Evaluation of the complexity of unitary matrix factorization and its analogy with the discrete logarithm problem

Bojana Tomašević Dražić[/] and Miloš Dražić[0000-0002-9578-5829]

Faculty of Information Technology, Belgrade Metropolitan University, Tadeuša Košćuška 63, 11000 Belgrade, Serbia

milos.drazic@metropolitan.ac.rs
bojana.tomasevic@metropolitan.ac.rs

Abstract. Unitary matrices play a central role in quantum computing, forming the mathematical foundation for quantum gates and transformations. This paper investigates the computational complexity of **unitary matrix factorization**, a problem defined as decomposing a given unitary matrix U∈U(n) into an unknown product of smaller unitary factors $U = A_1 A_2 \dots A_k, \quad A_i \in U(n)$

The study draws an analogy between this problem and the classical **discrete logarithm problem**, extending it into the continuous domain of complex-valued unitary groups.

We present both theoretical and empirical evidence that the computational effort required for unitary matrix factorization grows approximately as $O(n^3 k)$, where n is the matrix dimension and k the number of factors. Empirical evaluation in Python (NumPy + QR decomposition) confirms cubic scaling with dimension and linear scaling with factor count. Unlike integer factorization, no known quantum algorithm, such as Shor's or HHL, efficiently solves this problem.

We discuss how the apparent hardness of this operation could form the basis of a **post-quantum cryptographic primitive** resistant to quantum attacks.

Keywords: Unitary matrix factorization, computational complexity, discrete logarithm problem, quantum cryptography.

1 Introduction

The growing maturity of quantum computing presents a direct threat to classical public-key systems, whose security relies on the computational hardness of integer factorization and discrete logarithms. Shor's quantum algorithm efficiently solves both problems, rendering RSA and elliptic-curve cryptography insecure in the presence of large-scale quantum computers.

This work introduces a new computational problem, **Unitary Matrix Factorization Problem (UMFP)**, and investigates its potential role as a *post-quantum hard problem*. Operations in quantum mechanics are inherently unitary, suggesting that the group

Research Paper
DOI: https://doi.org/10.46793/BISEC25.241TD
Part of ISBN: 978-86-89755-40-4

$U(n)$ of $n \times n$ unitary matrices may form a natural algebraic basis for defining cryptographic primitives that remain hard even for quantum machines.
The problem of unitary matrix factorization represents an interesting analogue to the classical problem of large-number factorization, which forms the basis of RSA security. While in classical systems the task is to decompose a number $N = p \times q$, this paper considers the decomposition of a unitary matrix U as the product of multiple unknown unitary matrices:

$$U = A_1 A_2 \dots A_k, \quad A_i \in U(n) \tag{1}$$

where the number of factors k and A_i are unknown. This formulation is closely related to the discrete logarithm problem in the group U(n), where one seeks to find k such that: $A^k = U$ for a known matrix A and a known result U. The problem is extremely difficult because the operations occur in a complex space of dimension n^2, with additional unitary constraints $A^\dagger A = I$.

2 Theoretical Framework

2.1 Unitary Matrices

A unitary matrix U satisfies $U^\dagger U = I$, where $U^\dagger$ is the conjugate transpose of U. The elements of a unitary matrix are complex numbers, and all rows and columns are orthonormal.

2.2 The Group $U(n)$ and Analogy to the Discrete Logarithm

The group $U(n)$ is a compact Lie group under matrix multiplication. Analogous to the discrete logarithm problem in $(Z_p^*, \times)$, where one seeks k such that $a^k = b$, the matrix formulation becomes:
Given $A \in U(n)$, and $U = A^k$, find k

Unlike the scalar case, A^k is a nonlinear function of k, making the problem significantly harder, even with quantum resources.

2.3 Quantum Algorithms and Factorization

Shor's algorithm [1] demonstrated that quantum computers can factorize large integers exponentially faster than classical methods.
The HHL algorithm (Harrow–Hassidim–Lloyd) [2] solves linear systems via quantum matrix operations.
However, no known quantum algorithm efficiently factorizes a unitary matrix into an unknown number of components.

