Three quantum learning algorithms

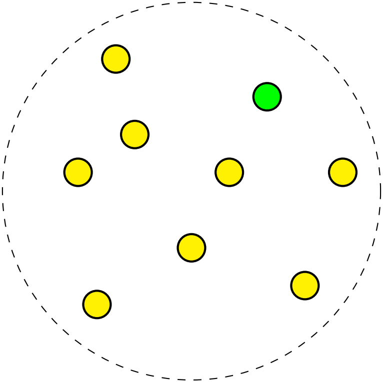
Ashley Montanaro

Talk based on joint work with Andris Ambainis and ongoing joint work with Scott Aaronson, David Chen, Daniel Gottesman and Vincent Liew.

18 January 2013

EPSRC Engineering and Physical Sciences Research Council

What is learning?



In this talk

Learning a set $S \equiv$ identifying an arbitrary, unknown object picked from S.

This talk

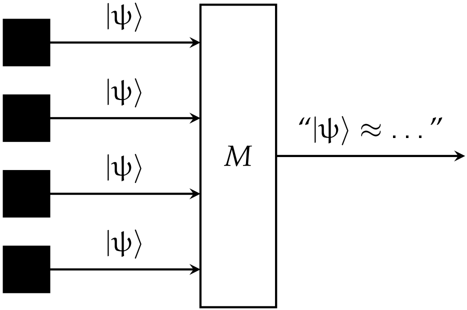
> “A little learning is a dangerous thing;
> drink deep, or taste not the Pierian spring:
> there shallow draughts intoxicate the brain,
> and drinking largely sobers us again.”
>
> — Alexander Pope

On this principle, I'll talk about three optimal quantum algorithms for learning an unknown...

- ...stabilizer state;
- ...low-degree multilinear polynomial;
- ...bit-string given access to “wildcard” queries.

Learning quantum states

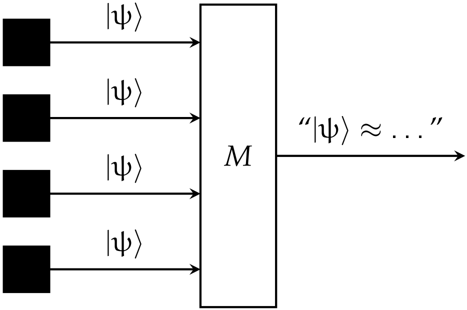
Consider the basic task of quantum state estimation.



- Given the ability to produce copies of an unknown n-qubit quantum state $|\psi\rangle$, we would like to estimate $|\psi\rangle$.
- Standard quantum state tomography uses $2^{\Theta(n)}$ copies of $|\psi\rangle$ to achieve constant fidelity.
- Can we do better?

Learning quantum states

Consider the basic task of quantum state estimation.



- To achieve constant fidelity between our guess and $|\psi\rangle$, we need $2^{\Omega(n)}$ copies of $|\psi\rangle$ [Bruss and Macchiavello '98].
- In order to determine $|\psi\rangle$ efficiently (using $\text{poly}(n)$ measurements) we must restrict to classes of states which have efficient descriptions, or change the problem.

Learning quantum states

Some examples where this has been achieved:

- [Cramer et al '11] give an efficient algorithm for learning matrix product states.
- [Aaronson '06] introduces "pretty good tomography": relax to attempting to predict the outcomes of "most" measurements on the state.
- [Flammia and Liu '11] and [da Silva et al '11] give efficient algorithms for certifying the production of certain states.

Learning stabilizer states

Today I'll talk about a learning algorithm for another important class of quantum states with efficient descriptions: stabilizer states.

- $|\psi\rangle$ is a stabilizer state of n qubits if there exists a subgroup G of 2^n pairwise commuting Pauli matrices (with phases) such that $M|\psi\rangle = |\psi\rangle$ for all $M \in G$.
- Examples include W states, cluster states, states occurring in quantum error-correcting codes, …

A stabilizer state of n qubits is completely specified by the identities of the elements of its stabilizer (n Pauli matrices on n qubits). There are $2^{\Theta(n^2)}$ stabilizer states of n qubits.

Learning stabilizer states

Theorem

There is a quantum algorithm which learns an unknown stabilizer state $|\psi\rangle$ given access to $O(n)$ copies of $|\psi\rangle$. The algorithm runs in time $O(n^3)$.

Notes on this result:

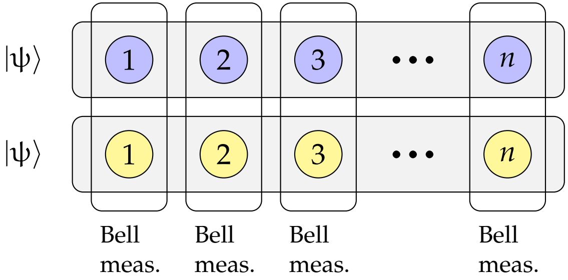
- By Holevo's theorem, this is optimal in terms of the scaling of the number of copies of $|\psi\rangle$ used.
- Any algorithm for learning stabilizer states requires $\Omega(n^2)$ time just to write the output.

