# Classical simulation of quantum circuits

Oliver Reardon-Smith

Uniwersytet Jagielloński

*oliver.reardon-smith@uj.edu.pl*

October 4, 2021

# Motivation (1)

**Simulating quantum computers is useful**

## NISQ

Present day & near future technology:
"Noisy, Intermediate Scale Quantum" computers

- Noisy means you want to compare outputs to theoretical predictions
- Intermediate scale means you *can* compare outputs to theoretical predictions

## Algorithm design

Can test and benchmark algorithms without having access to a real quantum computer

# Motivation (2)

**Simulating quantum computers is hard**

Figure 1: Bełchatów Power Station would need to run for roughly 22 years to provide enough energy to simulate Google's "quantum supremacy" circuits (Google estimate a simulation cost of 1 petawatt hour).

# Motivation (3)

**Simulating quantum computers is interesting!**

## Landscape of computations

- Quantum computations form a landscape in some high dimensional space
- For some reason some of these are easy (polynomial) to simulate and some are hard (exponential)
- Fundamentally we don't really know why

$$\begin{aligned} \text{Low qubit number} &\implies \text{easy to simulate} \\ \text{Low entanglement} &\implies \text{easy to simulate} \\ \text{Low ``magic''} &\implies \text{easy to simulate} \\ ??? &\implies \text{hard to simulate} \end{aligned}$$

# What is a quantum computer?

For the purposes of this talk:

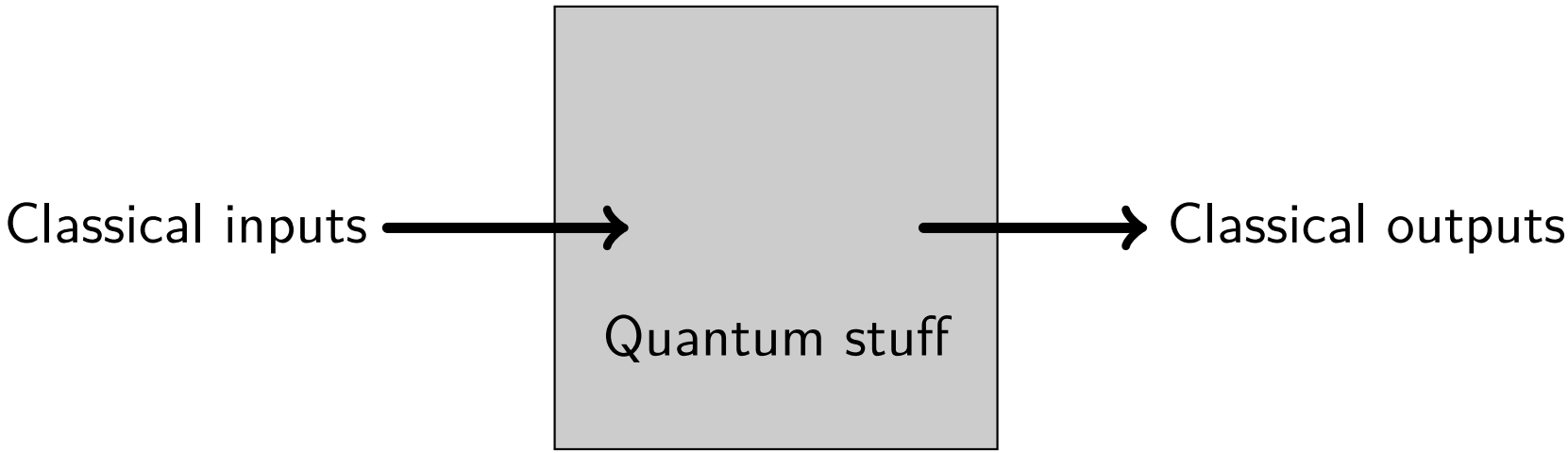



Figure 2: A quantum computer

# What is a quantum computer?

## With slightly more detail:

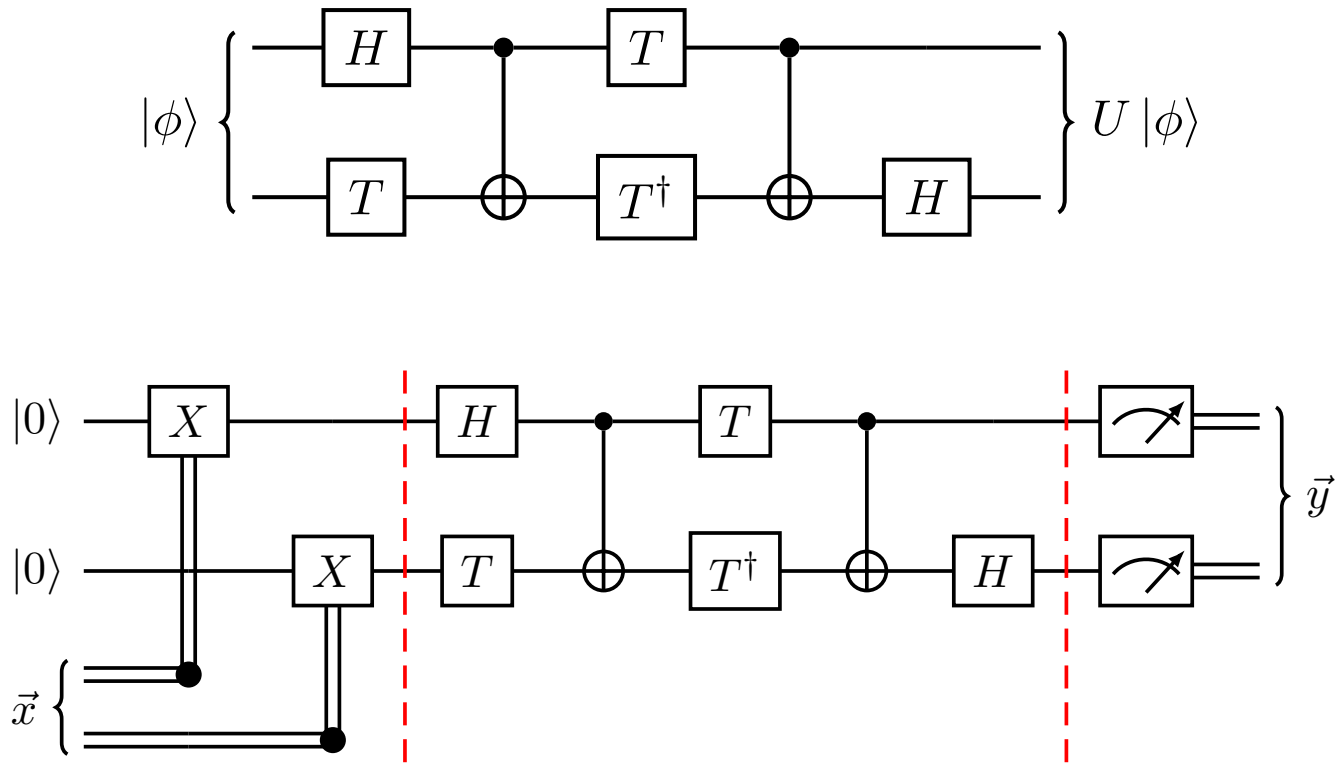



Figure 3: The quantum Fourier transform (top) used as a subroutine in a quantum computer (bottom)

