57º SBPO
SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
Gramado • RS
DE 5 A 9 DE OUTUBRO DE 2025

Boosting Shor's Factoring Success with the Jacobi Symbol in Single-Run Order Finding against large integer numbers

Vitor Pio Silva
vitor@vitorpio.com

Fabio Gomes dos Santos
fabiogsrj.2007@gmail.com

Luis Antonio Brasil Kowada
luis@ic.uff.br
Universidade Federal Fluminense
Av. Gal. Milton Tavares de Souza, s/n, Sao Domingos, Niteroi/RJ, 24210-346

ABSTRACT

Several works have already presented simulations of Shor's algorithm. However, the vast majority encounter limitations when applied to large integers. This is primarily because simulating the algorithm significantly demands classical hardware's memory and processing capabilities. Specifically, a substantial amount of memory is required to store the numerous states dictated by the algorithm. Shor's formulation suggests that the number of states is a power of two between N^2 and $2N^2$. For instance, factoring a 2048-bit number would require 2^{4096} states. This is clearly beyond the capacity of contemporary classical computers. This work presents a classical simulation reproducing Shor's integer-factoring algorithm capable of handling these capacity issues and generating noise-free quantum measurement outcomes. The built simulation algorithm retains the core quantum components from Shor's original work while optimizing classical post-measurement processing. This simulation enabled us to involve large integers, validate certain assumptions, and extract relevant information.

KEYWORDS. Number Theory. Cryptography. Quantum computing.

DOI: 10.59254/sbpo-2025-212078
https://proceedings.science/p/212078?lang=pt-br

1. Introduction

RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem whose security relies on the hardness of integer factorization. The robustness of cryptographic systems such as RSA rests on the difficulty of factoring large integers whose prime factorization is unknown [Rivest et al., 1978]. Shor's algorithm [Shor, 1999] demonstrates that an ideal universal quantum computer solves this problem in polynomial time by reducing factoring to finding the multiplicative order of an element of $\mathbb{Z}_N^*$. Since then, evaluating this algorithm's practical performance and variations has become fundamental to estimating the impact of large quantum processors on data security.

Despite the existence of simulators capable of executing Shor's quantum circuit gate by gate, such fine-grained approaches are constrained to a few tens of qubits due to the exponential cost of memory and time. An alternative line of research models the order-finding subroutine (Quantum Order-Finding Algorithm, QOFA) as a black box that directly returns the order value. This makes testing N modules of hundreds or thousands of bits possible. However, it obscures the statistical behavior inherent in quantum measurement. Specifically, the distribution of the peaks generated by the Quantum Fourier Transform is lost, influencing the classical post-processing stage. This paper proposes a simulation model to address this knowledge gap while keeping possible run tests for large numbers, aiming to validate some of the ideas proposed by this work.

2. Related Work

Simulations offer significant advantages for understanding and studying various algorithms, including quantum algorithms. Software simulations can be designed to address a system's broader aspects or focus more narrowly on specific components of the problem under investigation. The choice of simulation granularity should align with the specific information sought. For example, a more fine-grained simulation is preferred when analyzing error propagation within a routine. Conversely, a less granular solution could be employed if the goal is to analyze a post-processing algorithm based on pre-defined results.

In the case of Shor's algorithm [Shor, 1999] for factorization, we can consider the routine of order estimation (QOFA - Quantum Order Finding Algorithm) as a black box, as was done in [Bastos and Kowada, 2021] instead of simulate smaller steps of the algorithm. Considering the quantum routine as a black box that returns the order of a number, we can obtain simulation results of arbitrarily large numbers as those used in RSA, for instance, 1024 bits or more. In this way, we can get an estimate of how many successful QOFA executions are necessary, but not how many executions would be required, since some QOFA executions may not bring relevant information to the problem.