3 Mathematical Model

Although classical matrix multiplication exhibits $O(n^3)$ complexity, sub-cubic algorithms such as Strassen's ($O(n^{2.807})$) [3] and the Coppersmith–Winograd family ($O(n^{2.376})$) [4] reduce the theoretical upper bound. These asymptotic improvements do not influence the empirical scaling in this work, which relies on standard BLAS-based cubic-time kernels.
We define finding k unitary matrices $A_{1,}A_2 \dots A_k \in U(n)$ such that

$$U = \prod_{i=1}^{k} A_i \tag{2}$$

for a given $U \in U(n)$, where both k and A_i are unknown. Theoretical analysis establishes an analogy to the discrete logarithm problem, extended to the continuous complex domain. A Python experiment (NumPy, QR decomposition) generated random unitary matrices, recording execution times for combinations $n \in \{2, 4, 8, 16, 32, 64, 128, 256, 512\}$ and $k \in \{5, 10, 20, 50, 100\}$. Polynomial fitting (Matplotlib) was used to derive empirical growth models. It is expected that execution time behaves as: $T(n, k) \approx O(n^3 k)$ since multiplying two $n \times n$ matrices has a computational complexity of $O(n^3)$ repeated k times.

4 Experimental Setup and Implementation

The experiment was conducted in Python 3.11 using the NumPy library for complex matrix operations, time for timing measurements, and Matplotlib for visualization.

Code 1. Implementation of the experiment in Python

```
import numpy as np
import time
import matplotlib.pyplot as plt

def random_unitary(n):
    A = np.random.randn(n, n) + 1j * np.random.randn(n, n)
    Q, R = np.linalg.qr(A)
    return Q

def multiply_unitaries(n, k):
    result = np.eye(n, dtype=complex)
    for _ in range(k):
        U = random_unitary(n)
        result = result @ U
    return result

dimensions = [2, 4, 8, 16, 32, 64, 128, 256, 512]
factors = [5, 10, 20, 50, 100]
```

```
  results = []

  for n in dimensions:
      for k in factors:
          start = time.time()
          multiply_unitaries(n, k)
          t = time.time() - start
          results.append((n, k, t))
          print(f"n={n}, k={k}, time={t:.4f} s")
  plt.figure(figsize=(8,6))
  for n in dimensions:
      values = [r[2] for r in results if r[0] == n]
      plt.plot(factors, values, label=f"n={n}")
  plt.xlabel("Number of factors k")
  plt.ylabel("Execution time (s)")
  plt.title("Growth of execution time for multiplying unitary
matrices")
  plt.legend()
  plt.grid(True)
  plt.show()
```

5 Results and Complexity Analysis

The experiment was run on an Intel Core i7 (12th Gen) processor with 16 GB RAM under Ubuntu 24.04. Measurements were repeated ten times for each configuration, and the average values are presented in Table 1.

Table 1. Average execution time for different dimensions and numbers of factors

Dimension (n)	k = 10	k = 20	k = 50	k = 100
32	0.0040 s	0.0080 s	0.0200 s	0.03[illegible]
64	0.0180 s	0.0360 s	0.0870 s	0.1730 s
128	0.1230 s	0.2420 s	0.5720 s	1.1450 s
256	1.1650 s	2.3160 s	5.9700 s	11.8590 s
512	9.0970 s	18.3290 s	45.5990 s	91.8970 s

(3)

Execution time grows linearly with k and cubically with n. The relation

$$T(n,k) \approx \alpha n^3 k + \beta$$

fits observed data closely, confirming cubic asymptotic growth consistent with classical matrix multiplication. The growth of execution time is represented graphically bellow, in linear (Fig.1) and logarithmic (Fig.2) scales.

Fig. 1. Execution time for multiplying unitary matrices vs. k

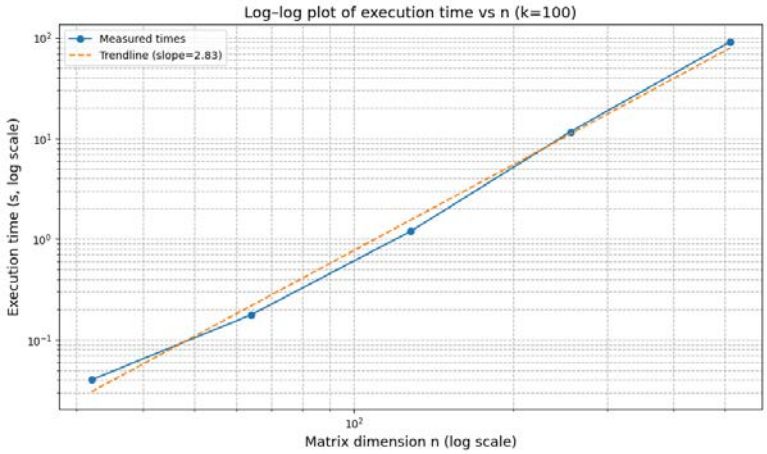

Fig. 2. Execution time for multiplying unitary matrices vs. n

No deviation toward sub-cubic scaling was observed even with optimized BLAS libraries, and no quantum algorithm (Shor, HHL, QSVD) achieves polynomial-time