# What actually is simulation?

| | Weak | Strong |
|---|---|---|
| Exact | Draw samples from Born rule distribution | Compute Born rule probabilities |
| Approximate | Draw samples from a distribution close to the Born distribution | Compute approximations to Born rule probabilities |

# Born rule probability estimation

## General problem statement

Given a quantum channel $\Phi$, an input state $\rho$, an output effect $E$ and positive real numbers $\epsilon$ and $\delta$, return a real number $p$ such that

$$\Pr\left(\left|\mathrm{tr}\left(\Phi(\rho)E\right) - p\right| > \epsilon\right) < \delta. \tag{1}$$

# Project overview

## Collaboration with

Hakop Pashayan, Kamil Korzekwa & Stephen Bartlett

## Aims

State of the art algorithms for estimating Born-rule probabilities for quantum circuits consisting of

1. Input computational basis states
2. Clifford gates: $\mathrm{H} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, $\mathrm{S} = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$, $\mathrm{CX} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
3. Non-Clifford $\mathrm{T}_\phi$ "magic" gates: $\mathrm{T}_\phi = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$
4. Output computational basis measurements

# Clifford / stabilizer overview (1)

## Pauli operators

These are $n$ qubit tensor products of

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad (2)$$

with a phase $\omega \in \{1, i, -1, -i\}$.

## Examples

$$P_1 = X \quad (3)$$

$$P_2 = X \otimes Z \otimes I \otimes Y \quad (4)$$

$$P_3 = -X \otimes X \quad (5)$$

## Notes

- Form a group (for each fixed qubit number)
- Each operator takes $4n + 4$ bits to store on a computer

# Clifford / stabilizer overview (2)

## Clifford operations

These are unitaries which map $n$ qubit Pauli operators to $n$ qubit Pauli operators

$$UP_1U^\dagger = P_2, \tag{6}$$

note that these form a (small!) subgroup of the full unitary group on $n$ qubits.

### One qubit Examples

| $P$ | $\mathrm{H}\,P\,\mathrm{H}^\dagger$ | $\mathrm{S}\,P\,\mathrm{S}^\dagger$ |
|---|---|---|
| $I$ | $I$ | $I$ |
| $X$ | $Z$ | $Y$ |
| $Y$ | $-Y$ | $-X$ |
| $Z$ | $X$ | $Z$ |

### Two qubit example

| $P$ | $\mathrm{CX}\,P\,\mathrm{CX}^\dagger$ |
|---|---|
| $X \otimes I$ | $X \otimes X$ |
| $I \otimes X$ | $I \otimes X$ |
| $Z \otimes I$ | $Z \otimes I$ |
| $I \otimes Z$ | $Z \otimes Z$ |

Storing a Clifford unitary takes $2n(4n+4)$ bits.

# Clifford / stabilizer overview (3)

## Stabilizer states

A set of Pauli operators $S = \{P_1, \ldots P_n\}$ is *independent* if

$$i \neq j \implies p_i p_j \neq I. \tag{7}$$

A quantum state $|\psi\rangle$ on $n$ qubits is a *stabilizer state* if there exists a set of $n$ independent Pauli operators $\{P_i\}$ such that

$$P_i|\psi\rangle = |\psi\rangle \qquad \forall i. \tag{8}$$

## Examples

| state | stabilizers | alternative notation |
|---|---|---|
| $\|0\rangle$ | $\{Z\}$ | $\{Z_1\}$ |
| $(\|00\rangle + \|11\rangle)/\sqrt{2}$ | $\{X \otimes X, Z \otimes Z\}$ | $\{X_1X_2, Z_1Z_2\}$ |
| $\|x\rangle$ | ??? | $\{(-1)^{x_j} Z_j \,\|\, j = 1 \ldots n\}$ |

# Clifford / stabilizer overview (4)

## Consequences of $|\psi\rangle$ being an $n$ qubit stabilizer state

- The set of Pauli operations stabilising $|\psi\rangle$ form a group written $\mathrm{Stab}(|\psi\rangle)$
- $\mathrm{Stab}(|\psi\rangle)$ has $2^n$ elements
- $\mathrm{Stab}(|\psi\rangle)$ is Abelian
- The phases of each Pauli operator in $\mathrm{Stab}(|\psi\rangle)$ are real
- $\mathrm{Stab}(|\psi\rangle) \sim \mathbb{Z}_2^n$
- Storing a stabilizer state requires $n(4n+1)$ bits
- If $U$ is a Clifford unitary then $\mathrm{Stab}(U|\psi\rangle) = U\mathrm{Stab}(|\psi\rangle)U^\dagger$

# Clifford / stabilizer overview (5)

## Gottesman-Knill theorem[1]

Circuits consisting of input stabilser states, Clifford unitaries and computational basis measurements can be simulated (strongly or weakly) in polynomial time on a classical computer.
Moreover the group of Clifford unitaries is generated by the single qubit $\mathrm{H}$ and $\mathrm{S}$ gates, and the two qubit $\mathrm{CX}$ gate for every qubit.

[1] Gottesman, Daniel (1998). "The Heisenberg Representation of Quantum Computers". arXiv:quant-ph/9807006

# Breaking out of the Clifford/stabilizer subtheory

## Main idea - "gadgetisation"

We replace all the $\mathrm{T}_\phi$ gates using the identity

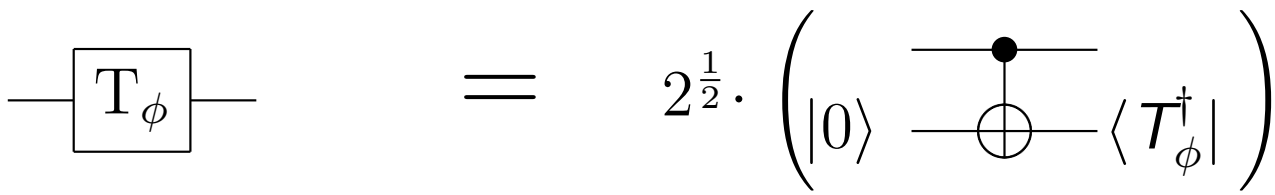



Figure 4: Single gadget

Where

$$|T_\phi^\dagger\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{-i\phi}|1\rangle\right) \tag{9}$$

This turns our $n$ qubit universal quantum circuit into an $n+t$ qubit Clifford circuit with a non-stabilizer measurement at the end.