Shor's algorithm simulations, which include the QOFA simulation, have very few bits. See, for example, [Wang et al., 2017; Tankasala and Ilatikhameneh, 2019] with N being a number with a few bits. There is also [Willsch et al., 2023] where they could execute QOFA with numbers about 70/80 bits, but they had to provide large hardware to accomplish their results. The algorithm simulation with small numbers may not be realistic in the sense of being a degenerate situation. For example, if the number is tiny, there is a high chance of finding a factor at random. In addition to the level of granularity, another essential property is the set of information that simulations use. Generally, the strictest simulations, with finer granularity, do not use information other than the problem's input. However, there are situations where other information can help with the simulation. For example, it is only possible to simulate factorization for arbitrary vast numbers, knowing their factors in advance, as in simulations in [Bastos and Kowada, 2021]. Knowing and using this information does not invalidate the simulation if the results are similar to those in the real situation. Another interesting related work can be found in [Ekerå, 2021], where the authors built upon the work of [Miller, 1975]. They also propose a solution that improves the order-finding execution, aiming to raise the probability of running this routine only once, albeit through a different method.

DOI: 10.59254/sbpo-2025-212078
https://proceedings.science/p/212078?lang=pt-br

3. Our contributions

First, we introduce a probabilistic model that reproduces the distribution of the r peaks generated by the Quantum Fourier Transform: for each i the simulator sets $j = \lfloor iQ/r \rfloor$ and randomly picks the right-hand neighbor $\lceil iQ/r \rceil$ with probability $1 - (iQ \bmod r)/r$.

This mechanism preserves the statistics of a QFT and obliges the classical post-processing stage to examine the two candidates $j-1, j+1$. Second, we generalize the arithmetic setting from the usual safe-prime case $p = 2p' + 1$, $q = 2q' + 1$ to primes of the broader form $p = 2^c t + 1$,$q = 2^c u + 1$ with $c \geq 1$ and t, u odd. We prove and confirm experimentally with the simulation that restricting the choice of the element x to those satisfying $\left(\frac{x}{N}\right) = -1$, where $\left(\frac{x}{N}\right)$ means Jacobi symbol for x and n, guarantees a non-trivial factor in exactly one execution of the order-finding subroutine for this wider family of moduli.

4. Background

This section gathers the concepts needed for the simulation proposed later in this work. First, we recall the order-finding problem that underlies Shor's algorithm, then we outline the measurement statistics of the Quantum Fourier Transform (QFT) that our simulator preserves, and finally, we review the Jacobi symbol and explain why filtering the base x with $\left(\frac{x}{N}\right) = -1$ allows one to factor N after a single call to the quantum routine even beyond the usual safe-prime setting.

Starting the order finding and factorization stage. Let $N = p\,q$ be the product of two distinct odd primes. For $x \in \mathbb{Z}_N^*$, the multiplicative order $r = \mathrm{ord}(x, N)$ is the least positive integer such that $x^r \equiv 1 \pmod N$. If r is even and $x^{r/2} \not\equiv -1 \pmod N$, then the greatest common divisor (gcd) $\gcd\big(x^{r/2} - 1,\, N\big)$ yields a non-trivial divisor of N. Shor's circuit extracts r by phase estimation applied to the unitary $U : |y\rangle \mapsto |x^y \bmod N\rangle$.

Let $n = \lceil \log_2 N \rceil$ and set $Q = 2^{2n}$. After the inverse QFT, measuring that register returns an integer j that encodes a rational approximation i/r [Shor, 1999].

Immediately before measurements, the probability mass concentrates on the two integers closest to iQ/r. Consequently, the classical post-processing inspects $(j-1)/Q$, j/Q, and $(j+1)/Q$ to recover information about i/r, in particular its denominator r.

Our simulator reproduces this by drawing i uniformly, setting $j = \big\lfloor iQ/r \big\rfloor$, and then sampling $t \sim \mathrm{Unif}[0, 1)$; it increments j to $\big\lceil iQ/r \big\rceil$ whenever $t > (i \cdot Q \bmod r)/r$.

For an odd modulus $N = pq$, the *Jacobi symbol* is defined by

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right)\left(\frac{x}{q}\right)$$

and takes values in $\{-1, 0, 1\}$. When p and q are safe primes ($p = 2p' + 1,\ q = 2q' + 1$), choosing x with

$$\left(\frac{x}{N}\right) = -1$$

implies

$$1 < \gcd\big(x^{r/2} - 1,\, N\big) < N,$$

so a factor of N is found after a single call to the order-finding subroutine. The present paper extends these results to primes of a more general form

$$p = 2^c t + 1, \quad q = 2^c u + 1, \qquad c \geq 1, \;\; t, u \text{ odd}.$$

We prove that the Jacobi symbol filter still guarantees factorization in one quantum run for this broader family of moduli and confirm the claim experimentally for 128, 512, and 1024-bit composites.

