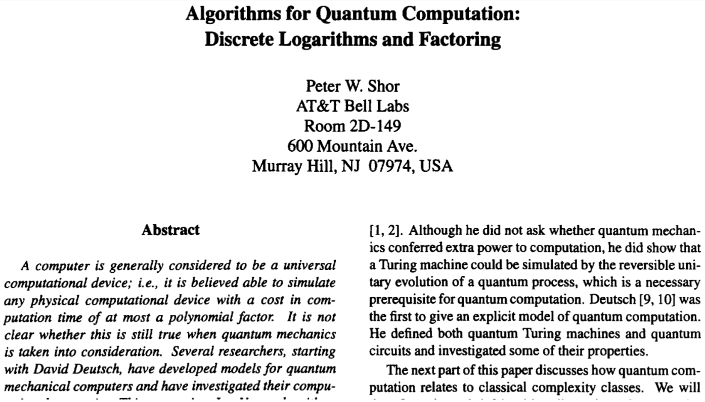
Space Optimization in Shor's Algorithm

André Schrottenloher
Joint work with Clémence Chevignard & Pierre-Alain Fouque

Inria Rennes
Team CAPSULE

The beginning


Algorithms for Quantum Computation: Discrete Logarithms and Factoring

Peter W. Shor
AT&T Bell Labs
Room 2D-149
600 Mountain Ave.
Murray Hill, NJ 07974, USA

Abstract

A computer is generally considered to be a universal computational device; i.e., it is believed able to simulate any physical computational device with a cost in computation time of at most a polynomial factor. It is not clear whether this is still true when quantum mechanics is taken into consideration. Several researchers, starting with David Deutsch, have developed models for quantum mechanical computers and have investigated their compu-

[1, 2]. Although he did not ask whether quantum mechanics conferred extra power to computation, he did show that a Turing machine could be simulated by the reversible unitary evolution of a quantum process, which is a necessary prerequisite for quantum computation. Deutsch [9, 10] was the first to give an explicit model of quantum computation. He defined both quantum Turing machines and quantum circuits and investigated some of their properties.

The next part of this paper discusses how quantum computation relates to classical complexity classes. We will

"This paper gives Las Vegas algorithms for finding discrete logarithms and factoring integers on a quantum computer that take a number of steps which is polynomial in the input size, e.g., the number of digits of the integer to be factored."

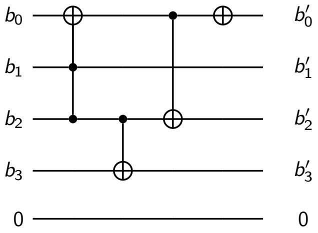
Outline

The Algorithm

Preliminaries: reversible circuits

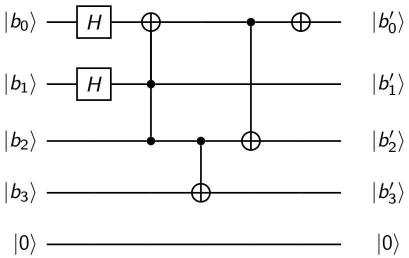
A **classical reversible circuit**:

- Takes as input n Boolean values $(b_0, \ldots, b_{n-1})$
- Operates in place using CNOT, CCNOT (Toffoli) and NOT gates



Any classical algorithm can be turned into a reversible circuit (using **ancilla bits** if necessary).

From reversible to quantum



Quantum circuit are an extension of classical circuits:

- Input is a **linear combination**

$$\sum_{\text{bit-strings } s} \alpha_s |s\rangle \text{ instead of } s$$

- Output as well
- "Classical" gates do not modify the combination
- New "really quantum" gates modify it!