# Gadgetisation

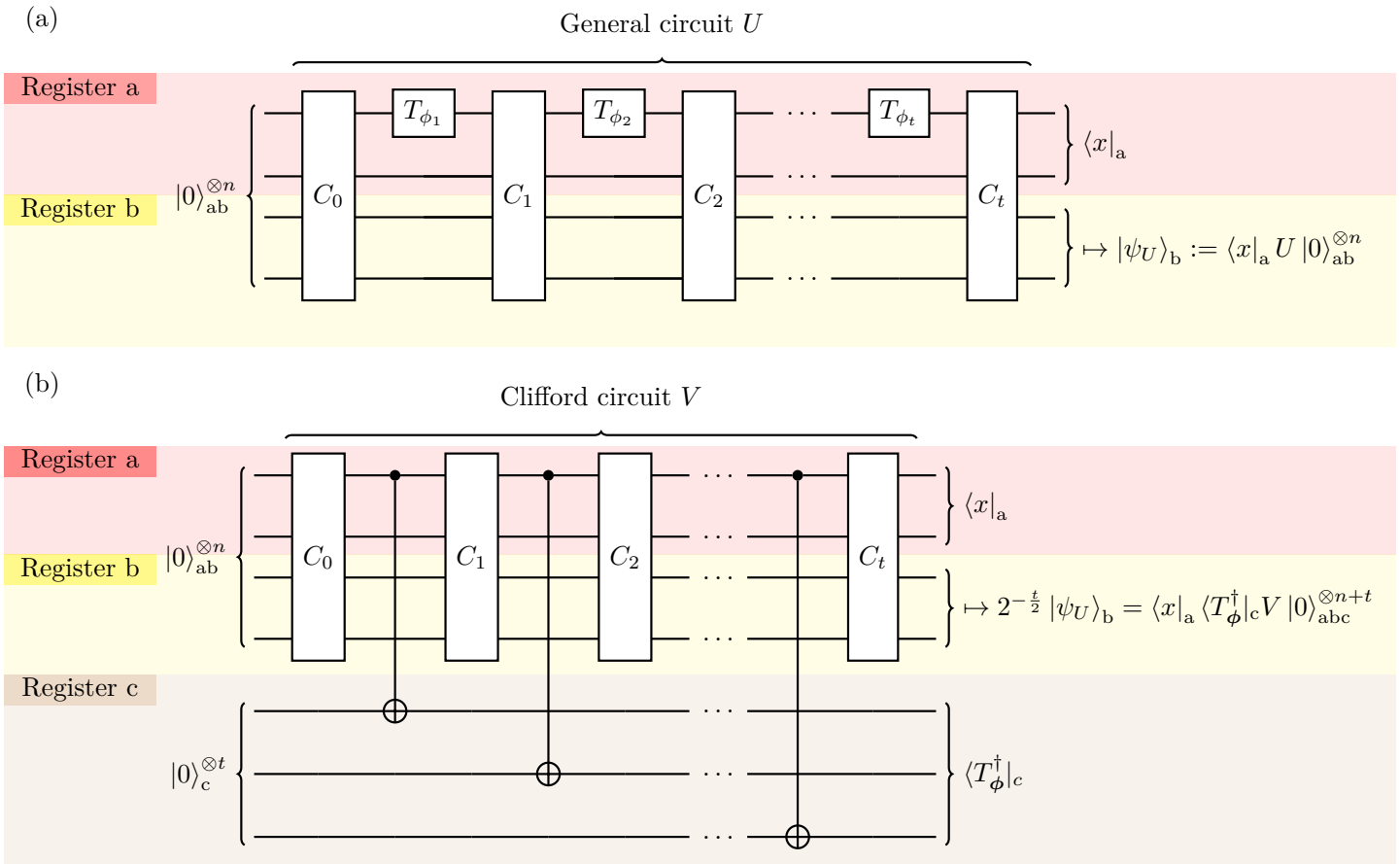



Figure 5: Clifford+T circuit (a) is gadgetised and becomes a Clifford circuit with a non-stabilizer measurement at the end (b).

# The Compress algorithm (1)

## Representation

Gadgetisation gives us a powerful representation of a quantum state, in terms of a stabilizer state and a non-stabilizer measurement at the end (which just gets stored as a list of one $\phi$ angle for each $\mathrm{T}_\phi$ gate.

## Compression

Playing with this representation lets our algorithm prove stuff about the state being represented

- Reduce number of qubits (measured/unmeasured/magic)
- Reduce number of stabilizers
- Output "compression number" $r$
- Runs in polynomial time (in all relevant parameters)

# The Compress algorithm (2)

## Input/Output

**Input**: Initial computational basis state, Clifford unitary and computational basis measurement outcome.
**Output**: Integers $v$ and $r$ and a set of $t-r$ qubit Pauli operators on $t$ qubits, $g_i$ such that

$$p = 2^{v-w} \langle T_\phi^\dagger | \prod_{i=1}^{t-r} (I + g_i) | T_\phi^\dagger \rangle, \tag{10}$$

the $g_i$ generate a group isomorphic to $\mathbb{Z}_2^{t-r}$

$$p = 2^{v-w} \sum_{g \in \langle G \rangle} \langle T_\phi^\dagger | g | T_\phi^\dagger \rangle. \tag{11}$$

# The Compute algorithm (1)

## Naive runtime for compute

We want to compute

$$p = 2^{v-w} \sum_{g \in \langle G \rangle} \langle T_\phi^\dagger | g | T_\phi^\dagger \rangle. \tag{12}$$

The elements of the runtime are

1. Computing each $g$ involves multiplying (on average) $\frac{t-r}{2}$ product operators on $t$ qubits - $O(t(t-r))$
2. Each inner product takes $O(t)$ since the states and operators are tensor products
3. Total runtime $O(2^{t-r}t(t-r))$

# The Compute algorithm (2)

## Better runtime for compute

There are efficiently computable functions $f : [0, 1 \ldots, 2^n - 1] \to \langle G \rangle$ and $g : [0, 1 \ldots, 2^n - 1] \to G$ such that

$$f(x+1) = g(x)f(x), \tag{13}$$

and moreover $f$ is a bijection.

# The Compute algorithm (3)

## Better runtime for compute

We want to compute

$$p = 2^{v-w} \sum_{g \in \langle G \rangle} \langle T_\phi^\dagger | g | T_\phi^\dagger \rangle. \tag{14}$$

The elements of the runtime are

1. Computing each $g$ involves multiplying (on average) ~~$\frac{t-r}{2}$~~ 2 product operators on $t$ qubits - ~~$O(t(t-r))$~~ $O(t)$
2. Each inner product takes $O(t)$ since the states and operators are tensor products
3. Total runtime ~~$O(2^{t-r}t(t-r))$~~ $O(2^{t-r}t)$

# Better runtime in detail

## How does this actually work?

- Recall we have a set of $t - r$ generators $G$ generating a group $\langle G \rangle \sim \mathbb{Z}_2^{t-r}$
- This group corresponds to the set of vertices of a $t - r$ dimensional hypercube where each generator in $G$ corresponds to moving unit distance in an orthogonal direction
- $f$ represents a path that visits each vertex of the hypercube exactly once, $g$ tells you which direction to walk to get to the next node in the path

# (Aside) How do you build an $n$ dimensional cube?

## Recipe

1. Take an $n-1$ dimensional cube and clone it
2. Translate the clone $1$ unit in the $n^{\text{th}}$ dimension
3. Join each vertex in the original with its corresponding vertex in the clone

Figure 6: $n = 0$

# (Aside) How do you build an $n$ dimensional cube?

## Recipe

1. Take an $n-1$ dimensional cube and clone it
2. Translate the clone $1$ unit in the $n^{\text{th}}$ dimension
3. Join each vertex in the original with its corresponding vertex in the clone

