Quantum Error Correcting Codes

Lecture Notes for the course:

Introduction to Quantum Computation

Lecturer: Dorit Aharonov

Scribe: Elad Hazan

1. Threshold Theorem

The main result of this session is the Threshold Theorem that asserts:

For every quantum circuit Q there exists a quantum circuit Q' that calculates the same function as Q (up to some error ε), is of size polynomial in the size of Q, and works under noise $\eta \le \eta_0$ (η_0 is a constant $\approx 10^{-6}$).

2. Polynomial Codes

2.1 Classical Polynomial Codes

These error-correcting codes use polynomials over a finite-field (the finite field is F_p -where p is a large prime) vector space $f : F^r \to F$ of degree $\le r$ in order to encode $r+1$ numbers inside a codeword of length m, where $r < m \le p$.

An encoding of characters $x_0,...,x_r$ is to take the unique polynomial q of degree $\le r$ such that $q(i) = x_i$ (and there is such a unique polynomial according to results from numerical analysis) and the codeword is: $q(0), q(1),..., q(m-1)$.

Because every two polynomials of degree $\le r$ can agree on at most d points (and this is fairly easy to prove), the hamming distance between any two code words is at least $m-r$.

This allows us to detect $m-r-1$ errors and to correct $\left\lfloor \frac{m-r-1}{2} \right\rfloor$ errors.

2.2 Polynomial Quantum Error Correcting Codes

First we define:

Qupit – a quantum state $\sum_i \alpha_i |i\rangle$ where every basic state $|i\rangle$ is a number $\in F_p$.

If we define:

$$B:|i\rangle \to |(i+1) \bmod p\rangle$$
$$P:|i\rangle \to \omega_p^i |i\rangle$$

Where as usual: $\omega_p = \exp(\frac{2\pi i}{p})$

Then, as explained in the previous lecture, the set of functions $\{B^x P^y : x, y \in F_p\}$ is a basis for all errors on one qupit (all linear combinations of bit-flip errors – B and phase errors – P). Also, like in the case of linear error-correcting codes, the Fourier Transform maps bit errors to phase errors and vice versa.

The coding itself:

Let $a \in F_p$. We choose $\alpha_1,...,\alpha_m \in F_p - \{0\}$ arbitrary elements in the finite field excluding zero.

Then a is encoded into:

$$|S_a\rangle = |S_a^r\rangle = \frac{1}{\sqrt{p^r}} \sum_{\substack{f(0)=a \\ \deg(f)\le r}} |f(\alpha_1),...,f(\alpha_m)\rangle$$

As explained previously, the representation of a polynomial of degree $\le r$ with m values allows one to correct $\left\lfloor \frac{m-r-1}{2} \right\rfloor$ bit-flip errors.

In order to correct phase errors as well, we first define:

$$|\tilde{S}_a\rangle = \frac{1}{\sqrt{p^{(m-r-1)/2}}} \sum_{\substack{f(0)=a \\ \deg(f)\le m-r-1}} |f(\alpha_1),...,f(\alpha_m)\rangle$$

(Which is same as before, except for the degree of the polynomials)

And the final encoding of a is:

$$|a\rangle \to |C_a\rangle = \frac{1}{\sqrt{p}} \sum_{b=0}^{p-1} \omega^{ab} |\tilde{S}_b\rangle$$

And using the shifting from bit-flip errors to phase errors property of the Fourier Transform, this code enables one to correct $\left\lfloor \frac{m-r-1}{2} \right\rfloor$ bit-flip errors and $\left\lfloor \frac{r+1}{2} \right\rfloor$ phase errors.

Altogether, the code can correct $\min\{\left\lfloor \frac{m-r-1}{2} \right\rfloor, \left\lfloor \frac{r+1}{2} \right\rfloor\}$ errors.

3. Quantum Fault Tolerant Computation

We now wish to use the quantum polynomial error correcting codes to construct fault tolerant quantum circuits.

For this purpose we will first show how to build fault tolerant gates. Overall we shall describe a universal set of fault-tolerant gates. Then we explain how to build a fault-tolerant error correction circuit.