The algorithm

The algorithm is based on the following subroutine.

Bell sampling

1. Create two copies of $|\psi\rangle$.
2. Measure each pair of qubits of $|\psi\rangle^{\otimes 2}$ in the Bell basis.



Learning stabilizer states

- For $z, x \in \{0,1\}$, write $\sigma_{zx} := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}^z \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}^x$.
- For $s \in \{0,1\}^{2n}$, write

$$\sigma_s := \sigma_{s_1 s_2} \otimes \cdots \otimes \sigma_{s_{2n-1} s_{2n}}.$$

Fact

Let $|\psi\rangle$ be a state of n qubits. Performing Bell sampling on $|\psi\rangle^{\otimes 2}$ returns outcome s with probability

$$p_\psi(s) := \frac{|\langle\psi|\sigma_s|\psi^*\rangle|^2}{2^n}.$$

Bell sampling and stabilizer states

- Up to an overall phase every stabilizer state $|\psi\rangle$ can be written in the form

$$|\psi\rangle = \frac{1}{\sqrt{|A|}} \sum_{x \in A} i^{\ell(x)} (-1)^{q(x)} |x\rangle,$$

  where A is an affine subspace of $\mathbb{F}_2^n$, and $\ell, q : \{0,1\}^n \to \{0,1\}$ are linear and quadratic (respectively) polynomials over $\mathbb{F}_2$ [Dehaene and Moor '02].

- As ℓ is linear, $\ell(x) = s \cdot x$ for some $s \in \{0,1\}^n$.

- So $(-1)^{\ell(x)} = \prod_{i \in S} (-1)^{x_i}$ for some $S \subseteq [n]$.

- Hence

$$|\psi^*\rangle = \sigma_{10}^{\otimes S} |\psi\rangle.$$

Bell sampling and stabilizer states

- If we perform Bell sampling on $|\psi\rangle^{\otimes 2}$, we receive outcome t with probability

$$\frac{|\langle\psi|\sigma_t|\psi^*\rangle|^2}{2^n} = \frac{|\langle\psi|\sigma_t \sigma_{10}^{\otimes S}|\psi\rangle|^2}{2^n}.$$

- Let G stabilize $|\psi\rangle$ and let T denote the set of strings $t \in \{0,1\}^{2n}$ such that $\sigma_t \in G$, up to a phase. Then T is an n-dimensional linear subspace of $\mathbb{F}_2^{2n}$.

- So Bell sampling gives an outcome r which is uniformly distributed on the set $\{t \oplus s : t \in T\}$ for some $s \in \{0,1\}^{2n}$.

Bell sampling and stabilizer states

- For any two such outcomes r_1, r_2, the sum $r_1 \oplus r_2$ is uniformly distributed in T.
  - In order to find a basis for T, we can therefore produce $k+1$ Bell samples $r_0, r_1, \ldots, r_k$ and consider the uniformly random elements of T given by $r_1 \oplus r_0, r_2 \oplus r_0, \ldots, r_k \oplus r_0$.
  - If the dimension of the subspace of $\mathbb{F}_2^{2n}$ spanned by these vectors is n, any basis of this subspace is a basis for T.
- Although T does not contain information about phases, determining T suffices to uniquely determine $|\psi\rangle$.
  - Once we have found a basis for T, we can measure $|\psi\rangle$ in the eigenbasis of each corresponding Pauli matrix M to decide whether $M|\psi\rangle = |\psi\rangle$ or $M|\psi\rangle = -|\psi\rangle$.

Learning stabilizer states

The algorithm

1. Set $S = \emptyset$.
2. Create two copies of $|\psi\rangle$ and perform Bell sampling, obtaining outcome r_0.
3. Repeat the following $2n$ times:
    1. Create two copies of $|\psi\rangle$ and perform Bell sampling, obtaining outcome r.
    2. Add $r \oplus r_0$ to S.
4. Determine a basis for S; call this basis B.
5. For each element of B, measure a copy of $|\psi\rangle$ in the eigenbasis of the corresponding Pauli matrix M to determine whether $M|\psi\rangle = |\psi\rangle$ or $M|\psi\rangle = -|\psi\rangle$.

Summary of learning stabilizer states

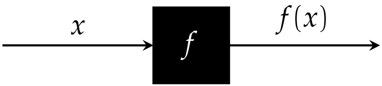
- The algorithm uses $O(n)$ copies of $|\psi\rangle$. Time complexity is dominated by finding a basis for S ($O(n^3)$ time or better).

- The algorithm fails (i.e. does not identify $|\psi\rangle$) if each of the $2n$ samples $r \oplus r_0$ lies in a subspace of T of dimension at most $n-1$. This occurs with probability at most 2^{-n}.

- We also have an alternative algorithm which uses $\Theta(n^2)$ copies of $|\psi\rangle$ but only makes single-copy Pauli measurements.

Learning classical oracles

Consider the following purely classical problem.