Figure 6: $n = 1$

# (Aside) How do you build an $n$ dimensional cube?

## Recipe

1. Take an $n-1$ dimensional cube and clone it
2. Translate the clone $1$ unit in the $n^{\text{th}}$ dimension
3. Join each vertex in the original with its corresponding vertex in the clone

Figure 6: $n = 1$

# (Aside) How do you build an $n$ dimensional cube?

## Recipe

1. Take an $n-1$ dimensional cube and clone it
2. Translate the clone $1$ unit in the $n^{\text{th}}$ dimension
3. Join each vertex in the original with its corresponding vertex in the clone

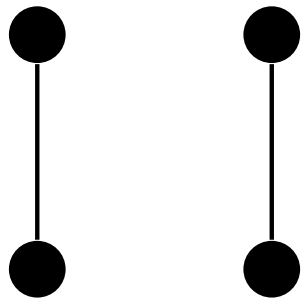


Figure 6: $n = 2$

# (Aside) How do you build an $n$ dimensional cube?

## Recipe

1. Take an $n-1$ dimensional cube and clone it
2. Translate the clone $1$ unit in the $n^{\text{th}}$ dimension
3. Join each vertex in the original with its corresponding vertex in the clone

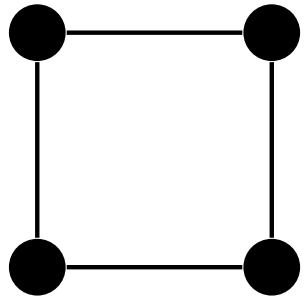


Figure 6: $n = 2$

# (Aside) How do you build an $n$ dimensional cube?

## Recipe

1. Take an $n-1$ dimensional cube and clone it
2. Translate the clone $1$ unit in the $n^{\text{th}}$ dimension
3. Join each vertex in the original with its corresponding vertex in the clone

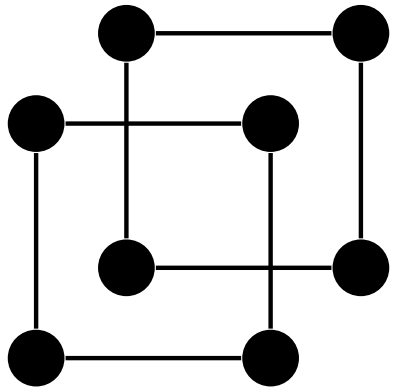


Figure 6: $n = 3$

# (Aside) How do you build an $n$ dimensional cube?

## Recipe

1. Take an $n-1$ dimensional cube and clone it
2. Translate the clone $1$ unit in the $n^{\text{th}}$ dimension
3. Join each vertex in the original with its corresponding vertex in the clone

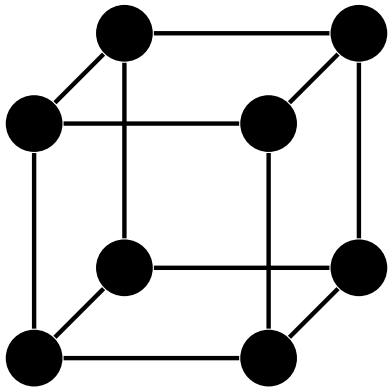


Figure 6: $n = 3$

# (Aside) How do you build an $n$ dimensional cube?

## Recipe

1. Take an $n-1$ dimensional cube and clone it
2. Translate the clone $1$ unit in the $n^{\text{th}}$ dimension
3. Join each vertex in the original with its corresponding vertex in the clone

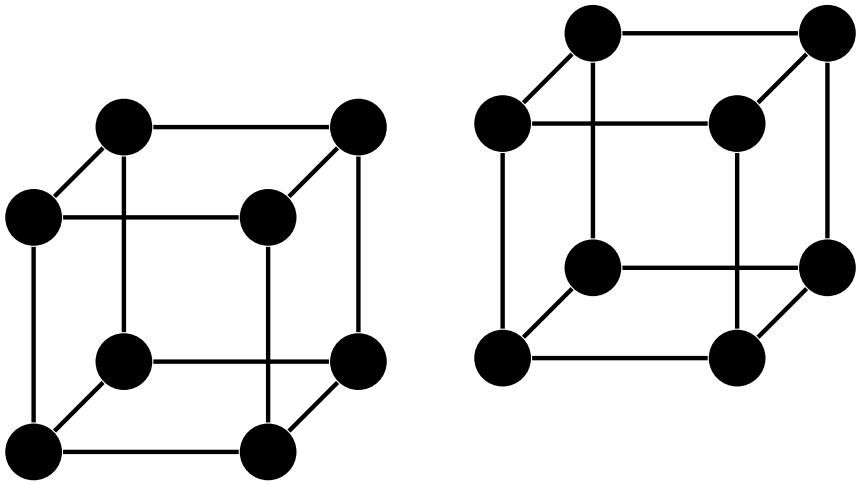


Figure 6: $n = 4$

# (Aside) How do you build an $n$ dimensional cube?

## Recipe

1. Take an $n-1$ dimensional cube and clone it
2. Translate the clone $1$ unit in the $n^{\text{th}}$ dimension
3. Join each vertex in the original with its corresponding vertex in the clone

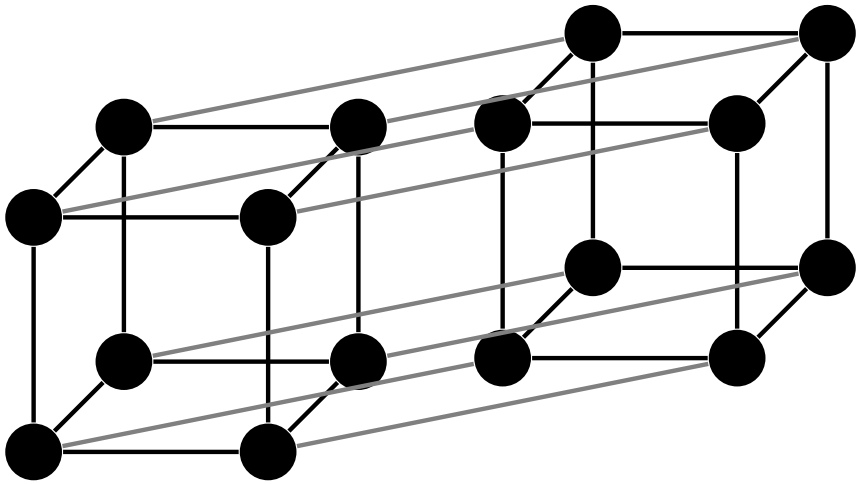


Figure 6: $n = 4$

# Binary reflected "Gray" code

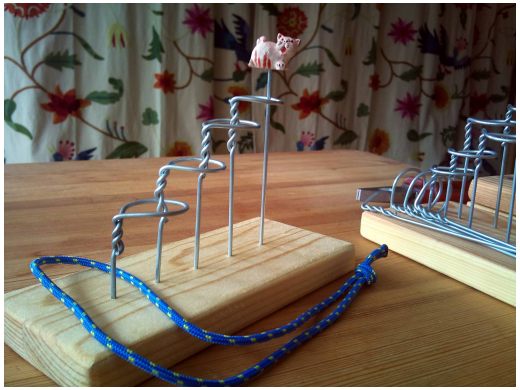
## A path through every node of the hypercube