In the next section we shall describe how to combine these ingredients in order to build fault tolerant quantum circuits, and prove the Threshold Theorem.

3.1 Bitwise Fault Tolerant Gates

Fault tolerant gates allow for n errors in the input to propagate into at most n errors in output. Bitwise gates are defined by applying a certain gate on every input qupit separately. Obviously, a single error in the gate can affect only one output qupit.

We now show implementations of several bitwise fault tolerant gates:

3.1.1 Gate 1: $|a\rangle \to |a+1(\mathrm{mod}\ p)\rangle$

The result of this gate is equivalent to the following action on the encoded states:

$$|S_a\rangle = \sum |f(\alpha_1),...,f(\alpha_m)\rangle \to \sum |f(\alpha_1)+1(\mathrm{mod}\ p),...,f(\alpha_m)+1(\mathrm{mod}\ p)\rangle = |S_{a+1}\rangle$$

The above process can be carried out for each bit in the encoding separately. Hence the limited-propagation property holds.

3.1.2 Gate 2: $|a\rangle \to |a+b(\mathrm{mod}\ p)\rangle$

The implementation of this gate so that error propagation is limited as required, is similar to the previous gate.

3.1.3 Gate 3 – Phase Shift: $|a\rangle \to \omega^a |a\rangle$

In the encoding domain the equivalent action is:

$$|S_a\rangle \to \omega^a |S_a\rangle$$

In order to achieve this, define $\{c_i\}$ as the correlation coefficients:

$$\forall f \in F[x], \deg(f) \le m-1, f(0) = \sum_{i=1}^{m} c_i f(\alpha_i)$$

(there is a unique solution to this equation list, because the dimension of the space of all polynomials of degree $\le m-1$ is m)

Now, the bitwise phase shift gate is simply to apply $|f(\alpha_i)\rangle \to \omega^{c_i}|f(\alpha_i)\rangle$ for every coefficient:

$$|S_a\rangle = \sum |f(\alpha_1),...,f(\alpha_m)\rangle \to \sum |\omega^{c_1} f(\alpha_1),...\omega^{c_m} f(\alpha_m)\rangle = \omega^{\sum_i c_i} |S_{a+1}\rangle = \omega^{f(0)} |S_{a+1}\rangle = \omega^a |S_{a+1}\rangle$$

That is, for each element in the encoding we apply a phase shift of ω^{c_i}, and the total phase shift is as required.

3.1.4 Gate 4: The Fourier Transform over Z_p: $|a\rangle \to \frac{1}{\sqrt{p}} \sum_{b \in F_p} \omega^{a \cdot b} |b\rangle$

The equivalent action in the encoded domain is (well, not exactly equivalent, notice that we transform a code of degree r to degree m-r-1):

$$|S_a^r\rangle \to \frac{1}{\sqrt{p}} \sum_{b \in F_p} \omega^{a \cdot b} |S_b^{m-r-1}\rangle$$

We now describe how to perform this procedure in a bitwise manner. Let $\{c_i\}$ be the correlation coefficients as in gate 3 description. Define:

1. $\omega_l \equiv \omega^{c_l}$ for each l from 1 to m.

2. $W(\omega_l)|a\rangle \equiv \frac{1}{\sqrt{p}} \sum_{b \in F_p} \omega_l^{a \cdot b} |b\rangle$

The procedure is simply to apply $W(\omega_l)$ for the l'th coefficient in the encoding:

$$|S_a\rangle \to W(\omega_1) \otimes W(\omega_2) \otimes ...I \otimes W(\omega_m) |S_a\rangle$$

Thus the new state is:

$$|S_a\rangle = \frac{1}{\sqrt{p^r}} \sum_{\substack{f(0)=a \\ \deg(f) \le r}} |f(\alpha_1),..., f(\alpha_m)\rangle \to$$

$$\frac{1}{\sqrt{p^{r+m}}} \sum_{b1,b2,...bm \in F} \sum_{\substack{f(0)=a \\ \deg(f) \le r}} \omega^{\sum_{l=1}^{m} c_l f(\alpha_l) b_l} |b_1,...,b_m\rangle \equiv |\alpha\rangle$$