DOI: 10.59254/sbpo-2025-212078
https://proceedings.science/p/212078?lang=pt-br

These theoretical elements underpin the two central contributions of this work: (i) a simulator that preserves the genuine peak distribution with the classical routine to test $j-1, j, j+1$ and (ii) a Jacobi-based filter that achieves factorization for arbitrary primes of the form $p = 2^c t+1,\ q = 2^c u + 1$, going well beyond the safe-prime setting explored in earlier studies.

5. Simulation Algorithm

The simulator begins exactly where a physical implementation of Shor's order-finding routine would hand information to the classical side at the integer index j obtained by measuring the first register after the inverse Quantum Fourier Transform (QFT). The quantum circuit itself is not emulated; a sampler produces a value j drawn from the same distribution that an ideal QFT measurement would exhibit. From then on, the algorithm follows the classical post-processing specified by Shor, augmented by a Jacobi symbol filter and a new exponent-inflation heuristic that guarantees factorization in a single synthetic run.

5.1. Instance and base generation

Each experiment starts by constructing a composite $N = pq$ whose prime factors satisfy $p = 2^c t + 1,\ q = 2^c u + 1,\ c \geq 1,\ t, u$ odd.

Fixing the exponent c in advance leaves the choice of t and u uniform within the required bit-length. The familiar safe-prime case corresponds to $c = 1$. Knowledge of the complete factorization of $\varphi(N) = (p-1)(q-1)$ is thus available at generation time and will be exploited to compute multiplicative orders without recursion to a quantum oracle. The provisional base x is then uniformly sampled from $\{2, \ldots, N-2\}$ until the Jacobi symbol satisfies $\left(\frac{x}{N}\right) = -1$. The acceptance probability is one-half because the Jacobi symbol is equidistributed on non-zero residues.

Theorem 1 proves that this simple filter forces the classical test $\gcd\left(x^{r/2}-1,\ N\right)$ to reveal a factor after a single order-finding attempt, irrespective of the chosen exponent c.

5.2. Synthetic peak sampler

Let $r = \mathrm{ord}(x, N)$ denote the multiplicative order of x modulo N. Because the prime decompositions of $p-1$ and $q-1$ are known, the exact values

$$r_p = \mathrm{ord}(x, p) \quad \text{and} \quad r_q = \mathrm{ord}(x, q)$$

can be obtained by repeated division tests, and the global order follows as the least common multiple (lcm)

$$r = \mathrm{lcm}(r_p,\, r_q).$$

Let $n = \lceil \log_2 N \rceil$ and set $Q = 2^{2n}$.

The sampler then reproduces the peak selection implemented in the listing:

1. Draw $\mathtt{i}$ uniformly from $\{0, \ldots, r-1\}$ and compute $j = \mathtt{i} \cdot Q \,//\, r$.

2. Sample a real t uniformly from $[0, 1)$; if $t > (\mathtt{i} \cdot Q \bmod r)/r$, set $j \leftarrow j + 1$.

The integer j is the one datum handed to the classical post-processing stage. Because this measurement can differ from the exact position iQ/r by at most one unit, the routine later examines $(j-1)/Q,\ j/Q,\ (j+1)/Q$ to retrieve a reduced denominator r' that divides the true order r.

DOI: 10.59254/sbpo-2025-212078
https://proceedings.science/p/212078?lang=pt-br

5.3. Classical post-processing

Once the sampler has delivered an integer peak j, the goal is to extract from j/Q a denominator r' that divides the true order r and, ultimately, factorize the modulus N.

Step 1: Continued fractions

Given j, the routine inspects the three neighboring rationals

$$\frac{j-1}{Q}, \qquad \frac{j}{Q}, \qquad \frac{j+1}{Q}.$$

For each $k \in \{-1, 0, 1\}$ we have a function that produces the continued-fraction expansion and then collects from all convergents the denominators v that satisfy $1 < v < N$. The result is a list $\mathcal{L}$ (variable `L`) of candidate moduli for the exponent tests that follow.

Step 2: Detection of the divisor of r.