- We are given access to a function $f : \{0,1\}^n \to \{0,1\}$. We would like to identify f.
- If f is arbitrary, we need 2^n queries (uses of f).
- If f is picked from a known set $\mathcal{F}$, we need at least $\log_2 |\mathcal{F}|$ queries.
- We say that $\mathcal{F}$ can be learned using t queries if any function $f \in \mathcal{F}$ can be identified with t uses of f (perhaps allowing some probability of error).

Learning classical oracles on a quantum computer

- On a quantum computer, we have the ability to query f in superposition, i.e. to perform the map

$$|x\rangle|z\rangle \mapsto |x\rangle|z + f(x)\rangle.$$

- One of the oldest results in quantum computing: the Bernstein-Vazirani algorithm [Bernstein and Vazirani '97].

Theorem (Bernstein and Vazirani)

The class of linear functions $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be learned with certainty using 1 quantum query.

f is linear if $f(x+y) = f(x) + f(y)$; equivalently, $f(x) = \ell \cdot x$ for some $\ell \in \mathbb{F}_2^n$.

Learning multilinear polynomials

$f : \mathbb{F}_q^n \to \mathbb{F}_q$ is a degree d multilinear polynomial:

$$f(x) = \sum_{S \subseteq [n], |S| \leqslant d} \alpha_S \prod_{i \in S} x_i$$

for some coefficients $\alpha_S \in \mathbb{F}_q$, where $[n] := \{1, \ldots, n\}$.

- Note that for $S = \emptyset$ we define $\prod_{i \in S} x_i = 1$.
- For example, any multilinear polynomial of degree 3 can be written as

$$f(x) = \alpha_\emptyset + \sum_i \alpha_{\{i\}} x_i + \sum_{i<j} \alpha_{\{i,j\}} x_i x_j + \sum_{i<j<k} \alpha_{\{i,j,k\}} x_i x_j x_k.$$

- In the important special case $q = 2$ (boolean functions), every polynomial is multilinear.
- The set of degree d polynomials over $\mathbb{F}_2$ are known as the binary Reed-Muller code of order d.

Learning multilinear polynomials

Fact

The class of degree d multilinear polynomials in n variables over $\mathbb{F}_q$ can be learned using $O(n^d)$ classical queries, and this is optimal.

- Upper bound: It suffices to query $f(x)$ for all strings $x \in \mathbb{F}_q^n$ that contain only 0 and 1, and such that $|x| \leqslant d$.
- Lower bound: there are $q^{\Theta(n^d)}$ distinct multilinear degree d polynomials of n variables over $\mathbb{F}_q$; each classical query to f only provides $\log_2 q$ bits of information.

Learning multilinear polynomials

Theorem

The class of degree d multilinear polynomials in n variables over $\mathbb{F}_q$ can be learned exactly using $O(n^{d-1})$ quantum queries, and this is optimal.

Notes:

- The lower bound follows from Holevo's theorem.
- The Bernstein-Vazirani algorithm is the case $q = 2$, $d = 1$.
- Rötteler previously gave a bounded-error quantum algorithm for the case $q = 2$, $d = 2$ [Rötteler '09].
- A quantum algorithm for estimating a quadratic form over the reals had previously been given by Jordan [Jordan '08].

The algorithm

We use the following lemma [de Beaudrap et al '02, van Dam et al '02].

Lemma 1

Let $f : \mathbb{F}_q^n \to \mathbb{F}_q$ be linear, and let $g : \mathbb{F}_q^n \to \mathbb{F}_q$ be the function $g(x) = f(x) + \beta$ for some constant $\beta \in \mathbb{F}_q$. Then f can be determined exactly using one quantum query to g.

- Proof: query f in superposition and use the QFT.

The algorithm

For $S \subseteq [n]$, $|S| = k$, define

$$f_S(x) = \sum_{\beta_1,\dots,\beta_k \in \{0,1\}} (-1)^{k-\sum_{i=1}^k \beta_i} f\left(x + \sum_{j=1}^{k} \beta_j e_{S_j}\right).$$

Here e_i is the i'th element in the standard basis for $\mathbb{F}_q^n$; the inner sum is over $\mathbb{F}_q^n$ and the outer sum is over $\mathbb{F}_q$.

- For example, if $S = \{1, 2\}$:

$$f_S(x) = f(x) - f(x + e_1) - f(x + e_2) + f(x + e_1 + e_2).$$

- A query to f_S can be simulated using 2^k queries to f.
- Define the discrete derivative of f in direction $i \in [n]$ as

$$(\Delta_i f)(x) := f(x + e_i) - f(x).$$

- Then $f_S(x) = (\Delta_{S_1} \Delta_{S_2} \dots \Delta_{S_k} f)(x)$.

The algorithm

We will be interested in querying f_S for sets S of size $d-1$. In this case, we have the following characterisation for multilinear polynomials f.

Lemma 2

Let $f : \mathbb{F}_q^n \to \mathbb{F}_q$ be a multilinear polynomial of degree d with expansion

$$f(x) = \sum_{T \subseteq [n], |T| \leqslant d} \alpha_T \prod_{i \in T} x_i.$$

Then, for any S such that $|S| = d-1$,

$$f_S(x) = \alpha_S + \sum_{k \notin S} \alpha_{S \cup \{k\}} x_k.$$

Proof: follows easily from expressing f in terms of discrete derivatives.