For each string $b_1,...,b_m \in F_p$, associate the unique polynomial $b(x)$, which satisfies $b(\alpha_l) = b_l$ and has degree $\deg(b) \le m-1$. The exponent of w in the above equation can be written in a much simpler form when $b(x)$ is of degree $\deg(b) \le m-r-1$. For such $b(x)$, the polynomial $h(x) \equiv b(x) f(x)$ is of degree $\deg(h) \le m-1$ so:

$$\sum_{l=1}^{m} c_l f(\alpha_l) b(\alpha_l) = \sum_{l=1}^{m} c_l h(\alpha_l) = h(0) = f(0)b(0)$$

Hence, the sum over all $b(x)$ with $\deg(b) \le m-r-1$ in the equation gives:

$$\frac{1}{\sqrt{p^{r+m}}} \sum_{\substack{b1,b2,...bm\in F \\ \deg(b)\le m-r-1}} \sum_{\substack{f(0)=a \\ \deg(f)\le r}} \omega^{b(0)f(0)} \left|b_1,...,b_m\right\rangle =$$

$$\frac{1}{\sqrt{p^{m-r}}} \sum_{\substack{b1,b2,...bm\in F \\ \deg(b)\le m-r-1}} \omega^{b(0)a} \left|b_1,...,b_m\right\rangle =$$

$$\frac{1}{\sqrt{p}} \sum_{b\in F} \omega^{b\cdot a} \frac{1}{\sqrt{p^{m-r-1}}} \sum_{\substack{b1,b2,...bm\in F \\ \deg(b)\le m-r-1 \\ b(0)=b}} \left|b_1,...,b_m\right\rangle =$$

$$\frac{1}{\sqrt{p}} \omega^{b\cdot a} \sum_{b\in F} S_b^{m-r-1}$$

Now, we claim that the sum over the rest of the *b*'s (with degree higher then *m-r-1*) must vanish. The reason is that the norm of the above vector is 1. Now $\left|\alpha\right\rangle$ can be written as a sum of two vectors: The contribution from *b*'s with $\deg(b) \le m-r-1$, and from the rest of the *b*'s. The two are orthogonal, since different *b*'s are orthogonal. Hence, the squared norm of $\left|\alpha\right\rangle$, which is 1, (because the operation is unitary and we started with a norm one vector) is the sum of the squared norms of the contribution of $\deg(b) \le m-r-1$, which is also 1, and the norm of the orthogonal vector. Thus, the norm of the sum over *b*'s with $\deg(b) > m-r-1$ must vanish.

Notice that the result of this FT gate is coded by a code of degree *m-r-1* (the input code degree is r). We will use $m = 3r+1$, so that we will be left with a code of degree *2r*. We will show how to reduce the degree of the resulting code from *2r* to *r* later on.

3.1.5 Gate 5: The Toffoli gate $\left|a,b,c\right\rangle \to \left|a,b,c+ab\right\rangle$

One could consider implementing this gate in a straightforward bitwise manner:

$$\left|S_a^r\right\rangle\left|S_b^r\right\rangle\left|S_c^r\right\rangle = \sum \left|f(\alpha_1),...,f(\alpha_m)\right\rangle\left|g(\alpha_1),...,g(\alpha_m)\right\rangle\left|h(\alpha_1),...,h(\alpha_m)\right\rangle \to$$

$$\sum \left|f(\alpha_1),...,f(\alpha_m)\right\rangle\left|g(\alpha_1),...,g(\alpha_m)\right\rangle\left|h(\alpha_1)+f(\alpha_1)g(\alpha_1),...,h(\alpha_m)+f(\alpha_m)g(\alpha_m)\right\rangle = ?$$

$$\left|S_a^{2r}\right\rangle\left|S_b^{2r}\right\rangle\left|S_c^{2r}\right\rangle$$

However, the final equality doesn't hold. The reason is that the multiplicative elements in the final superposition are a code of degree *2r*, while the additive part (the *h* factors) are of degree *r* code.