More details

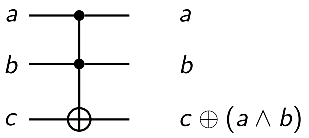
- The linear combinations are over $\mathbb{C}$, and they are **normalized**: $\sum_s |\alpha_s|^2 = 1$
- The quantum gates **preserve the normalization**
- The output of a quantum circuit must be seen as a **probability distribution**:

$$\sum_{\text{bit-strings } s} \alpha_s |s\rangle \rightarrow \text{ obtain } s \text{ with probability } |\alpha_s|^2$$

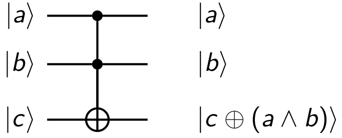
The power of quantum computations comes from operating on a large state (2^n for n qubits), even though we only extract n bits of information.

Example

A Toffoli gate "in superposition":



$\Longrightarrow$



$$|001\rangle \to |001\rangle\,, \qquad |111\rangle \to |110\rangle$$
$$\frac{1}{\sqrt{2}}|001\rangle + \frac{1}{\sqrt{2}}|111\rangle \to \frac{1}{\sqrt{2}}|001\rangle + \frac{1}{\sqrt{2}}|110\rangle$$

The typical “really quantum” operations

The one-qubit **Hadamard gate**:

$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \qquad H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$$

The n-qubit **Hadamard transform**:

$$\forall x, H|x\rangle = \frac{1}{\sqrt{2^n}}\sum_y (-1)^{x\cdot y}|y\rangle$$

The 2^n-Quantum Fourier Transform:

$$\forall x, QFT_{2^n}|x\rangle = \frac{1}{\sqrt{2^n}}\sum_y \omega^{xy}|y\rangle \quad \left(\omega = \exp(2i\pi/2^n)\right)$$

Etc.

QFT_{2^n} **is efficiently implementable**.

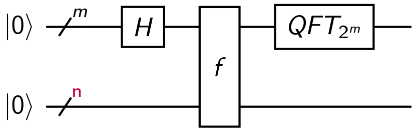
Shor's algorithm for factorization

First of all, **reduce factorization to order-finding**.

- Let N be the number to factor. Select a constant a
- Finding r such that $a^r = 1 \mod N$ allows (w.h.p.) to factor N
- r is solution to a hidden-period problem

Define $f(x) = a^x \mod N$. Then for all x, $f(x + r) = f(x)$.

Order-finding



- Initialize m-bit **input register** for x, n-bit **workspace register** for $f(x)$, additional work qubits
- Build uniform superposition of inputs
- Apply f

$$\frac{1}{\sqrt{2^m}} \sum_x |x\rangle\, |f(x)\rangle$$

- Apply QFT_{2^m} on input register
- Measure

What we obtain

Measuring the workspace gives an integer b and **collapses**:

$$\sum_{x|f(x)=b} |x\rangle$$

The periodicity intervenes here:

$$\sum_{x|f(x)=b} |x\rangle = |x_0\rangle + |x_0 + r\rangle + |x_0 + 2r\rangle + \dots$$

Now what does the QFT_{2^m} do on this state? We have:

$$QFT_{2^m} |x\rangle = \sum_y \omega^{xy} |y\rangle$$

So after QFT:

$$\sum_y \omega^{x_0 y} |y\rangle + \sum_y \omega^{(x_0+r)y} |y\rangle + \sum_y \omega^{(x_0+2r)y} |y\rangle + \dots$$
$$= \sum_y \omega^{x_0 y} \left(1 + \omega^{ry} + \omega^{2ry} + \dots\right) |y\rangle$$

What we obtain (ctd.)

After QFT, the probability to measure y is proportional to:

$$\left|\omega^{x_0 y}\left(1+\omega^{ry}+\omega^{2ry}+\dots\right)\right|^2=\left|1+\omega^{ry}+\omega^{2ry}+\dots\right|^2$$

It gets bigger when ω^{ry} is closer to 1 $\implies$ $\frac{ry}{2^m}$ closer to an integer.

- We measure y such that $\frac{ry}{2^m}$ is "close to an integer".
- With sufficient precision, recover r using a classical post-processing.

