Reducing the Number of Qubits in Quantum Factoring

24/02/2025

Clémence Chevignard & Pierre-Alain Fouque & André Schrottenloher

Team Capsule, Inria

Shor's algorithm

Goal: factor an integer $N = p \times q$.

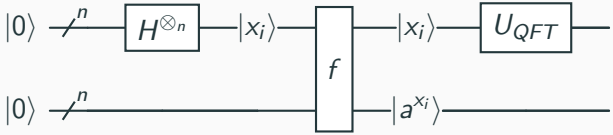
Idea:

1. Take $a \in \mathbb{Z}/N\mathbb{Z}$ randomly.
2. Find the order x of a in $\mathbb{Z}/N\mathbb{Z}$. ..... this is the quantum step
3. If x even and $a^{x/2} \neq -1 \mod N$, then $p, q = \gcd(a^{x/2} \pm 1, N)$.
   Otherwise, repeat.

Shor's algorithm – factor $N = p \times q$ – Find the order x of a in $\mathbb{Z}/N\mathbb{Z}$

2.(a) Set up $|s\rangle$ as a superposition of all $|x_i\rangle$ $|a^{x_i}\rangle$ states, $x_i \in \{1, O(N)\}$, all with the same proba.

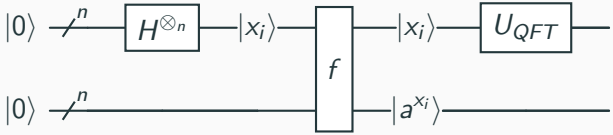
"input" register "output" register



Shor's algorithm – factor $N = p \times q$ – Find the order x of a in $\mathbb{Z}/N\mathbb{Z}$

2.(a) Set up $|s\rangle$ as a superposition of all $|x_i\rangle$ $|a^{x_i}\rangle$ states, $x_i \in \{1, O(N)\}$, all with the same proba.

"input" register "output" register



2.(b) Apply a Fourier transform on the input registers

$$U_{QFT}\,|x_i\rangle := \frac{1}{\sqrt{O(N)}} \sum_{y_j=0}^{O(N)-1} \omega^{x_i y_j}\,|y_j\rangle\,, \qquad \omega = e^{2i\pi/N}$$

2.(c) Do a measure and deduce stuff.

What do we aim at

Original Shor algorithm $\rightarrow$ space complexity of $\mathcal{O}(n)$ qubits, $n := \lceil \log_2 N \rceil$.

The best memory complexity currently $\rightarrow 1.5n + 2$ qubits, by Zalka.[1]

[1] See Zalka, "Shor's algorithm with fewer (pure) qubits", arxiv, 2008

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

What do we aim at

Original Shor algorithm $\rightarrow$ space complexity of $\mathcal{O}(n)$ qubits, $n := \lceil \log_2 N \rceil$.

The best memory complexity currently $\rightarrow 1.5n + 2$ qubits, by Zalka.[1]

We take it down to $\frac{n}{2} + o(n)$.

Ekerå-Håstad size reduction method for RSA integers[2] $\rightarrow$ input register size down to $\frac{n}{2} + o(n)$ qubits.

$\rightarrow$ What about the **output register size**?

[1] See Zalka, "Shor's algorithm with fewer (pure) qubits", arxiv, 2008

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

The hash function idea – May and Schlieper[3]

Let H be a hash function.

$\rightarrow$ Create a superposition of all $|x\rangle\,|H(a^x)\rangle$ instead of $|x\rangle\,|a^x\rangle$.

[3] May and Schlieper, "Quantum Period Finding is Compression Robust",IACR Trans. Symmetric Cryptology, 2022

Which hash function to choose

Let's take $H : z \mapsto z \mod 2^r \rightarrow$ output register of r qubits!

($r = 22$).