The solution is to use some extra qubits at the state $\left|S_0^{2r}\right\rangle$ (that is, 0 coded in degree 2r code) to hold the outcome of the bitwise multiplication alone. Hence, we can accept the state: $\left|S_a^r\right\rangle\left|S_b^r\right\rangle\left|S_{a\cdot b}^{2r}\right\rangle\left|S_c^r\right\rangle$. Then we can apply the degree reduction gate (which is explained in the next section) for the degree *2r* code, and add the result to the encoding of *c*. Altogether this implements the Toffoli gate bitwise.

3.1.6 Gate 6: The Fault Tolerant Error Correction gate

This gate is essential for the construction of complete fault tolerant circuits from the bit-wise gates described above. Its functionality is to detect and correct both phase and bit flips. This is done by first detecting and correcting bit-flips using classical error correction techniques, then applying the Fourier Transform and again detecting and correcting phase-flips using classical error correction techniques.

Short summery of classical error correction:

(The following is true for codes over F_2)

Let C be an error-correction code over F_2. The *parity check matrix* H is defined to be the matrix for which the kernel are the words of C. If an error occurred in a word ω, it can be written as: $\omega + e$ where *e* is the error vector. $H \cdot (\omega + e) = H \cdot e = s$ is called the *syndrome*. Hence, given a word in the code vector space, one can calculate the syndrome of the word, and from it deduce the error vector (by multiplying by the inverse of the parity check matrix). From the error vector, one can deduce the original code word by xor-ing the error vector to the word. (the error correction is correct if the distance between the word and the code word that generated it is less then the weight of the minimal code word in C).

Performing classical error correction fault tolerantly:

As explained above, the first step is to compute the syndrome bits. For that we need to multiply the parity check matrix by the input word.

The straightforward approach of multiplying a row *j* H by the word to obtain the *j'th* bit in the syndrome is NOT bit-wise. This is because the *j* multiplications all end up in the same variable, and an error could propagate back into the word and onwards to all other syndrome bits calculation.

Therefore, an interesting technique is used to calculate the *j'th* bit in the syndrome: A state $\left|S\right\rangle$ is built which is a superposition of all states with an even number of 1's. This state is composed of *l* qubits, where *l* is the number of 1's in the *j'th* row of H. Now, we XOR (using a CNOT gate) every qubit in the word (for which there is a 1 in the *j'th* row) to the appropriate qubit in the state S. Then, we can use classical computation to determine the parity of the state S. This parity is clearly the *j'th* bit in the syndrome.

The above procedure is bit-wise. There is one difficulty not addressed – generating the state S fault-tolerantly. Shor [ref2] has shown how to generate this state and proved that it is indeed fault-tolerant. His proof uses measurements and assumes noiseless classical computation. In [ref1] a different proof without measurements is shown.

Once the syndrome is available, we can proceed to calculate the error vector. However, we compute independently each bit in the error vector from a different copy of the syndrome, in order to avoid error propagation from the bits of the error vector to the entire error vector (and finally to the entire corrected word).

As explained previously, after the correction of bit-flips, we apply the FT and perform the same procedure to correct phase flips. Then we can apply the FT again to return to the original (corrected) word.

3.2 Using the Fault Tolerant Gates to build Fault Tolerant Circuits

In the previous section we saw how to implement five bitwise gates, which avoid error-propagation. In this section we shall complete the ingredients needed in order to build general fault tolerant circuits.

Recall that the implementation of gates 4 & 5 (FT and Toffoli) transfer the code into a higher degree. First we show how to reduce the degree of the code with a degree-reduction gate.

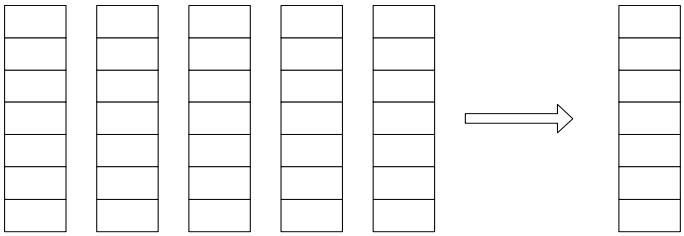
Then we show a gate that implements error correction and is fault tolerant in itself.