Details & Reducing the Space

Components

Operations:

- Algorithm for $H \implies$ trivial
- Algorithm for $f(x) = a^x \mod N$
- Algorithm for $QFT_{2^m} \implies$ not trivial, but easy

The most costly part is also the classical one!

Space:

- Input register
- Workspace register

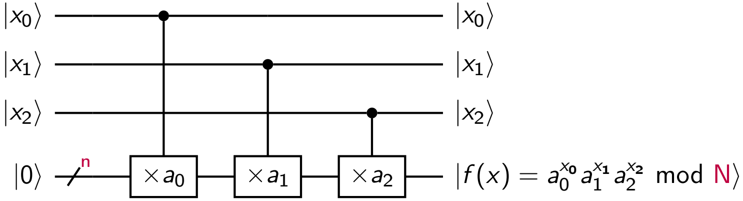
Implementation of f

Reduce **exponentiation** to **controlled multi-product** modulo N:

$$f(x) = a^x = \prod_i \left(a^{2^i}\right)^{x_i} = \prod_i \left(\underbrace{a^{2^i}}_{:=a_i}\right)^{x_i}$$

The constants a_i are precomputed.

- Asymptotic best: $\mathcal{O}(\mathrm{n} \times (\mathrm{n} \log \mathrm{n}))$ operations
- Typical: $\mathcal{O}(\mathrm{n} \times (\mathrm{n}^2))$ operations



Reducing the space

- The input register can be reduced to a **single qubit!**
- The space depends now only on the workspace register

- In practice, implementations of n-bit modular product require at least 2n qubits.

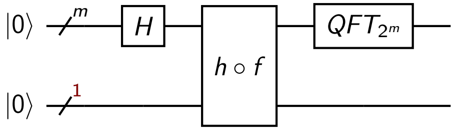
$\Longrightarrow$ space-optimized versions to date need typically $2n + \mathcal{O}(1)$ qubits.

- The best: $1.5n + 2$ by Zalka

Zalka, "Shor's algorithm with fewer (pure) qubits", 2008

Reducing the space even more?

What if we **compress** f into a single-bit output?



- The function $h \circ f$ **is still periodic**.
- If $h : \{0,1\}^n \to \{0,1\}$ is selected from a universal hashing family, then we obtain (almost) the same measurement results!

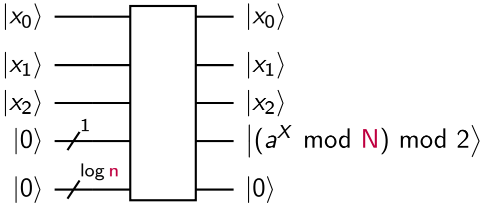
But to compute $h \circ f$, you need first to... compute f ☹

May, Schlieper, "Quantum period-finding is compression robust", ToSC 2022

Compressing the Workspace

Our result

- We can compute **the first bit** of a^x mod N **with** $o(n)$ **space** ($= \mathcal{O}(\log n)$ at best)
- And we can do that in $\mathcal{O}(n^3)$ gates



Our circuit is a **classical arithmetic circuit**, but a bad one: it fails on some random inputs.

The tool: RNS

Start from the **controlled multi-product**:

$$(x_0, \ldots, x_{m-1}) \mapsto a_x := a_0^{x_0} a_1^{x_1} \cdots a_{m-1}^{x_{m-1}}$$

Before reduction modulo N, it has n^2 bits.

We use a Residue Number System: $\mathcal{O}\left(n^2/\log n\right)$ primes p of size $\mathcal{O}(\log n)$ bits, so that $M := \prod p > 2^{n^2}$. By the CRT:

$$\mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_\ell} \simeq \mathbb{Z}_{p_1 \times p_2 \cdots \times p_\ell}$$

$$\Downarrow$$

$$a_x = \left(\sum_p (a_x \bmod p) \times \text{ some cofactor} \right) \bmod M$$

a_x is not a big product anymore, but a **big sum.**

Computing the reduction mod N

Eventually we have:

$$a_x \bmod N = a_x - \lfloor a_x/N \rfloor N$$
$$\implies a_x \bmod N \bmod 2 = a_x \bmod 2 \oplus \lfloor a_x/N \rfloor \bmod 2$$