We can compute $a^x \mod N \mod 2^r$ naively?

$$a^x \to [a^x]_N \to [[a^x]_N]_{2^r}$$

Notation: $[a]_N := a \mod N$

$$x = x_0 + 2x_1 + \ldots + 2^{n-1}x_{n-1} \quad \to \quad a^x = \prod_{i=0}^{n-1} a^{2^i x_i} = \prod_{i=0}^{n-1} \left(a^{2^i}\right)^{x_i}$$

$$\underbrace{a^{(i)}}_{\substack{\in \mathbb{N} \\ \text{Bit-size of } n}} := a^{2^i} \mod N \quad \to \quad a^x \mod N = \underbrace{\prod_{i=0}^{n-1} \left(a^{(i)}\right)^{x_i}}_{\substack{\in \mathbb{Z} \\ \text{Bit-size of } n^2}} \mod N$$

$$a^x \to [a^x]_N \to [[a^x]_N]_{2^r}$$

Notation: $[a]_N := a \mod N$

$$x = x_0 + 2x_1 + \ldots + 2^{n-1}x_{n-1} \quad \to \quad a^x = \prod_{i=0}^{n-1} a^{2^i x_i} = \prod_{i=0}^{n-1} \left(a^{2^i}\right)^{x_i}$$

$$\underbrace{a^{(i)}}_{\substack{\in \mathbb{N} \\ \text{Bit-size of } n}} := a^{2^i} \mod N \quad \to \quad a^x \mod N = \underbrace{\prod_{i=0}^{n-1} \left(a^{(i)}\right)^{x_i}}_{\substack{\in \mathbb{Z} \\ \text{Bit-size of } n^2}} \mod N$$

Notation: $Y \leftarrow \prod_{i=0}^{n-1} \left(a^{(i)}\right)^{x_i}$ of bit-size n^2.

$$Y \to [a^x]_N = [Y]_N \to [[a^x]_N]_{2^r}$$

Notation: $[a]_N := a \mod N$, $Y \leftarrow \prod_{i=0}^{n-1} \left(a^{(i)}\right)^{x_i}$, $x = x_0 + 2x_1 + \ldots + 2^{n-1}x_{n-1}$.

$$Y \to [a^x]_N = [Y]_N \to [[a^x]_N]_{2^r}$$

Notation: $[a]_N := a \mod N$, $Y \leftarrow \prod_{i=0}^{n-1} \left(a^{(i)}\right)^{x_i}$, $x = x_0 + 2x_1 + \ldots + 2^{n-1}x_{n-1}$.

Strategy:

- Compute Y by
  - computing $Y \mod p$ **for small primes** p
  - and then $Y = Y \mod \prod p$

  $\to$ Residue Number System + Chinese Remainder Theorem.
- Compute $Y \mod N \to$ Barrett's reduction trick.
- Apply the $\mod 2^r$ operation.

$$Y \to [a^x]_N = [Y]_N \to [[a^x]_N]_{2^r}$$

Notation: $[a]_N := a \mod N$, $Y \leftarrow \prod_{i=0}^{n-1} \left(a^{(i)}\right)^{x_i}$, $x = x_0 + 2x_1 + \ldots + 2^{n-1}x_{n-1}$.

Choose $M = p_1 \ldots p_\ell$ such that $M > 2^{n^2}$.

$$\mathbb{Z}/M\mathbb{Z} \simeq \mathbb{Z}/p_1\mathbb{Z} \times \ldots \mathbb{Z}/p_\ell\mathbb{Z}$$

$$\Downarrow$$

$$Y = [Y]_M = \sum_{p_i} [Y]_{p_i} \times (\text{ some cofactor }) \mod M$$

$\ell \simeq \dfrac{n^2}{\log n}$, $p_i \simeq 2^{\log n}$

$$Y \to [a^x]_N = [Y]_N \to [[a^x]_N]_{2^r}$$

Notation: $[a]_N := a \mod N$, $Y \leftarrow \prod_{i=0}^{n-1} \left(a^{(i)}\right)^{x_i}$, $x = x_0 + 2x_1 + \ldots + 2^{n-1}x_{n-1}$.