3.2.1 Fixing Gates 4 & 5: The Degree reduction gate

The term degree reduction means the following. Let C' and C be two quantum polynomials codes of the same length m, which use polynomials of degree d' and d respectively, such that $d'>d$, where the polynomials are evaluated at the same m points in the field.

A degree reduction procedure takes a word in the first code $\left|\widetilde{S}_a\right\rangle$ to the word $\left|S_a\right\rangle$ in the second code, which encodes the same logical pit. The restriction on our procedure is that it is fault tolerant. This means that we cannot simply decode a and encode it again to get $\left|S_a\right\rangle$, because then even one error cannot be corrected, since the state of the environment will depend on the encoded pit.

The following solution, which generalizes classical degree reduction techniques by Ben-Or, is best illustrated in the following figure:



The idea goes as follows. Let $\sum_{i=1}^{m} c_i f(\alpha_i) = f(0)$ so $\{c_i\}$ are the interpolation coefficients for polynomials of degree less than *m*. We would have wanted to apply interpolation on the coordinates of the encoded state, so as to get *a*, and then to encode *a* by the code *C*. However, this is not fault tolerant.

The trick is to first encode each coordinate in the original state $\left|\widetilde{S}_a\right\rangle$, so each coordinate is *opened* to an encoded word in the code *C*. This is done by first adding *m* encoded states $\left|S_0\right\rangle$, and then, from each coordinate, say the i'th coordinate, apply *m* generalized CNOT gates to all the coordinates in the i'th state $\left|S_0\right\rangle$, so that the extra state now encodes the coordinate.

This is illustrated in the figure by the $m \times m$ matrix, the columns of which are the encoded words, each encoding one coordinate in the original encoded word. We can now apply interpolation pit-wise from the opened coordinates to an extra state $\left|S_0\right\rangle$, by computing along the rows of the matrix. If the matrix elements are symbolically denoted by $a_{i,j}$, we apply the sum $\sum_j c_j a_{i,j}$ over the elements in the i'th row of the matrix, and write it on the i'th coordinate of the extra state $\left|S_0\right\rangle$. Note that this computation is done separately on each row, or pit-wise, on the encoded opened words. It is easy to check that if no errors occurred, this indeed achieves the correct state $\left|S_a\right\rangle$ on the extra state, which was the target of the interpolation.

We would now like to take care of errors that might occur during this procedure, and this will force us to change the above procedure a little, and add error corrections. Note that an error during the opening of each coordinate might cause the entire encoded state to be wrong, since this part was not done fault tolerantly. Note that the interpolation is completely ruined, if even one of the columns of the matrix encodes a wrong pit. Thus, we must apply error corrections before we apply the interpolation. We first apply an error correction on each of the encoded states, so that it is a word in the code. Then, we apply an error correction according to the code *C'*, *encoded* by the code *C*, to the overall word. We will show how to do this in the next sub-section. Given this transformation, it is easy to see that everything we have done is fault tolerant: We first open each coordinate, then apply error corrections, then apply the interpolation, and finally close each encoded state back to one coordinate. This achieves the transformation of $\left|\widetilde{S}_a\right\rangle$ to $\left|S_a\right\rangle$ fault tolerantly, and completes the description of the degree reduction.

3.2.2 Fault Tolerant Encoded Error Correction

We shall sketch out how to apply an encoded error correction on a doubly encoded word. This means the following. The state on which we operate is generated by encoding a state using the code *C*, and then encoding the resulting state again, using the code *C'*. We want to apply on the entire state an encoded error correction procedure. This means that the procedure is exactly the error correction procedure according to the code *C*, but each gate in the procedure is encoded by the code *C'*. We thus want to correct *C'*-encoded pit flips and phase flips, which are equivalent to the following operators on words in *C'*:

$$\varepsilon_{S_b(C')} : \left|S_a(C')\right\rangle \to \left|S_{a+1}(C')\right\rangle$$

$$\varepsilon_{S_f(C')} : \left|S_a(C')\right\rangle \to \omega^a \left|S_a(C')\right\rangle$$

The natural way to correct these errors on words in the code *C* encoded by *C'* is simply to apply the error correction procedure of the code *C*, where each gate is replaced by a fault tolerant procedure (gate) according to the code *C'*. However, one must be careful, in order not to make a circular argument here. The problem is that we want to apply an encoded error correction in spite of the fact that we not yet have the fault tolerant procedures for the entire set of gates used in the error correction, and in particular, we cannot apply encoded Fourier transform, because in this procedure we use degree reduction, which uses encoded error correction. It turns out that this does not pose any difficulty since in this case we can omit the degree reduction everywhere.