Step 1. approximate division by N as:

$$\lfloor a_x/N \rfloor \simeq \lfloor a_x \times \lfloor 2^t/N \rfloor / 2^t \rfloor \simeq 2^t\text{-th bit in a big sum}$$

Step 2. approximate the high bits of a sum by truncating the least significant bits. **Here some failure probability appears.**

$\implies$ instead of summing numbers with $\mathcal{O}\left(n^2\right)$ bits, we sum numbers with $\mathcal{O}\left(\log n\right)$ bits

Cost of this approach

Dominating cost: computation of $(a_x \bmod p)$ for $\mathcal{O}\left(n^2/\log n\right)$ primes $p = \mathcal{O}\left(n\right)$.

$$(a_x \bmod p) = (a_0 \bmod p)^{x_0} \cdots (a_{m-1} \bmod p)^{x_{m-1}}$$

- For each prime, the multi-product costs $\mathcal{O}\left(n \log n\right)$ operations

$\Longrightarrow$ $\mathcal{O}\left(n^3\right)$ gates

We're still working on optimizing the computation of $(a_x \bmod p)$ in practice.

Factoring RSA moduli

When $N = PQ$ with $P \simeq Q \simeq 2^{n/2}$, Ekerå and Håstad modify Shor's algorithm:

- Input register reduced to $n/2 + o(n)$ qubits
- More runs might be required
- Special lattice-based post-processing

By combining:

1. May-Schlieper output compression
2. Ekerå-Håstad post-processing
3. Our RNS-based algorithm

we can factor n-bit RSA with $n/2 + o(n)$ **qubits**

Ekerå, Håstad, "Quantum algorithms for computing short discrete logarithms and factoring RSA integers", PQCrypto 2017

Conclusion

Conclusion

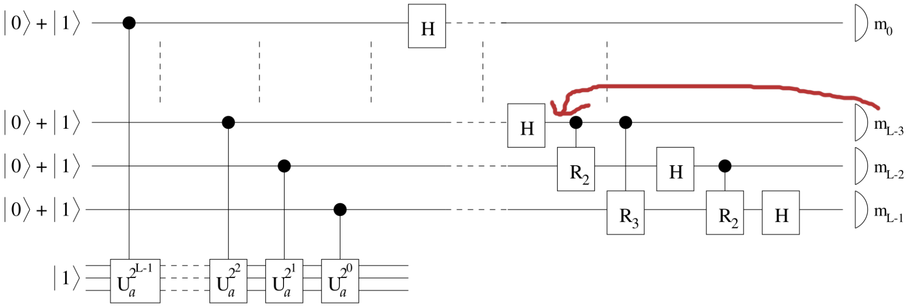
- Since the circuit is classical, we tested it on RSA-2048
- Our estimate for RSA-2048: 1617 qubits incl. 474 ancillas (might be reduced further!) instead of $\simeq 4096$ for "standard" Shor
- Gate counts remain non-negligibly above "standard" Shor (+ more runs required)
- Improvements are possible (needs more engineering of reversible circuits)

Backup slides

Reducing the space: semiclassical Fourier transform

- The input qubits are used only once
- We measure immediately after the QFT

$\Longrightarrow$ use a **single qubit** for input register



Picture from "Efficient factorization with a single pure qubit and log N mixed qubits", Parker, Plenio, Physical Review Letters 2000

Semiclassical Fourier transform (ctd.)

- **Use a single qubit** for input register which is measured and re-initialized m times
- Reduces the space (in theory) to $n + 1 = \log_2 N$ qubits

- In practice, all implementations of n-bit modular product require at least $2n$ qubits.

$\Longrightarrow$ most space-optimized versions to date need $2n + \mathcal{O}(1)$ qubits.