Learning all the degree d terms

The algorithm

foreach $S \subseteq [n]$ *such that* $|S| = d - 1$ **do**
  Use one query to f_S to learn $\alpha_{S \cup \{k\}}$, for all $k \notin S$;
end
Output the function $f_d(x) = \sum_{S \subseteq [n], |S| = d} \alpha_S \prod_{i \in S} x_i$;

Proof of correctness:

- By Lemma 2, for any S such that $|S| = d - 1$, knowledge of the degree 1 component of f_S is sufficient to determine $\alpha_{S \cup \{k\}}$ for all $k \notin S$.

Learning all the degree d terms

The algorithm

foreach $S \subseteq [n]$ *such that* $|S| = d - 1$ **do**
  Use one query to f_S to learn $\alpha_{S \cup \{k\}}$, for all $k \notin S$;
end
Output the function $f_d(x) = \sum_{S \subseteq [n], |S| = d} \alpha_S \prod_{i \in S} x_i$;

Proof of correctness:

- By Lemma 2, for any S such that $|S| = d - 1$, knowledge of the degree 1 component of f_S is sufficient to determine $\alpha_{S \cup \{k\}}$ for all $k \notin S$.
- So knowing the degree 1 part of f_S for all $S \subseteq [n]$ such that $|S| = d - 1$ is sufficient to completely determine all degree d coefficients of f.

Learning all the degree d terms

The algorithm

foreach $S \subseteq [n]$ *such that* $|S| = d - 1$ **do**

  Use one query to f_S to learn $\alpha_{S \cup \{k\}}$, for all $k \notin S$;

end

Output the function $f_d(x) = \sum_{S \subseteq [n], |S| = d} \alpha_S \prod_{i \in S} x_i$;

Proof of correctness:

- By Lemma 1, for any S with $|S| = d - 1$, the degree 1 component of f_S can be determined with one quantum query to f_S.

Learning all the degree d terms

The algorithm

foreach $S \subseteq [n]$ *such that* $|S| = d - 1$ **do**

  Use one query to f_S to learn $\alpha_{S \cup \{k\}}$, for all $k \notin S$;

end

Output the function $f_d(x) = \sum_{S \subseteq [n], |S| = d} \alpha_S \prod_{i \in S} x_i$;

Proof of correctness:

- By Lemma 1, for any S with $|S| = d - 1$, the degree 1 component of f_S can be determined with one quantum query to f_S.
- So the algorithm completely determines the degree d component of f using $\binom{n}{d-1}$ queries to f_S, each of which uses 2^{d-1} queries to f.

Finishing up

- Once the degree d component of f has been learned, f can be reduced to a degree $d-1$ polynomial by crossing out the degree d part whenever the oracle for f is called.
- Whenever the oracle is called on x, we subtract $f_d(x)$ from the result (recall f_d is the degree d part of f), at no extra query cost.
- Inductively, f can be determined completely using

$$2^{d-1}\binom{n}{d-1} + 2^{d-2}\binom{n}{d-2} + \cdots + 2n + 1 + 1$$

  queries; the last query is to determine the constant term $\alpha_\emptyset$, which can be achieved by classically querying $f(0^n)$.
- The number of queries used is therefore $O(n^{d-1})$ for constant d.

Search with wildcards

- We are given an n-bit string x.

Search with wildcards

- We are given an n-bit string x.
- Our task is to determine x using the minimum expected number of queries.

Search with wildcards

- We are given an n-bit string x.
- Our task is to determine x using the minimum expected number of queries.
- The different possible queries are given by strings $s \in \{0, 1, *\}^n$. A query returns 1 if $x_i = s_i$ for all i such that $s_i \neq *$, and returns 0 otherwise.

Search with wildcards

- We are given an n-bit string x.
- Our task is to determine x using the minimum expected number of queries.
- The different possible queries are given by strings $s \in \{0, 1, *\}^n$. A query returns 1 if $x_i = s_i$ for all i such that $s_i \neq *$, and returns 0 otherwise.
- A generalisation of the simple model where we are allowed to query individual bits of x.

Search with wildcards

- We are given an n-bit string x.
- Our task is to determine x using the minimum expected number of queries.
- The different possible queries are given by strings $s \in \{0, 1, *\}^n$. A query returns 1 if $x_i = s_i$ for all i such that $s_i \neq *$, and returns 0 otherwise.
- A generalisation of the simple model where we are allowed to query individual bits of x.

Classically, we need n queries to determine x (each query gives one bit of information).

Search with wildcards

- We are given an n-bit string x.
- Our task is to determine x using the minimum expected number of queries.
- The different possible queries are given by strings $s \in \{0, 1, *\}^n$. A query returns 1 if $x_i = s_i$ for all i such that $s_i \neq *$, and returns 0 otherwise.
- A generalisation of the simple model where we are allowed to query individual bits of x.

Classically, we need n queries to determine x (each query gives one bit of information).