The complete technical details of how exactly this is done can be found at [ref1].

4. Back to the Threshold Theorem

In section 3 we saw how to build a set of quantum gates that avoid error propagation (bitwise gates). It turns out that this set of 5 gates (combined with the degree reduction gate) is a universal set of gates. This is proved in [ref1].

Altogether we have a universal set of fault tolerant gates and a fault tolerant EEC procedure.

Now we can prove the Threshold Theorem:

Let *C* be some quantum circuit (composed of the gates in our universal set – since any circuit can be composed out of a universal set of gates).

Let *C'* be the following circuit:

1. Replace each qubit in the circuit with its encoding (*m* qubits, we choose *m=5;r=2* and hence our code can correct one bit flip error or phase flip error).
2. Replace each gate in *C* with the equivalent fault tolerant gate constructed from the universal set of gates described above.
3. After each "replacement circuit" insert an EEC fault-tolerant gate on the output bits.

According to its construction, *C'* is equivalent to *C*. Furthermore, *C'* can correct at most one error for each gate.

Suppose the error rate, which shall be denoted as η, is the probability that an error occurs in some gate in the circuit. Our model of noise assumes that the error rate is independent (in fact, the polynomial error correcting code introduced in this session are effective even with a stronger assumption of exponentially small correlations between errors, but we ignore that fact. For more details see [ref1]). We now bound from above the probability that circuit *C'* will not produce a correct result:

First we calculate the probability that a single gate in *C'* will produce a wrong result:

Define A - The size of the largest implementation of one of the 5 gates + EEC gate.

$$P(C'(x) = error) \le \left(\tfrac{A}{2}\right) \cdot \eta^2$$

This is an upper bound because it is the probability that at least 2 in place in the circuit an error occurred (recall that the gate can recover from one error with the aid of the EEC).

If our scheme shall be effective at all, then we demand:

$$P(C'(x) = error) \leq \left(\tfrac{A}{2}\right) \cdot \eta^2 < \eta$$

(Otherwise, the probability of error would just increase after all the effort of error correction…)

This produces:

$$\eta < \frac{1}{\left(\tfrac{A}{2}\right)}$$

and the best constructions of the gates described above yield:

$$\eta \cong 10^{-6}$$

That is, if the effective error rate η_{eff} holds: $\eta_{eff} < \eta^{1+\varepsilon}$ then we could repeat the scheme above, and construct for *C'* another circuit that will approximate it and be more error-resistant, and so on recursively build a series of circuits: $C' = C_1, C_1, ..., C_k$ for which the error probability per gate would decrease by a factor of $\eta_{eff} < \eta^{1+\varepsilon}$.

Hence, the overall probability per gate would be: $\eta^{(1+\varepsilon)^k}$.

Say the original size of the circuit was S, then by repeating the error correcting scheme a logarithmic number of times (in S) we would achieve an error probability of:

$$\eta^{(1+\varepsilon)^k} < \frac{1}{S}$$

And hence, the probability that an error will occur in the circuit overall would be according to the union bound:

$$P(error) < \frac{1}{S} \cdot S < 1$$

Thus, with positive (constant) probability the resulting circuit computes C's function correctly in the presence of noise up to $\eta_{eff} < \eta^{1+\varepsilon}$.

Because the number of recursions in which the gates were replaces was poly-logarithmic, the size of the resulting circuit is by factor at most poly-logarithmic in the size of the original circuit. Note that the error probability can be reduced by standard amplification techniques.

This completes the proof of the Threshold Theorem.

5. References:

[ref1] – Aharonov & Ben-Or – Fault-Tolerant quantum computation with constant error rate, 1997

[ref2] – Shor P W – Fault tolerant quantum computation, 1996