This recursive structure is what we exploit to build a path through the $n$ dimensional cube

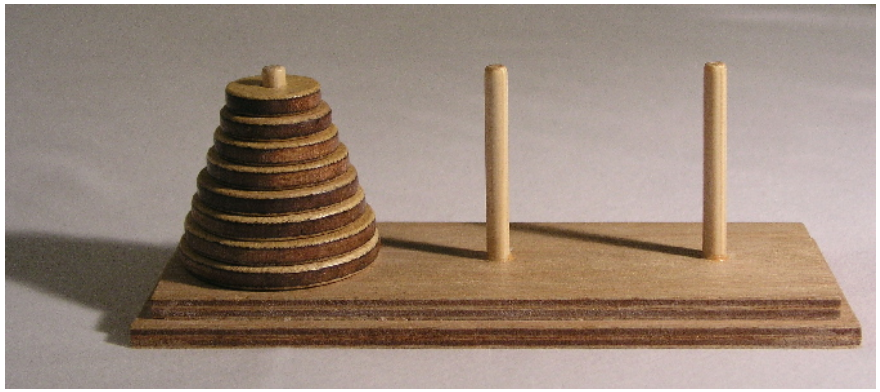
1. Follow a path that visits every node in an $n-1$ dimensional cube once
2. Now move one unit in the direction orthogonal to every edge in this path
3. Now follow the path *backwards* in the second $n-1$ dimensional cube that makes up the $n$ dimensional cube

This path forms a bijection from the natural numbers $\{0, 1, \ldots 2^n - 1\}$ to themselves known as the *binary reflected Gray code* after Robert Gray who patented it in 1947.

(a) Édouard Lucas



(b) Baguenaudier (time waster)



(c) Towers of Hanoi

Figure 7: Édouard Lucas invented the invented an elegant solution to the Baguenaudier (1872) and invented the Towers of Hanoi puzzle (1883), both employing the binary reflected code we employ in our algorithm.

# Compute vs Statevector simulator

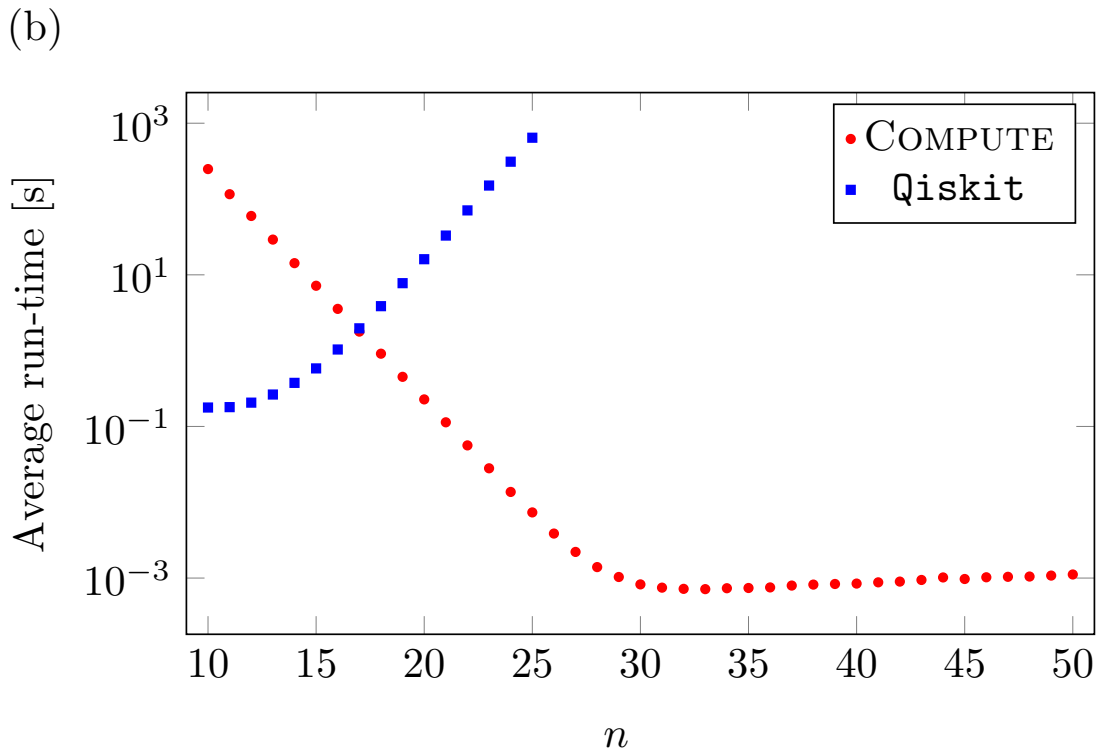



Figure 8: Performance of the Compute algorithm for random circuits. Random circuits with $c = 10^3$ Clifford gates, $t = 30$ $T$ gates and $w = 10$ measured qubits with average taken over $100$ random circuits for each $n$.

# RawEstim (1)

## Compress

In addition to the stuff for the Compute algorithm Compress gives us a $t$-qubit unitary $W$ as a circuit of $O(t^2)$ Clifford gates including $O(t)$ Hadamards such that

$$p = \left|\left|\langle 0|^{\otimes(t-r)} W |T_\phi^\dagger\rangle\right|\right|^2, \tag{15}$$

where

$$|T_\phi^\dagger\rangle = \prod_{j=1}^{t} |T_{\phi_j}^\dagger\rangle = 2^{-\frac{t}{2}} \prod_{j=1}^{t} \left(|0\rangle + e^{-i\phi_j}|1\rangle\right) = \prod_{j=1}^{t} \left(\alpha_{\phi_j}|+\rangle + \alpha'_{\phi_j}|-i\rangle\right). \tag{16}$$

# RawEstim (2)

## Expanding the magic state

Define $|\tilde{0}\rangle := |+\rangle$, $|\tilde{1}_t\rangle := |-i\rangle$ and $|\tilde{y}\rangle := \prod_j |\tilde{y}_j\rangle$ then,

$$|T_\phi^\dagger\rangle = \prod_{j=1} \left(\alpha_{\phi_j}|\tilde{0}\rangle + \alpha'_{\phi_j}|\tilde{1}\rangle\right) \tag{17}$$

$$= \sum_{y=0}^{2^t-1} \left(\prod_j \alpha_{\phi_j}^{1-y_j} (\alpha'_{\phi_j})^{y_j}\right) |\tilde{y}\rangle \tag{18}$$

$$= \sqrt{\xi^*} \sum_{y=0}^{2^t-1} p(y) \left(\prod_j e^{i\psi_j(1-y_j)} e^{i\psi'_j y_j}\right) |\tilde{y}\rangle, \tag{19}$$

where $p(y)$ is a normalized product probability distribution the $\psi_j$ and $\psi'_j$ are real numbers and $\xi^*$ is the *stabilizer extent*.