Theorem

Search with wildcards can be solved with $O(\sqrt{n})$ quantum queries on average.

Solving SWW

The solution to SWW is based on this claim:

Measurement Lemma

Fix $n \geqslant 1$ and, for any $0 \leqslant k \leqslant n$, set

$$|\psi_x^k\rangle := \frac{1}{\binom{n}{k}^{1/2}} \sum_{S \subseteq [n], |S|=k} |S\rangle |x_S\rangle,$$

where $|x_S\rangle := \bigotimes_{i \in S} |x_i\rangle$. Then, for any $k = n - O(\sqrt{n})$, there is a quantum measurement (POVM) which, on input $|\psi_x^k\rangle$, outputs $\widetilde{x}$ such that the expected Hamming distance $d(x, \widetilde{x})$ is $O(1)$.

Solving SWW

The solution to SWW is based on this claim:

Measurement Lemma

Fix $n \geqslant 1$ and, for any $0 \leqslant k \leqslant n$, set

$$|\psi_x^k\rangle := \frac{1}{\binom{n}{k}^{1/2}} \sum_{S \subseteq [n], |S|=k} |S\rangle |x_S\rangle,$$

where $|x_S\rangle := \bigotimes_{i \in S} |x_i\rangle$. Then, for any $k = n - O(\sqrt{n})$, there is a quantum measurement (POVM) which, on input $|\psi_x^k\rangle$, outputs $\widetilde{x}$ such that the expected Hamming distance $d(x, \widetilde{x})$ is $O(1)$.

This is surprising because the equivalent classical statement is not true!

Solving SWW

The solution to SWW is based on this claim:

Measurement Lemma

Fix $n \geqslant 1$ and, for any $0 \leqslant k \leqslant n$, set

$$|\psi_x^k\rangle := \frac{1}{\binom{n}{k}^{1/2}} \sum_{S \subseteq [n], |S|=k} |S\rangle |x_S\rangle,$$

where $|x_S\rangle := \bigotimes_{i \in S} |x_i\rangle$. Then, for any $k = n - O(\sqrt{n})$, there is a quantum measurement (POVM) which, on input $|\psi_x^k\rangle$, outputs $\widetilde{x}$ such that the expected Hamming distance $d(x, \widetilde{x})$ is $O(1)$.

This is surprising because the equivalent classical statement is not true!

Why does this let us solve SWW?

The measurement lemma ⇒ solving SWW

- Our algorithm for SWW repeatedly uses the lemma to learn $O(\sqrt{n})$ bits of x at a time in superposition.

The measurement lemma ⇒ solving SWW

- Our algorithm for SWW repeatedly uses the lemma to learn $O(\sqrt{n})$ bits of x at a time in superposition.
- Imagine we have $|\psi_x^k\rangle$. For $k' > k$, this can be mapped to

$$\sum_{S':S\subseteq[n],|S'|=k'} |S'\rangle \left(\sum_{S:S\subseteq S',|S|=k} |S\rangle|x_S\rangle \right) = \sum_{S:S\subseteq[n],|S|=k'} |S\rangle|\psi_{x_S}^k\rangle,$$

so if we can map $|\psi_{x_S}^k\rangle \mapsto |x_S\rangle$, we've made $|\psi_x^{k'}\rangle$.

The measurement lemma ⇒ solving SWW

- Our algorithm for SWW repeatedly uses the lemma to learn $O(\sqrt{n})$ bits of x at a time in superposition.
- Imagine we have $|\psi_x^k\rangle$. For $k' > k$, this can be mapped to

$$\sum_{S':S \subseteq [n], |S'|=k'} |S'\rangle \left(\sum_{S:S \subseteq S', |S|=k} |S\rangle |x_S\rangle \right) = \sum_{S:S \subseteq [n], |S|=k'} |S\rangle |\psi_{x_S}^k\rangle,$$

  so if we can map $|\psi_{x_S}^k\rangle \mapsto |x_S\rangle$, we've made $|\psi_x^{k'}\rangle$.

- By the lemma, we can do this when $k = k' - O(\sqrt{k'})$.

The measurement lemma $\Rightarrow$ solving SWW

- Our algorithm for SWW repeatedly uses the lemma to learn $O(\sqrt{n})$ bits of x at a time in superposition.

- Imagine we have $|\psi_x^k\rangle$. For $k' > k$, this can be mapped to

$$\sum_{S':S\subseteq[n],|S'|=k'} |S'\rangle \left(\sum_{S:S\subseteq S',|S|=k} |S\rangle|x_S\rangle \right) = \sum_{S:S\subseteq[n],|S|=k'} |S\rangle|\psi_{x_S}^k\rangle,$$

  so if we can map $|\psi_{x_S}^k\rangle \mapsto |x_S\rangle$, we've made $|\psi_x^{k'}\rangle$.

- By the lemma, we can do this when $k = k' - O(\sqrt{k'})$.

- After each measurement, an expected $O(1)$ bits are incorrect.