Choose $M = p_1 \ldots p_\ell$ such that $M > 2^{n^2}$.

$$\mathbb{Z}/M\mathbb{Z} \simeq \mathbb{Z}/p_1\mathbb{Z} \times \ldots \mathbb{Z}/p_\ell\mathbb{Z}$$

$$\ell \simeq \frac{n^2}{\log n},\ p_i \simeq 2^{\log n}$$

$$\Downarrow$$

$$Y = [Y]_M = \sum_{p_i} [Y]_{p_i} \times \underbrace{(\text{ some cofactor })}_{\text{can be precomputed}} \mod M$$

$$Z \to Y = [Z]_M \to [a^x]_N = [Y]_N \to [[a^x]_N]_{2^r}$$

Notation: $Y = \underbrace{\sum_p [Y]_p \times (\text{ some cofactor })}_{Z} \mod M.$

$$Z \to Y = [Z]_M \to [a^x]_N = [Y]_N \to [[a^x]_N]_{2^r}$$

Notation: $Y = \underbrace{\sum_p [Y]_p \times (\text{ some cofactor })}_{Z} \mod M.$

Barrett's reduction trick for modulo's computations:
Approximate $\lfloor Z/M \rfloor \simeq \lfloor Z \times \lfloor 2^u/M \rfloor / 2^u \rfloor$

$$\to Z \mod M = Z - \lfloor Z/M \rfloor M \simeq Z - \lfloor Z \times \lfloor 2^u/M \rfloor / 2^u \rfloor M.$$

$$\text{Note} \begin{cases} \boldsymbol{q_M} = \lfloor Z/M \rfloor \\ \boldsymbol{Q_N} = \lfloor Y/N \rfloor \end{cases} \quad \rightarrow [a^x]_N = Z - q_M M - Q_N N$$

$$\text{Note} \begin{cases} \boldsymbol{q_M} = \lfloor Z/M \rfloor \\ \boldsymbol{Q_N} = \lfloor Y/N \rfloor \end{cases} \quad \rightarrow [a^x]_N = Z - q_M M - Q_N N$$

For all p dividing M, compute $[Y]_p$.

- $Z = \sum_p [Y]_p \times (\text{ some cofactor })$.
- $q_M = \left\lfloor \frac{1}{2^u} \left(\sum_p [Y]_p \times \text{cofac} \lfloor 2^u/p \rfloor \right) \right\rfloor + 1$
- $Q_N = \left\lfloor \frac{1}{2^{u'}} \left(\sum_p \sum_i ([Y]_p)_{\text{bit } i} \times \text{cofac} + \sum_i (q_M)_{\text{bit } i} \times \text{cofac'} \right) \right\rfloor$

Performances

Space needed: $\frac{n}{2} + o(n)$ instead of $1.5 + o(n)$ for previous best implementations.

Concretely, for a 2048-bit integer $\rightarrow$ 1730 qubits.

Gate count: $\mathcal{O}(n^3) \rightarrow \mathcal{O}(n^2 \log n)$ for the original Shor.

Depth: $\mathcal{O}(n^2 poly(\log(n)))$.

Summary

Key tools for this optimisation:

- Residue Number System
- Barrett's reduction
- Successive computation of the $[Y]_p$

Summary

Key tools for this optimisation:

- Residue Number System
- Barrett's reduction
- Successive computation of the $[Y]_p$

Open questions:

- Can we optimise Shor for elliptic curve problems? For factoring any kind of integers?
- Can we reduce the gate counte overhead? Yes.[a]

[a] Craig Gidney, "How to factor 2048 bit RSA integers with less than a million noisy qubits", 2025

Summary

Key tools for this optimisation:

- Residue Number System
- Barrett's reduction
- Successive computation of the $[Y]_p$

Open questions:

- Can we optimise Shor for elliptic curve problems? For factoring any kind of integers?
- Can we reduce the gate counte overhead? Yes.[a]

[a] Craig Gidney, "How to factor 2048 bit RSA integers with less than a million noisy qubits", 2025

Question time!