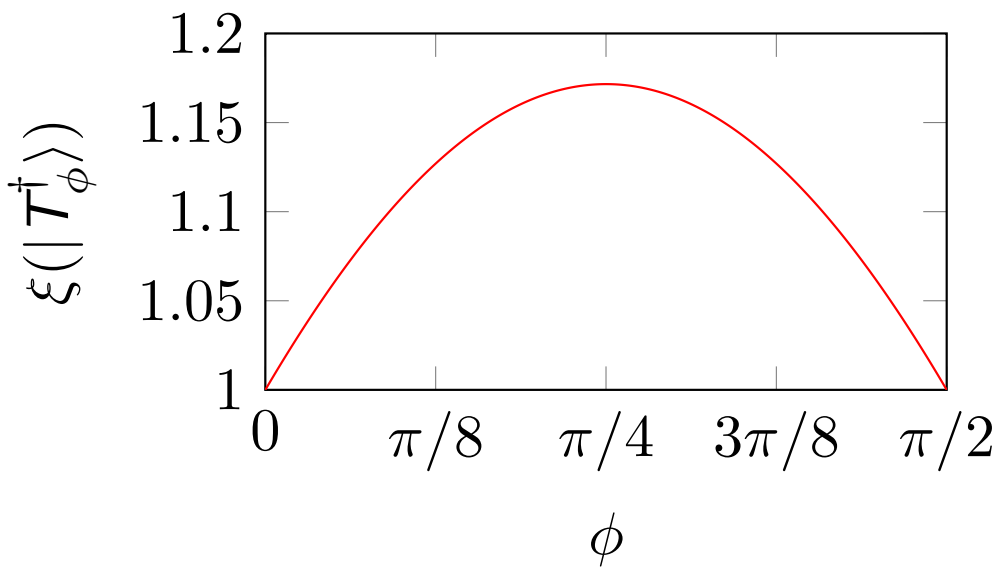
# RawEstim (3)

## Quantifying "magic"

- The stabilizer extent $\xi^*$ will be an important factor in the runtime of our algorithm
- Intuition - think of $\log \xi^*$ as a measure of how far your state is from a stabilizer state

## Explicitly we have

$$\xi(|T_{\phi_j}^\dagger\rangle) = \left(|\alpha_{\phi_j}| + |\alpha'_{\phi_j}|\right)^2$$
$$= \left(\sqrt{1-\sin\phi_j} + \sqrt{1-\cos\phi_j}\right)^2$$
$$\xi^* = \prod_j \xi(|T_{\phi_j}^\dagger\rangle)$$



# Sampling

## Basic idea

Substitute in the decomposition of the magic state from before

$$p = \xi^* \left\| \sum_{y=0}^{2^t-1} p(y) \left( \prod_j e^{i\psi_j(1-y_j)} e^{i\psi'_j y_j} \right) \langle 0|^{\otimes(t-r)} W|\tilde{y}\rangle \right\|^2 \qquad (20)$$

- The thing inside the norm is an (unnormalised) state written as the mean of a (stabiliser state-valued) probability distribution
- Take $s$ samples from this distribution and approximate the true mean with the sample mean

# Sampling (2)

## Theorem: Hayes[2]

Let $X$ be a very-weak martingale taking values in $\mathbb{R}^N$ such that $X_0 = 0$ and for every $j$, $\|X_j - X_{j-1}\|_2 \leq 1$. Then for every $a > 0$:

$$\Pr\left(||X_s|| \geq a\right) \leq 2e^{1-(a-1)^2/2s} < 2e^2 \exp\left(-a^2/2s\right). \qquad (21)$$

**Intuitively**: Take enough samples and the sample mean vector will be close to the expected value.

[2] Thomas P Hayes, "A large-deviation inequality for vector-valued martingales," Combinatorics, Probability and Computing(2005)

# Norm estimation

## Fast norm estimation[3]

Given real numbers $\epsilon$ and $\delta$ and an $n$ qubit state of the form

$$|\psi\rangle = \sum_{j=1}^{k} b_j |\phi_j\rangle, \tag{22}$$

the fast norm estimation algorithm returns a number $\eta$ such that

$$\Pr\left(|\eta - ||\psi|||| > \epsilon\, ||\psi||\right) < \delta, \tag{23}$$

with runtime $O\left(kn^3\epsilon^{-2}\log\delta^{-1}\right)$.

[3] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard, "Simulation of quantumcircuits by low-rank stabilizer decompositions," Quantum 3, 181 (2)

# Putting it together

## RawEstim

Given the output of the Compress algorithm and two positive integers $s$ and $L$, RawEstim outputs an estimate $\hat{p}$ of the outcome probability $p$ such that for all $\epsilon_{\text{tot}} > 0$ and $\epsilon \in (0, \epsilon_{\text{tot}})$:

$$\Pr\left(|\hat{p} - p| \geq \epsilon_{\text{tot}}\right) \leq 2e^2 \exp\left(\frac{-s(\sqrt{p+\epsilon} - \sqrt{p})^2}{2(\sqrt{\xi^*} + \sqrt{p})^2}\right) \tag{24}$$

$$+ \exp\left(-\left(\frac{\epsilon_{\text{tot}} - \epsilon}{p+\epsilon}\right)^2 L\right) =: \delta_{\text{tot}}. \tag{25}$$

The run-time $\tau_{\text{RawEstim}}$ of the algorithm scales as

$$\tau_{\text{RawEstim}} = O\left(st^3 + sLr^3\right). \tag{26}$$

# RawEstim summary

## Good

- Runtime linear in $\xi^*$ (since you need $S = O(\xi^*)$)
- Takes advantage of Compress - $r$ appears in runtime instead of $t$
- Complementary to Compute - good for small $r$
- Very good constants in the $O(\cdot)$ statements
- Polynomial memory use

## Bad

- The $s$ and $L$ you need depend on the unknown probability $p$
- Unclear how to divide error/runtime between norm estimation and vector estimation
- Doesn't actually solve the problem I gave at the start

# The Estimate algorithm (1)

## Recall

$$\delta_{\text{tot}} = 2e^2 \exp\left(\frac{-s(\sqrt{p+\epsilon}-\sqrt{p})^2}{2(\sqrt{\xi^*}+\sqrt{p})^2}\right) + \exp\left(-\left(\frac{\epsilon_{\text{tot}}-\epsilon}{p+\epsilon}\right)^2 L\right)$$

## Note

- Each exponential is increasing in $p$
- The first exponential is decreasing in $s$
- The second exponential is decreasing in $L$
- The $\epsilon$ lets you choose how the error is allocated between the magic sampling and the norm estimation

Taking the obvious upper bound $p \leq 1$ will give us a correct estimation algorithm but lose a lot of performance.