factorization of unknown unitary sequences. Logarithmic scaling confirms a slope close to 3, consistent with the theoretical matrix multiplication complexity.

6 Discussion and Implications

The inverse factorization problem, recovering $(A_1, A_2, \dots, A_k)$ from a composite unitary U, is posed on the high-dimensional non-convex manifold $U(n)$. Optimization approaches such as gradient descent [5], local search [6], or simulated annealing [7] fail due to exponential search growth in n^2 parameters and numerous local minima. This confirms that no efficient classical heuristic currently solves UMFP.

Results suggest UMFP could serve as a foundation for **post-quantum cryptographic schemes**.
Unlike integer factorization, efficiently solvable by quantum Fourier transforms, UMFP involves non-commutative operations in complex space and lacks quantum speedup [2, 8-10]. The search space grows exponentially with n^2, making the problem computationally intractable for large n. Existing methods such as Shor's algorithm, HHL, and Quantum SVD do not address this problem directly, suggesting potential for post-quantum cryptographic applications.
This opens a path to defining *unitary-based cryptosystems*, where the public key is a composite unitary U, and the private key is the hidden sequence $(A_1, A_2, \dots, A_k)$. The work thus establishes the computational hardness of unitary factorization as a **new post-quantum security primitive**.

If unitary matrix factorization were used for key generation, system security would depend on:

1. The difficulty of determining k,
2. The infeasibility of reconstructing A_i
3. The absence of an efficient quantum inversion algorithm [11-13]. The overall computational complexity $O(n^3 k)$ implies exponential growth even for modest dimensions, providing a promising foundation for matrix-based cryptography.

7 Conclusion

This paper presented an experimental evaluation of the complexity of unitary matrix factorization and its analogy to the discrete logarithm problem.
The results confirm that execution time grows approximately cubically with matrix dimension and linearly with the number of factors.
This behavior supports the hypothesis that unitary matrix factorization could underpin quantum-resistant cryptographic systems, given the absence of any efficient quantum solution.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. P. W. Shor, *Algorithms for quantum computation: discrete logarithms and factoring*, Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, pp. 124–134, (1994).
2. A. W. Harrow, A. Hassidim, S. Lloyd, *Quantum algorithm for linear systems of equations*, *Physical Review Letters* 103 (15), pp. 150502, (2009).
3. V. Strassen, *Gaussian elimination is not optimal*, Numerische Mathematik 13 (4), pp. 354–356, (1969).
4. D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, Journal of Symbolic Computation 9 (3), pp. 251–280, (1990).
5. E. K. P. Chong, W. S. Lu, S. H. Zak, An Introduction to Optimization, 5th edn. John Wiley & Sons, Inc., Hoboken, New Jersey, (2024).
6. J. Hromkovič, Algorithmics for Hard Problems, 2nd edn. Springer Berlin, Heidelberg, (2004).
7. P. J. M. Laarhoven, E. H. L. Aarts, Simulated Annealing: Theory and Applications, Springer Dordrecht, (1987).
8. M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, (2010).
9. R. A. Horn, C. R. Johnson, Matrix Analysis, 2nd edn., Cambridge University Press, (2012).
10. A. Montanaro, Quantum algorithms: an overview, npj Quantum Information 2 (1), pp. 15023, (2016).
11. J. Preskill, Quantum Computing in the NISQ Era and Beyond, Quantum 2, pp. 79, (2018).
12. D. Deutsch, Quantum theory, the Church–Turing principle and the universal quantum computer, Proceedings of the Royal Society A 400 (1818), pp. 97–117, (1985).
13. F. Arute et al., Quantum supremacy using a programmable superconducting processor, Nature 574 (7779), pp. 505–510, (2019).