For each $v \in \mathcal{L}$ we have a procedure that computes

$$d \;=\; \gcd\big(a^{\,v} - 1,\, N\big),$$

with the same base a used in the modular multiplications. If $1 < d < N$, a non-trivial factor of N has been found and the run terminates successfully (counter r' is set). Otherwise, the algorithm proceeds to inflate v into a multiple of the true order.

Step 3: Exponent inflation. For each candidate denominator $v \in \mathcal{L}$ the routine attempts to construct, without knowing r, an exponent that is a multiple of the true order. It fixes the small-prime budget

$$\texttt{peq_primos} = \{2{:}4,\ 3{:}3,\ 5{:}3,\ 7{:}2,\ 11{:}1,\ 13{:}1,\ 17{:}1,\ 19{:}1\},$$

computes the product $F = \prod_p p^{\texttt{peq_primos}[p]}$, and forms

$$\texttt{r'} \;=\; v \cdot F.$$

The code then tests the congruence $a^{\texttt{r'}} \equiv 1 \pmod N$. If the test fails, it returns the sentinel value $-v$, signaling that the inflation with F did not reach a multiple of the order.

If the test succeeds, the function performs an in-place thinning of r' using the same small primes: for each p in $\{2, 3, 5, 7, 11, 13, 17, 19\}$ it repeatedly divides out a single factor p while the congruence $a^{\texttt{r'}} \equiv 1 \pmod N$ remains true; when a trial division breaks the congruence, it restores one factor p and proceeds to the next prime. The value returned is thus a partially reduced multiple of the order, still denoted r', which is then passed to the final reduction in Step 4.

Step 4: greedy reduction.

The value r' returned by a function that tests multiples of r, is passed to the helper routine called `reduzR` for a final, local pruning with respect to small primes. The function first checks that $a^{\texttt{r'}} \equiv 1 \pmod N$; otherwise it returns -1, signaling that the input was not (any longer) multiple of the order.

It then scans once the fixed list of primes

$$\{2, 3, 5, 7, 11, 13, 17, 19\}.$$

For each prime p in this list, while `r'` is divisible by p the routine tentatively cancels a single factor ($\texttt{r'} \leftarrow \texttt{r'}/p$) and recomputes $a^{\texttt{r'}} \bmod N$. If the congruence $a^{\texttt{r'}} \equiv 1 \pmod N$ still holds, the division is kept and the loop tries to remove another p; if the congruence breaks, one factor p is restored ($\texttt{r'} \leftarrow p \cdot \texttt{r'}$) and the algorithm proceeds to the next prime.

At termination, the returned value-denoted here by r^* (and stored back in the variable `r'`)-still satisfies $a^{\,r^*} \equiv 1 \pmod N$, and no further cancellation by any $p \in \{2, 3, 5, 7, 11, 13, 17, 19\}$

DOI: 10.59254/sbpo-2025-212078
https://proceedings.science/p/212078?lang=pt-br

is possible without destroying this congruence. In other words, r^* is a "lean" multiple (with respect to those small primes) of the underlying order, ready for the GCD tests of Step 5.

Step 5: greatest-common-divisor tests. The final stage applies GCD checks to the denominators produced by the continued-fraction phase. Let L denote the list of denominators returned by the continued-fraction analysis. For each $v \in L$ we compute $d = \gcd(a^v - 1,\, N)$. If $1 < d < N$, a non-trivial factor of N is immediately obtained, and the run terminates.

If the direct attempt fails, we switch to the adjusted-exponent list L_{lin}. For each $v \in L$ we form $r' = v \cdot F$ with $F = 2^4 3^3 5^3 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$ and test whether $a^{r'} \equiv 1 \pmod N$. When this holds, we greedily strip powers of the small primes in $S = \{2, 3, 5, 7, 11, 13, 17, 19\}$ from r' while preserving the congruence, obtaining a "lean" multiple $r_* > 0$. From r_* we then try $\gcd\big(a^{r_*/q} - 1,\, N\big)$ for every $q \in S$ with $q \mid r_*$. If the inflation check already fails (i.e., $a^{vF} \not\equiv 1 \pmod N$), we simply fall back to the direct test with $\gcd\big(a^v - 1,\, N\big)$.