The measurement lemma $\Rightarrow$ solving SWW

- Our algorithm for SWW repeatedly uses the lemma to learn $O(\sqrt{n})$ bits of x at a time in superposition.
- Imagine we have $|\psi_x^k\rangle$. For $k' > k$, this can be mapped to

$$\sum_{S':S\subseteq[n],|S'|=k'} |S'\rangle \left(\sum_{S:S\subseteq S',|S|=k} |S\rangle|x_S\rangle \right) = \sum_{S:S\subseteq[n],|S|=k'} |S\rangle|\psi_{x_S}^k\rangle,$$

  so if we can map $|\psi_{x_S}^k\rangle \mapsto |x_S\rangle$, we've made $|\psi_x^{k'}\rangle$.
- By the lemma, we can do this when $k = k' - O(\sqrt{k'})$.
- After each measurement, an expected $O(1)$ bits are incorrect.
- How to fix these?

Combinatorial group testing (CGT)

Proposed by [Dorfman '43] as a means of "weeding out all syphilitic men called up for induction".

Combinatorial group testing (CGT)

Proposed by [Dorfman '43] as a means of "weeding out all syphilitic men called up for induction".

The abstract problem is:

- We have a set of n items $x_1, \ldots, x_n \in \{0, 1\}$.

Combinatorial group testing (CGT)

Proposed by [Dorfman '43] as a means of "weeding out all syphilitic men called up for induction".

The abstract problem is:

- We have a set of n items $x_1, \ldots, x_n \in \{0, 1\}$.
- At most $k \ll n$ items x_i are special and have $x_i = 1$.

Combinatorial group testing (CGT)

Proposed by [Dorfman '43] as a means of "weeding out all syphilitic men called up for induction".

The abstract problem is:

- We have a set of n items $x_1, \ldots, x_n \in \{0, 1\}$.
- At most $k \ll n$ items x_i are special and have $x_i = 1$.
- We are allowed to query any subset $S \subseteq [n] := \{1, \ldots, n\}$. A query returns 1 if any items in S are special.

Combinatorial group testing (CGT)

Proposed by [Dorfman '43] as a means of "weeding out all syphilitic men called up for induction".

The abstract problem is:

- We have a set of n items $x_1, \dots, x_n \in \{0, 1\}$.
- At most $k \ll n$ items x_i are special and have $x_i = 1$.
- We are allowed to query any subset $S \subseteq [n] := \{1, \dots, n\}$. A query returns 1 if any items in S are special.
- We want to output the identities of all of the special items using the minimal number of queries.

Combinatorial group testing (CGT)

Proposed by [Dorfman '43] as a means of "weeding out all syphilitic men called up for induction".

The abstract problem is:

- We have a set of n items $x_1, \dots, x_n \in \{0, 1\}$.
- At most $k \ll n$ items x_i are special and have $x_i = 1$.
- We are allowed to query any subset $S \subseteq [n] := \{1, \dots, n\}$. A query returns 1 if any items in S are special.
- We want to output the identities of all of the special items using the minimal number of queries.

In particular, we would like to minimise the dependence on n.

Classical results

- The number of classical queries required to solve CGT is $\Theta(k \log(n/k))$.
  - Lower bound: information-theoretic argument.
  - Upper bound: (essentially) binary search.

Classical results

- The number of classical queries required to solve CGT is $\Theta(k \log(n/k))$.
  - Lower bound: information-theoretic argument.
  - Upper bound: (essentially) binary search.

- If we restrict to nonadaptive queries, the bound becomes essentially $\Theta(\min\{k^2 \log n, n\})$.

Classical results

- The number of classical queries required to solve CGT is $\Theta(k \log(n/k))$.
  - Lower bound: information-theoretic argument.
  - Upper bound: (essentially) binary search.

- If we restrict to nonadaptive queries, the bound becomes essentially $\Theta(\min\{k^2 \log n, n\})$.

- Many applications known: molecular biology, data streaming algorithms, compressed sensing, pattern matching in strings, …

Classical results

- The number of classical queries required to solve CGT is $\Theta(k \log(n/k))$.
  - Lower bound: information-theoretic argument.
  - Upper bound: (essentially) binary search.

- If we restrict to nonadaptive queries, the bound becomes essentially $\Theta(\min\{k^2 \log n, n\})$.

- Many applications known: molecular biology, data streaming algorithms, compressed sensing, pattern matching in strings, ...

- See the book "Combinatorial Group Testing and Its Applications" [Du and Hwang '00] for more.

Quantum algorithms for CGT

The $k = 1$ case

If $k = 1$, CGT can be solved exactly using one quantum query.

Quantum algorithms for CGT

The $k = 1$ case

If $k = 1$, CGT can be solved exactly using one quantum query.

Basic idea:

- To learn x, suffices to be able to compute the function $x \cdot s = \bigoplus_i x_i s_i$ for arbitrary $s \in \{0,1\}^n$ (as with e.g. the quantum oracle interrogation algorithm of [van Dam '98]).

Quantum algorithms for CGT

The $k = 1$ case

If $k = 1$, CGT can be solved exactly using one quantum query.