# The Estimate algorithm (2)

## Solution!

Roughly speaking:

1. Use the obvious upper bound $p_0 \leq 1$ but take a very large allowed error $\epsilon_0$
2. Since $\epsilon_0$ is large the RawEstim doesn't take very long even though we're choosing $s$ and $L$ based on $p_0 = 1$
3. RawEstim returns an estimate $p_0^*$ such that (with high probability) $p \leq p_0^* + \epsilon_0$
4. Now choose $p_1 = p_0^* + \epsilon_0$ and run RawEstim with $\epsilon_1 < \epsilon_0$

# The Estimate algorithm (3)

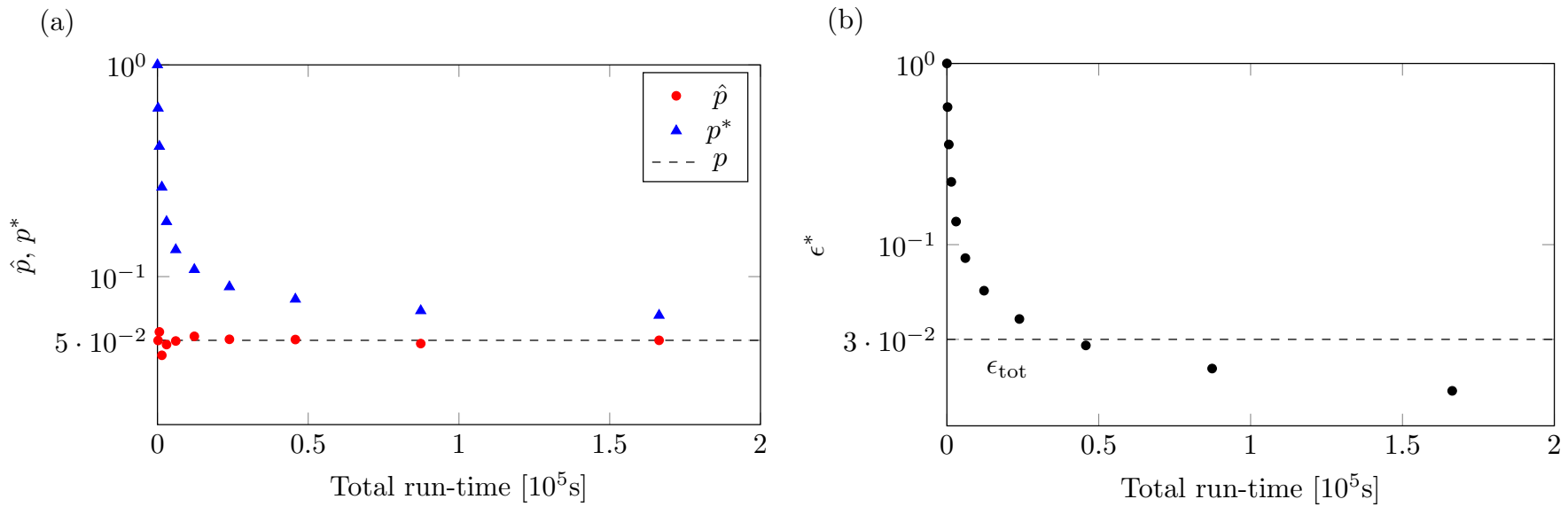



Figure 10: **Performance of the** Estimate **algorithm.** An $n = 50$ qubit, $r = 10$, $t = 60$ non-Clifford gate circuit designed to have $p = 0.05$. The circuit consists of approximately $2000$ Clifford. The parameters are chosen such that the total circuit has stabiliser extent $\xi^* \approx 3767$, equivalent to $52$ $T_{\pi/4}$ gates. The total run-time of approximately $1.6 \times 10^5$s includes approximately $34$s of overhead from the Estimate algorithm.

# Estimate summary

## Good

- All of the "good" column from RawEstim
- Works out the $s$ and $L$ you need for you
- "Relatively small" overhead over calling RawEstim with the optimal parameters
- Does actually solve the problem I gave at the start

## Bad

- The runtime is probabilistic
- The runtime depends on the initially unknown parameter $p$

# Qiskit

## Overview

- A pretty popular library - $52380$ downloads in the $30$ days preceding $23.09.2021$
- Useful for all sorts of things
  1. Quantum algorithms development
  2. Wide variety of classical simulation algorithms
  3. Access to (IBM) quantum hardware
  4. Noise mitigation research
  5. Circuit optimisation
- Started and largely developed by IBM research, technically in the hands of a community team
- By far the biggest computer library for quantum computing & information work (that I know of)

# Implementation

## What I'm doing

- I already wrote an implementation of our algorithms in C
- Qiskit is a python library but the guts are written in C++
- In addition to writing it I'm talking to the qiskit people about
  1. The user interface
  2. Which bits should/shouldn't be parallelised
  3. Various improvements
  4. Testing & documentation
- So far I have contributed code representing roughly half our paper
- I also fixed some issues in other simulation algorithms for them

# Matchgate overview

## Matchgates

Matchgate circuits are generated by two qubit gates of the form

$$G(A,B) = \begin{pmatrix} A_{11} & 0 & 0 & A_{21} \\ 0 & B_{11} & B_{21} & 0 \\ 0 & B_{12} & B_{22} & 0 \\ A_{12} & 0 & 0 & A_{22} \end{pmatrix},$$

where $A$ and $B$ are single qubit unitaries with $\det(A) = \det B$, and the two qubit gates act on adjacent qubits.

# Fermionic linear optics overview (1)

## Creation and annihilation operators

Define Fermionic creation ($a_i^\dagger$) and annihilation ($a_i$) operators for $i = 1 \ldots n$ with

$$\{a_i, a_j\} = \{a_i^\dagger, a_j^\dagger\} = 0 \qquad \{a_i, a_j^\dagger\} = \delta_{ij}, \tag{27}$$

Note that if (for any given qubit)

$$\sigma^+ = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \qquad \sigma^- = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \tag{28}$$

then

$$a_i^\dagger = (-1)^{\sum_{j=1}^{i-1} N_j} \sigma_i^+ \qquad a_i = (-1)^{\sum_{j=1}^{i-1} N_j} \sigma_i^-, \tag{29}$$

where $N_j|0\rangle_j = 0$, $N_j|1\rangle_j = |1\rangle_j$.

# Fermionic linear optics overview (2)

## Fermionic linear optics

Now define the *Majorana Fermion* operators

$$c_{2i} = a_i + a_i^\dagger \tag{30}$$

$$c_{2i+1} = i(a_i - i a_i^\dagger), \tag{31}$$

Note

$$\{c_i, c_j\} = 2\delta_{i,j} \tag{32}$$

A unitary $U$ is *Fermionic* if (and only if)

$$U c_i U^\dagger = \sum_j R_{ij} c_j, \tag{33}$$

for a real, special orthogonal, $2n \times 2n$ matrix $R$.

# Terhal and DiVincenzo

## Theorem (1)

Matchgate circuits = Fermionic Linear optic circuits

## Theorem (2)

Given a Fermionic linear optic circuit $U$ and states

$$|x\rangle = (a_1^\dagger)^{x_1}(a_2^\dagger)^{x_2}\dots(a_n^\dagger)^{x_n}|0\rangle \tag{34}$$

$$|y\rangle = (a_1^\dagger)^{y_1}(a_2^\dagger)^{y_2}\dots(a_n^\dagger)^{y_n}|0\rangle, \tag{35}$$

there is a polynomial-time classical algorithm which computes

$$p = |\langle y|U|x\rangle|^2 . \tag{36}$$

# Hebenstreit, Jozsa, Kraus, Strelchuk and Yoganathan

**Theorem**

Matchgate circuits supplemented by single qubit measurements and magic states of the form

$$|\psi_\phi\rangle = |0000\rangle + |1100\rangle + |0011\rangle + e^{i\phi}|1111\rangle, \tag{37}$$

are universal for quantum computation.

