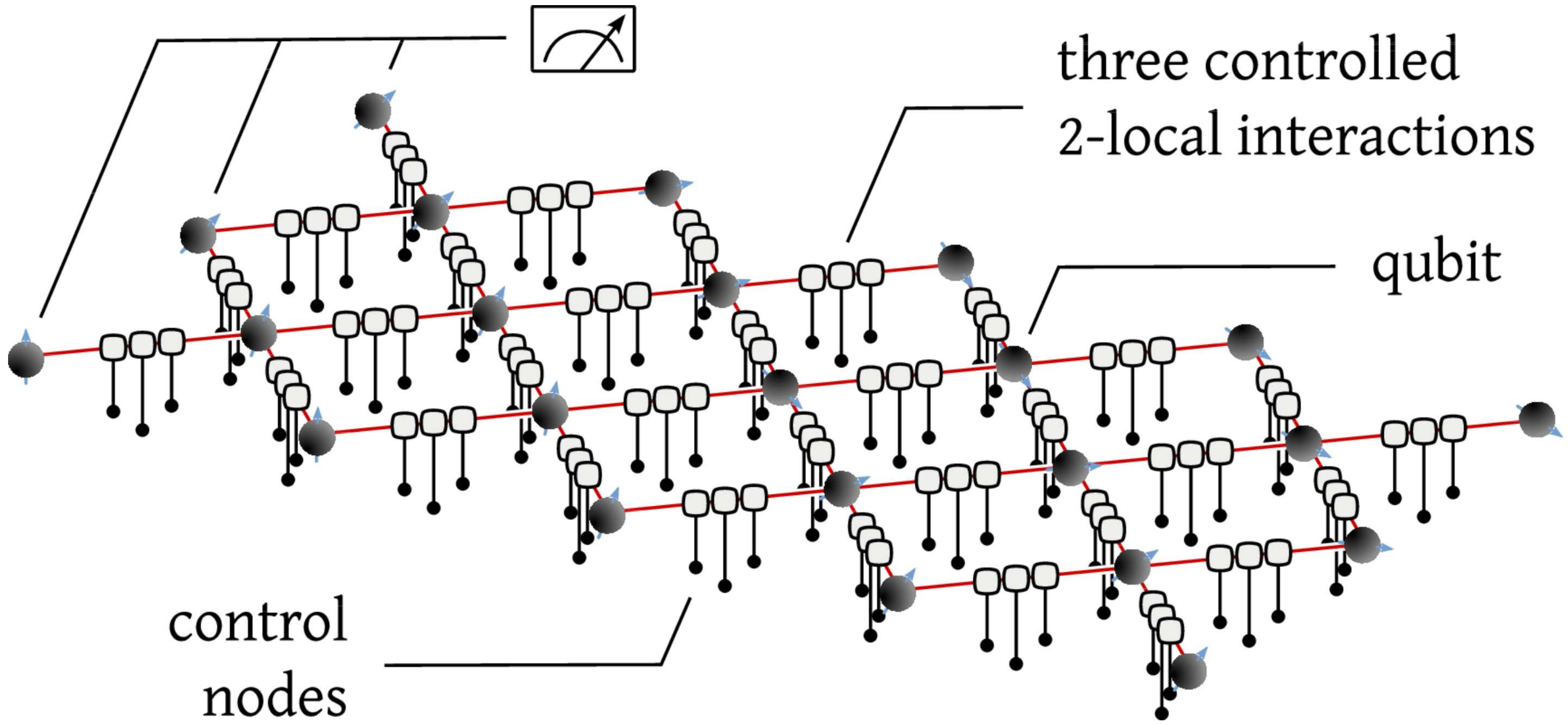
QUANTUM ANNEALING

Hamiltonian Ground States



d_5
d_4
d_3
d_2
d_1
d_6


E
10
5
0
−5
−10
−15



https://bitbucket.org/rumschuettel/liquidlearn

HAMILTONIAN SIMULATION

A Hamiltonian is a big matrix that describes the energy of a quantum system.

For instance: transverse Ising model:

$$H = \sum_{i \sim j} J_{ij} \sigma_z^{(i)} \otimes \sigma_z^{(j)} + \sum_i h_i \sigma_x^{(i)}$$

HAMILTONIAN SIMULATION

1. Simulating static properties: ground state energy
2. Simulating dynamics: approximate $\exp(\mathrm{i}tH)$.

These tasks are *hard*—at least on a classical computer.

(1) is known to be QMA-complete (depending on the precision), and (2) is known to be BQP-complete: we can, in fact, run a quantum computation with a Hamiltonian.

The problems we can exactly solve are very few. QC promise an exponential speedup over classical algorithms.

SUZUKI-TROTTER

$$\mathrm{e}^{t(A+B)} = \left(\mathrm{e}^{tA/r}\mathrm{e}^{tB/r}\right)^r + O\left(\frac{t^2}{r}\right)$$

- There exist much more sophisticated techniques
- The basic building blocks always show up: RUS, QPE, (oblivious) Grover, Quantum Walks...

THANK YOU!

QUESTIONS?