Basic idea:

- To learn x, suffices to be able to compute the function $x \cdot s = \bigoplus_i x_i s_i$ for arbitrary $s \in \{0,1\}^n$ (as with e.g. the quantum oracle interrogation algorithm of [van Dam '98]).
- In the CGT problem, we have access to an oracle which computes $f(s) = \bigvee_i x_i s_i$ for arbitrary $s \in \{0,1\}^n$. But if $|x| \leqslant 1$, then for any s, $\bigvee_i x_i s_i = x \cdot s$.

Quantum algorithms for CGT

The $k = 1$ case

If $k = 1$, CGT can be solved exactly using one quantum query.

Quantum algorithms for CGT

The $k = 1$ case

If $k = 1$, CGT can be solved exactly using one quantum query.

1. Create the state $\frac{1}{\sqrt{2^{n+1}}} \sum_{s \in \{0,1\}^n} |s\rangle(|0\rangle - |1\rangle)$.

Quantum algorithms for CGT

The $k = 1$ case

If $k = 1$, CGT can be solved exactly using one quantum query.

1. Create the state $\frac{1}{\sqrt{2^{n+1}}} \sum_{s \in \{0,1\}^n} |s\rangle(|0\rangle - |1\rangle)$.
2. Apply the oracle to create the state

$$\begin{aligned} & \frac{1}{\sqrt{2^{n+1}}} \sum_{s \in \{0,1\}^n} (-1)^{\bigvee_i s_i x_i} |s\rangle(|0\rangle - |1\rangle) \\ = \quad & \frac{1}{\sqrt{2^{n+1}}} \sum_{s \in \{0,1\}^n} (-1)^{s \cdot x} |s\rangle(|0\rangle - |1\rangle). \end{aligned}$$

Quantum algorithms for CGT

The $k = 1$ case

If $k = 1$, CGT can be solved exactly using one quantum query.

1. Create the state $\frac{1}{\sqrt{2^{n+1}}} \sum_{s \in \{0,1\}^n} |s\rangle(|0\rangle - |1\rangle)$.
2. Apply the oracle to create the state

$$\begin{aligned} & \frac{1}{\sqrt{2^{n+1}}} \sum_{s \in \{0,1\}^n} (-1)^{\bigvee_i s_i x_i} |s\rangle(|0\rangle - |1\rangle) \\ = \quad & \frac{1}{\sqrt{2^{n+1}}} \sum_{s \in \{0,1\}^n} (-1)^{s \cdot x} |s\rangle(|0\rangle - |1\rangle). \end{aligned}$$

3. Apply Hadamard gates to each qubit of the first register and measure to obtain x.

Generalising this idea to arbitrary k

Theorem

CGT can be solved using $O(k)$ quantum queries on average.

Generalising this idea to arbitrary k

Theorem

CGT can be solved using $O(k)$ quantum queries on average.

- Construct $S \subseteq [n]$ by including each $i \in [n]$ with prob. $1/k$.

Generalising this idea to arbitrary k

Theorem

CGT can be solved using $O(k)$ quantum queries on average.

- Construct $S \subseteq [n]$ by including each $i \in [n]$ with prob. $1/k$.
- Run the $k = 1$ algorithm on the subset of bits in S.

Generalising this idea to arbitrary k

Theorem

CGT can be solved using $O(k)$ quantum queries on average.

- Construct $S \subseteq [n]$ by including each $i \in [n]$ with prob. $1/k$.
- Run the $k = 1$ algorithm on the subset of bits in S.
- If S contains exactly one 1 bit at position i, which will occur with probability at least $(1 - 1/k)^{k-1} \geqslant 1/e$, we are guaranteed to learn i.

Generalising this idea to arbitrary k

Theorem

CGT can be solved using $O(k)$ quantum queries on average.

- Construct $S \subseteq [n]$ by including each $i \in [n]$ with prob. $1/k$.
- Run the $k = 1$ algorithm on the subset of bits in S.
- If S contains exactly one 1 bit at position i, which will occur with probability at least $(1 - 1/k)^{k-1} \geqslant 1/e$, we are guaranteed to learn i.
- We can check whether the index $\tilde{i}$ we received really is a 1 by making one more query to index $\tilde{i}$.

Generalising this idea to arbitrary k

Theorem

CGT can be solved using $O(k)$ quantum queries on average.

- Construct $S \subseteq [n]$ by including each $i \in [n]$ with prob. $1/k$.
- Run the $k = 1$ algorithm on the subset of bits in S.
- If S contains exactly one 1 bit at position i, which will occur with probability at least $(1 - 1/k)^{k-1} \geqslant 1/e$, we are guaranteed to learn i.
- We can check whether the index $\tilde{i}$ we received really is a 1 by making one more query to index $\tilde{i}$.
- Following each successful query, we reduce k by 1 and exclude the bit that we just learned from future queries.

Generalising this idea to arbitrary k

Theorem

CGT can be solved using $O(k)$ quantum queries on average.