# Decomposition

## Lemma (ORS)

The magic state $|\psi_\phi\rangle$ admits the decomposition

$$|\psi_\phi\rangle = \frac{e^{iz}N_{-z}}{1+e^{iz}}\frac{1}{N_{-z}}\left(|0000\rangle + e^{-iz}|1100\rangle + e^{-iz}|0011\rangle + e^{-2iz}|1111\rangle\right) + \frac{N_z}{1+e^{iz}}\frac{1}{N_z}\left(|0000\rangle + e^{iz}|1100\rangle + e^{iz}|0011\rangle + e^{2iz}|1111\rangle\right),$$

where $\cos(z) = \frac{1}{2}\left(1+e^{i\phi}\right)$, $N_z = 2e^{-2\,\mathrm{Im}(z)}\cosh(\mathrm{Im}(z))^2$.

The two highlighted terms are each Fermionic Gaussian states (states obtainable by the action of matchgate circuits on an initial $|0\rangle$ vacuum state).

# Decomposition (2)

## Lemma (ORS)

The projector $|\psi_\pi\rangle\langle\psi_\pi|$ admits the decomposition

$$\begin{aligned}|\psi_\pi\rangle\langle\psi_\pi| = \frac{1}{16}(I &+ c_0c_1c_2c_3c_4c_5c_6c_7 + c_0c_1c_5c_6 + c_2c_3c_4c_7 \\ &+ c_0c_3c_4c_5 + c_1c_2c_6c_7 + c_0c_1c_4c_7 + c_2c_3c_5c_6 \\ &+ c_0c_3c_6c_7 + c_1c_2c_4c_5 + c_0c_2c_5c_7 + c_1c_3c_4c_6 \\ &- c_0c_1c_2c_3 - c_4c_5c_6c_7 - c_0c_2c_4c_6 - c_1c_3c_5c_7)\end{aligned}$$

There are similar decompositions for $\phi \in (0, 2\pi)$, this state is magic except as long as $\phi \neq 2k\pi$ for some $k \in \mathbb{Z}$.

# Issues with these decompositons

## Vector decomposition

We don't have a phase sensitive (i.e. vector) matchgate simulation algorithm.

Maybe if we're smart we can invent one.

## Projector decomposition

Decomposition consists of $16$ terms - leading to an algorithm with runtime like $O(t^{16})$ - this is very, very slow!

Maybe if we're smart we can optimize this.

# Phase sensitive Fermionic linear optics simulator

## Basic issue

Efficient simulation is based on "adjoint action"

$$Uc_iU^\dagger = \sum_j R_{ij}c_j. \tag{38}$$

Any phase information is lost in this representation.

## Basic solution

Fix the phase of $U$ by fixing the action on some vector

$$U|0\rangle = |0\rangle. \tag{39}$$

# Preserving the vacuum (1)

## Task 1

Characterise the FLO unitaries $U$ for which

$$U|0\rangle = |0\rangle. \tag{40}$$

Solution: If $Uc_iU^\dagger = \sum_j R_{ij}c_j$ and $U|0\rangle = |0\rangle$ then

$$RJR^T = J, \tag{41}$$

where $J$ is the *symplectic form*

$$J = \bigoplus_{q=0}^{n-1} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \tag{42}$$

In other terms - $R \in \mathrm{SP}(2n, \mathbb{R})$

# Preserving the vacuum (2)

## Question:

What can we say about the real, orthogonal matrices $2n \times 2n$ matrices which are also symplectic?

# Preserving the vacuum (2)

## Question:

What can we say about the real, orthogonal matrices $2n \times 2n$ matrices which are also symplectic?

## Answer:

$$\mathrm{O}(2n, \mathbb{R}) \cap \mathrm{SP}(2n, \mathbb{R}) \cong \mathrm{U}(n, \mathbb{C}). \tag{43}$$

# Preserving the vacuum (2)

## Question:

What can we say about the real, orthogonal matrices $2n \times 2n$ matrices which are also symplectic?

## Answer:

$$\mathrm{O}(2n, \mathbb{R}) \cap \mathrm{SP}(2n, \mathbb{R}) \cong \mathrm{U}(n, \mathbb{C}). \tag{43}$$

Concretely we have, for any $R \in \mathrm{O}(2n, \mathbb{R}) \cap \mathrm{SP}(2n, \mathbb{R})$,

$$R = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}, \qquad I = cc^T + ss^T, \qquad 0 = cs^T - sc^T, \tag{44}$$

and these conditions are exactly equivalent to

$$c + is \in \mathrm{U}(n, \mathbb{C}). \tag{45}$$

# What is left over? (1)

## Generating set

The FLO unitaries on $n$ qubits are generated by a set of $6n$ one-parameter unitary groups $\exp(itH)$ where $H$ is one of

$$a_i^\dagger a_i \qquad a_{i+1}^\dagger a_{i+1} \tag{46}$$

$$a_i^\dagger a_{i+1} - a_i a_{i+1}^\dagger \qquad i(a_i^\dagger a_{i+1} + a_i a_{i+1}^\dagger) \tag{47}$$

$$a_i^\dagger a_{i+1}^\dagger - a_i a_{i+1} \qquad i(a_i^\dagger a_{i+1}^\dagger + a_i a_{i+1}) \tag{48}$$

Each can be written in terms of the Majorana Fermion operators

$$H = \frac{i}{4} \sum_{jk} \alpha_{jk} c_j c_k, \tag{49}$$

where $\alpha$ is a real, antisymmetric $2n \times 2n$ matrix.

# What is left over? (2)

## Type 1

- Preserve the vacuum
- $[\alpha, J] = 0$

## Type 2

- Do not preserve the vacuum
- $\{\alpha, J\} = 0$

### Lemma (ORS)

If $\alpha$ is a real antisymmetric matrix and $\{\alpha, J\} = 0$ then there exists a zero-preserving FLO unitary $U$ such that

$$U \exp\left(\sum_{jk} \alpha_{jk} c_j c_k\right) U^\dagger = \prod_i \exp\left(\lambda_i \left(c_{4i} c_{4i+2} - c_{4i+1} c_{4i+3}\right)\right). \tag{50}$$

# FLO CH-form?

## Conjecture

Any Gaussian Fermionic state $|\psi\rangle$ may be written in the form

$$|\psi\rangle = \omega U_0 \left[\prod_j \exp(\lambda_j (c_{4j}c_{4j+2} - c_{4j+1}c_{4j+3}))\right] |0\rangle \tag{51}$$

$$= \omega U_0 \left[\prod_j \left(\cos(2\lambda_j)\, I + \sin(2\lambda_j)\, a^\dagger_{2j} a^\dagger_{2j+1}\right)\right] |0\rangle, \tag{52}$$

for $\omega \in \mathbb{C}$, $\lambda_j \in \mathbb{R}$ and $U_0|0\rangle = |0\rangle$.

# Other projects

## In progress

- Verification, validation & characterisation of NISQ computers
- Quantum machine learning
- Improvements to Clifford+T algorithm

## Not really in progress

Other efficiently simulable subtheories

- Low entanglement - Schrödinger-Feynmann simulators
- Low qubit number - Statevector simulators

# Thank you for your time