The intuition is that, after stripping small prime factors, the exponent often annihilates the base modulo one of p or q but not the other; in that case $a^{r_*/q} \equiv 1 \pmod p$ while $a^{r_*/q} \not\equiv 1 \pmod q$, so the resulting greatest common divisor equals the corresponding prime factor. In practice, one of these two passes (the direct denominators or the inflated-and-reduced exponents) almost always succeeds. All operations here are integer GCDs and modular exponentiations with smooth exponents, keeping the post-processing lightweight.

Step 6: second base.

If neither the direct continued-fraction test nor the inflated-exponent test succeeds for the initial base a, the implementation optionally retries the GCD phase with a small number of fresh bases. Concretely, the routine draws $x_0, x_1 \in \{2, \ldots, N-2\}$ uniformly at random (the number of retries is controlled by the parameter `bases`, which is 2 by default), records their Jacobi symbols $\left(\frac{x_i}{N}\right)$ for reporting, and reuses unchanged the exponent candidates built from the measured peak-namely the list of denominators L and the corresponding list L' of inflated/reduced exponents produced in Steps 1-5. For each fresh base x_i drawn uniformly from $\{2, \ldots, N-2\}$, we reuse the same exponent candidates built from the measured peak (namely the denominators L and the adjusted list L') and apply the same GCD tests as in Step 5 with the new base. The procedure halts as soon as a nontrivial divisor of N is obtained. For reporting, we also record the Jacobi symbols $\left(\frac{x_i}{N}\right)$, but no Jacobi constraint is enforced at this stage.

6. Jacobi Filter: Proof of Single-Run Factorization

The Jacobi Symbol is a generalization of the Legendre symbol. Both belong to the field of number theory. The Jacobi Symbol is also known as the quadratic residue. The concept behind quadratic residues is that given two co-prime integers x and p, x is a quadratic residue modulo p if there is some y value that satisfies $y^2 \equiv x \bmod p$ (Jacobi Symbol = 1). Au contraire, if no y value satisfies this congruence, we say x is a quadratic non-residue modulo p (Jacobi Symbol = -1).

Theorem 1. *Let $N = p \cdot q$, where p and q are prime numbers. Further, let $p = 2^c \cdot u + 1$ and $q = 2^c \cdot v + 1$, where u and v are integers and c is a constant. For all $x \in \mathbb{Z}_N^*$, where $\gcd(x, N) = 1$ with $x > 1$, we have:*

$$\textit{If Jacobi Symbol}(x,\, N) = -1 \textit{ then } 1 < gcd(x^{r/2} - 1, N) < N$$

Proof. The Jacobi Symbol (JS) of x with $N = p \cdot q$ can be represented as a product of two Legendre symbols (LS) of x with p and x with q. In this way, given any integer x and any positive odd integer N, we can substitute $JS(x, N)$ for $LS(x, p) \cdot LS(x, q)$, where $N = p \cdot q$ with p and q being odd primes. By the definition of Legendre Symbol, we know that the result of $LS(a, b) \in \{-1, 0, 1\}$.

DOI: 10.59254/sbpo-2025-212078
https://proceedings.science/p/212078?lang=pt-br

$\lvert N\rvert$ (bits)	instances	trials	direct CF	inflation	extra bases	$\Pr[\gcd(r_p, r_q) = 1]$
128	1265	125088	47.9%	46.7%	5.4%	57.1%
256	26	2591	54.4%	42.9%	2.7%	46.2%

Table 1: Summary of single-run factoring outcomes. "direct CF" = factor via continued fractions only; "inflation" = success after exponent inflation+reduction; "extra bases" = success only after trying x_0, x_1.

So, if we have $JS(x, N) = -1$, then $\{LS(x, p) = 1$ and $LS(x, q) = -1\}$ or $\{LS(x, p) = -1$ and $LS(x, q) = 1\}$.

Without loss of generality, we suppose $LS(x, p) = 1$ and $LS(x, q) = -1$. In this case, with $LS(x, p) = 1$ we have $r_p = ord(x, p)$ divides $\frac{p-1}{2} = 2^{c-1}u$, where $ord(x, p)$ is the order of x modulo p. Then $2^c \not| r_p$ and $\gcd(2^c, r_p) < 2^c$.

On the other hand, $LS(x, q) = -1$ implies that $r_q = \mathrm{ord}(x, q)$ does not divide $\frac{q-1}{2} = 2^{c-1}v$. It means $2^c \mid r_q$. In other words, $\gcd(2^c, r_q) = 2^c$.