- Construct $S \subseteq [n]$ by including each $i \in [n]$ with prob. $1/k$.
- Run the $k = 1$ algorithm on the subset of bits in S.
- If S contains exactly one 1 bit at position i, which will occur with probability at least $(1 - 1/k)^{k-1} \geqslant 1/e$, we are guaranteed to learn i.
- We can check whether the index $\tilde{i}$ we received really is a 1 by making one more query to index $\tilde{i}$.
- Following each successful query, we reduce k by 1 and exclude the bit that we just learned from future queries.
- In order to learn x completely, the expected overall number of queries used is $O(k)$.

Back to search with wildcards

- When we measure $|\psi_x^k\rangle$, we get an outcome $\widetilde{x}$ such that $d(\widetilde{x}, x) = O(1)$.
- We want to determine x, which is equivalent to determining $\widetilde{x} \oplus x$, a string of Hamming weight $O(1)$.
- A wildcard query corresponding to $S \subseteq [n]$ and $\widetilde{x}_S \oplus y$, $y \in \{0, 1\}^{|S|}$, returns 1 iff all bits of $\widetilde{x}_S$ are correct.
- So we can use the algorithm for CGT to find, and correct, all incorrect bits in $O(1)$ queries.

Proving the measurement lemma

We finally need to prove we can distinguish the $|\psi_x^k\rangle$ states. We use the pretty good measurement (PGM).

Proving the measurement lemma

We finally need to prove we can distinguish the $|\psi_x^k\rangle$ states. We use the pretty good measurement (PGM).

Lemma

The probability that the PGM outputs y on input $|\psi_x^k\rangle$ is precisely $(\sqrt{G})_{xy}^2$, where

$$G_{xy} = \langle\psi_x^k|\psi_y^k\rangle = \frac{1}{\binom{n}{k}} \sum_{S\subseteq[n],|S|=k} [x_S = y_S] = \frac{\binom{n-d(x,y)}{k}}{\binom{n}{k}}.$$

Proving the measurement lemma

We finally need to prove we can distinguish the $|\psi_x^k\rangle$ states. We use the pretty good measurement (PGM).

Lemma

The probability that the PGM outputs y on input $|\psi_x^k\rangle$ is precisely $(\sqrt{G})^2_{xy}$, where

$$G_{xy} = \langle \psi_x^k | \psi_y^k \rangle = \frac{1}{\binom{n}{k}} \sum_{S \subseteq [n], |S|=k} [x_S = y_S] = \frac{\binom{n-d(x,y)}{k}}{\binom{n}{k}}.$$

- We want to bound $D_k := \sum_{y \in \{0,1\}^n} d(x,y) (\sqrt{G}_{xy})^2$.

Proving the measurement lemma

We finally need to prove we can distinguish the $|\psi_x^k\rangle$ states. We use the pretty good measurement (PGM).

Lemma

The probability that the PGM outputs y on input $|\psi_x^k\rangle$ is precisely $(\sqrt{G})^2_{xy}$, where

$$G_{xy} = \langle\psi_x^k|\psi_y^k\rangle = \frac{1}{\binom{n}{k}} \sum_{S\subseteq[n],|S|=k} [x_S = y_S] = \frac{\binom{n-d(x,y)}{k}}{\binom{n}{k}}.$$

- We want to bound $D_k := \sum_{y\in\{0,1\}^n} d(x,y)(\sqrt{G}_{xy})^2$.
- G_{xy} depends only on $x \oplus y$, so G is diagonalised by the Fourier transform over $\mathbb{Z}_2^n$ and D_k does not depend on x.

Proving the measurement lemma

We finally need to prove we can distinguish the $|\psi_x^k\rangle$ states. We use the pretty good measurement (PGM).

Lemma

The probability that the PGM outputs y on input $|\psi_x^k\rangle$ is precisely $(\sqrt{G})^2_{xy}$, where

$$G_{xy} = \langle \psi_x^k | \psi_y^k \rangle = \frac{1}{\binom{n}{k}} \sum_{S \subseteq [n], |S|=k} [x_S = y_S] = \frac{\binom{n-d(x,y)}{k}}{\binom{n}{k}}.$$

- We want to bound $D_k := \sum_{y \in \{0,1\}^n} d(x,y)(\sqrt{G}_{xy})^2$.
- G_{xy} depends only on $x \oplus y$, so G is diagonalised by the Fourier transform over $\mathbb{Z}_2^n$ and D_k does not depend on x.
- D_k can be upper bounded using Fourier duality and some combinatorics.

Summary

We can learn…

- … n-qubit stabilizer states with $O(n)$ copies;
- … degree d n-variate multilinear polynomials with $O(n^{d-1})$ queries;
- … n-bit strings with $O(\sqrt{n})$ wildcard queries.

Open problems:

- Determine the quantum query complexity of CGT.
- Other applications of SWW! A possible example: testing juntas.

Thanks!

Some further reading:

- The algorithm for learning multilinear polynomials: **arXiv:1105.3310**
- The algorithm for search with wildcards: **arXiv:1210.1148** (joint work with Andris Ambainis)
- The algorithm for learning stabilizer states: **arXiv:13??.????** (joint work with Scott Aaronson, David Chen, Daniel Gottesman and Vincent Liew)