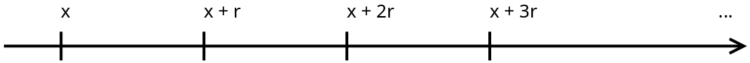
- The best: $1.5n + 2$ by Zalka

Zalka, "Shor's algorithm with fewer (pure) qubits", 2008

Regev's variant: the main idea

Recall that Shor's algorithm reduces factorization to:

one-dimensional hidden period problem "on the line"



- Regev's recent paper reduces it to a $\sqrt{n}$-dimensional hidden lattice problem instead.
- Only $\mathcal{O}\left(\sqrt{n}\right)$ instead of n multiplications modulo N
- However, needs $\mathcal{O}\left(\sqrt{n}\right)$ repetitions

Regev, "An efficient quantum factoring algorithm." arXiv preprint arXiv:2308.06572 (2023).

Regev's variant: the main idea

Let $b_1, \dots, b_d$ be **many small integers** and $a_i = b_i^2$. The goal is to find (a basis of) the d-dimensional lattice:

$$\mathcal{L} = \{(z_1, \dots, z_d), \prod_i a_i^{z_i} = 1 \mod N\}$$

which we expect to be of dimension d. Intuitively by choosing $d = \sqrt{\mathsf{n}}$ we can expect short vectors of norm $2^{\mathcal{O}(\sqrt{\mathsf{n}})}$.

The starting space in Shor's algorithm needs to be $\simeq$ as large as the period size.

- Before we needed a line of length $\simeq 2^{\mathsf{n}}$
- Now we need a cube of side $\simeq 2^{\sqrt{\mathsf{n}}}$

A small remark

$$d\text{-dimensional infinite period-finding} =$$

$$\underbrace{\text{1-dimensional infinite period-finding}}_{\text{Shor}} + \underbrace{d\text{-dimensional finite period-finding}}_{\text{Also Shor, without errors!}}$$

(That's how we do the DL case actually!)

However this basic reduction requires an input cube size which is **too large**. Regev **really needs** to solve the *d*-dimensional infinite problem directly.

The algorithm

1. Construct a superposition over the d-dimensional cube of side $D \simeq 2^{\sqrt{n}}$ (**not uniform**, but a discrete Gaussian)
2. Compute $\prod_i a_i^{z_i} \mod N$
3. Apply $(QFT_D)^{\otimes d}$ and measure

The technical part: like 1-dimensional infinite case, there are **approximation errors**. We must recover the **hidden lattice** from **noisy samples of its dual**.

The algorithm: implementation

Most time-consuming part:

$$(z_1, \ldots, z_d) \mapsto \prod_i a_i^{z_i} \mod N$$

Where $d \simeq \sqrt{n}$ and $z_i \simeq 2^{\sqrt{n}} \implies$ still n multiplications modulo N!

However:

$$\prod_i a_i^{z_i} = \prod_{i,j} a_i^{z_i^{\mathbf{0}} + 2z_i^{\mathbf{1}} + 2^{\mathbf{2}} z_i^{\mathbf{2}} + \ldots} = \left(\underbrace{\prod_i a_i^{z_i^{\mathbf{0}}}}_{\text{small}} \right) \left(\prod_{i,j} a_i^{z_i^{\mathbf{1}} + 2z_i^{\mathbf{2}} + \ldots} \right)^2 .$$

$\implies$ many multiplications of small integers + $\sqrt{n}$ large modular multiplications

Summary

- A single circuit gains a $\sqrt{n}$ factor in gates & depth compared to Shor
- However, we need to run $\sqrt{n}$ circuits to find the whole lattice (instead of 1) $\implies$ total gate count unchanged

Regev's original paper uses $\mathcal{O}\left(n^{3/2}\right)$ qubits due to a reversibility issue, but a more recent work achieves $\mathcal{O}\left(n\right)$ qubits also.

Ragavan, Vaikuntanathan, (2023). Optimizing Space in Regev's Factoring Algorithm. arXiv preprint arXiv:2310.00899.