Taking into account that $r = ord(x, N) = lcm(r_p, r_q)$ (see Lemma 2.3 in [Bastos and Kowada, 2021]), so we have $\gcd(r, 2^c) = 2^c$. Therefore, $r/2$ is multiple of r_p, but $r/2$ is not multiple of r_q. In this way, $x^{r/2} \equiv 1 \bmod p$ but $x^{r/2} \not\equiv 1 \bmod q$. It means that $x^{r/2} - 1$ is a multiple of p but is not a multiple of q. As $N = p \cdot q$ then $\gcd(x^{r/2} - 1, N) = p$.

□

7. Results and Discussion

All experiments were executed as a standalone Python script (standard library only), on OS Ubuntu 22.04, Python 3.12.10, Intel Core i5-7300HQ with 16 GB of memory. No quantum SDKs were used; the simulator draws a synthetic peak and performs purely classical post-processing.

We evaluated 1265 composites of 128 bits and 26 composites of 256 bits, using continued fractions alone recover a non-trivial factor in 47.9% (128-bit) and 54.4% (256-bit) of the measurements. Most of the remaining cases are resolved by our exponent-inflation-and-reduction mechanism 46.7% and 42.9%, respectively. So the overall single-run factoring rate at the instance level is effective 100% in both sets as seen in Table 1. Across instances, we observe $\Pr[\gcd(r_p, r_q) = 1] \approx 57.1\%$ (128-bit) and $\approx 46.2\%$ (256-bit), consistent with the behavior expected for random primes p, q. The consistency and accuracy of these outcomes suggest that the proposed approach enhances the reliability and efficiency of the order-finding stage, positioning it as a practical refinement of Shor's algorithm.

8. Conclusion and Future Work

We aimed to develop a simulation tool capable of testing Shor's algorithm on large numbers, allowing us to validate existing assumptions and uncover new insights. Specifically, we aimed to verify established results and explore novel aspects. For example, we confirmed that employing the Jacobi symbol to select the random value x within Shor's algorithm enhances the success rate of the Quantum Order-Finding Algorithm (QOFA), often requiring its execution only once, a significant improvement when compared with the original approach [Shor, 1999]. This finding echoes the result obtained by [Grosshans et al., 2015], but our proposal offers a broader scope based on Theorem 1. Unlike the prior work limited to numbers of the form $N = p_1 \cdot p_2 = (2q_1 + 1)(2q_2 + 1)$, with $q_1 \neq q_2$ and $q_1, q_2 > 2$ our result applies to numbers of the form $N = p \cdot q = (2^c \cdot u + 1)(2^c \cdot v + 1)$, where u and v are integers, with c being a constant. Finally, for future work, we will expand our tests to include composite numbers larger than 256 bits.

DOI: 10.59254/sbpo-2025-212078
https://proceedings.science/p/212078?lang=pt-br

References

Bastos, D. C. and Kowada, L. A. B. (2021). How to detect whether shor's algorithm succeeds against large integers without a quantum computer. *Procedia Computer Science*, 195:145–151.

Ekerå, M. (2021). On completely factoring any integer efficiently in a single run of an order-finding algorithm. *Quantum Information Processing*, 20(6):205.

Grosshans, F., Lawson, T., Morain, F., and Smith, B. (2015). Factoring safe semiprimes with a single quantum query. *arXiv preprint arXiv:1511.04385*.

Miller, G. L. (1975). Riemann's hypothesis and tests for primality. In *Proceedings of the seventh annual ACM symposium on Theory of computing*, p. 234–239.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Shor, P. W. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332.

Tankasala, A. and Ilatikhameneh, H. (2019). Quantum-kit: simulating shor's factorization of 24-bit number on desktop. *arXiv preprint arXiv:1908.07187*.

Wang, D. S., Hill, C. D., and Hollenberg, L. C. (2017). Simulations of shor's algorithm using matrix product states. *Quantum Information Processing*, 16:1–13.

Willsch, D., Willsch, M., Jin, F., De Raedt, H., and Michielsen, K. (2023). Large-scale simulation of shor's quantum factoring algorithm. *Mathematics*, 11(19):4222.

DOI: 10.59254/sbpo-2025-212078
https://proceedings.science/p/212078?lang=pt-br