Empirical Analysis of Deterministic Factorization Methods

Systematic Evaluation of Classical and Alternative Approaches

Abstract

This work documents empirical results from systematic testing of various factorization algorithms. 37 test cases were conducted using Trial Division, Fermat's Method, Pollard Rho, Pollard $p-1$, and the T0-Framework. The primary purpose is to demonstrate that deterministic period finding is feasible. All results are based on direct measurements without theoretical evaluations or comparisons.

Contents

1 Methodology

Tested Algorithms

The following factorization algorithms were implemented and tested:

1. **Trial Division**: Systematic division attempts up to $\sqrt{n}$
2. **Fermat's Method**: Search for representation as difference of squares
3. **Pollard Rho**: Probabilistic period finding in pseudorandom sequences
4. **Pollard $p-1$**: Method for numbers with smooth factors
5. **T0-Framework**: Deterministic period finding in modular exponentiation (classical Shor-inspired)

Test Configuration

Table 1: Experimental Parameters

Parameter	Value
Number of test cases	37
Timeout per test	2.0 seconds
Number range	15 to 16777213
Bit size	4 to 24 bits
Hardware	Standard desktop CPU
Repetitions	1 per combination

Metrics

For each test, the following were recorded:

- **Success/Failure**: Binary result
- **Execution time**: Millisecond precision
- **Found factors**: For successful tests
- **Algorithm-specific parameters**: Depending on method

2 T0-Framework Feasibility Demonstration

Purpose of Implementation

The T0-Framework implementation serves as a proof-of-concept to demonstrate that deterministic period finding is technically feasible on classical hardware.

Implementation Components

The T0-Framework implements the following components to demonstrate deterministic period finding:

```
class UniversalT0Algorithm:
def __init__(self):
self.xi_profiles = {
 'universal': Fraction(1, 100),
 'twin_prime_optimized': Fraction(1, 50),
 'medium_size': Fraction(1, 1000),
 'special_cases': Fraction(1, 42)
}
self.pi_fraction = Fraction(355, 113)
self.threshold = Fraction(1, 1000)
```

Adaptive ξ-Strategies

The system uses different ξ-parameters based on number characteristics:

Table 2: ξ-Strategies in the T0-Framework

Strategy	ξ-Value	Application
twin_prime_optimized	$1/50$	Twin prime semiprimes
universal	$1/100$	General semiprimes
medium_size	$1/1000$	Medium-sized numbers
special_cases	$1/42$	Mathematical constants

Resonance Calculation

Resonance evaluation is performed using exact rational arithmetic:

$$\omega = \frac{2 \cdot \pi_{\text{ratio}}}{r} \tag{1}$$

$$R(r) = \frac{1}{1 + \left|\frac{-(\omega-\pi)^2}{4\xi}\right|} \tag{2}$$

3 Experimental Results: Proof of Concept

The experimental results serve to demonstrate the feasibility of deterministic period finding rather than to compare algorithmic performance.

Success Rates by Algorithm

Table 3: Overall success rates of all algorithms

Algorithm	Successful tests	Success rate (%)
Trial Division	37/37	100.0
Fermat	37/37	100.0
Pollard Rho	36/37	97.3
Pollard $p-1$	12/37	32.4
T0-Adaptive	31/37	83.8

4 Period-based Factorization: T0, Pollard Rho, and Shor's Algorithm

Comparison of Period Finding Approaches

T0-Framework, Pollard Rho, and Shor's quantum algorithm are all period-finding algorithms with different computational paradigms:

Table 4: Period-Finding Algorithms Comparison

Aspect	Pollard Rho	T0-Framework	Shor's Algorithm
Computation	Classical prob.	Classical det.	Quantum
Period detect	Floyd cycle	Resonance analysis	Quantum FT
Arithmetic	Modular	Exact rational	Quantum superpos.
Reproducibility	Variable	100% reprod.	Prob. measurement
Sequence gen	$f(x) = x^2 + c$ mod n	$a^r \equiv 1 \pmod{n}$	a^x mod n
Success crit	gcd$(\lvert x_i - x_j\rvert, n) > 1$	Resonance thresh.	Period from QFT
Complexity	$O(n^{1/4})$ expect.	$O((\log n)^3)$ theor.	$O((\log n)^3)$ theor.
Hardware	Classical comp.	Classical comp.	Quantum comp.
Practical limit	Birthday paradox	Resonance tuning	Quantum decoher.

Shared Period-Finding Principle

All three algorithms exploit the same mathematical foundation:

- **Core idea**: Find period r where $a^r \equiv 1 \pmod{n}$

- **Factor extraction**: Use period to compute gcd$(a^{r/2} \pm 1, n)$
- **Mathematical basis**: Euler's theorem and order of elements in $\mathbb{Z}_n^*$

Theoretical Complexity Analysis

Both T0-Framework and Shor's algorithm share the same theoretical complexity advantage:

- **Period search space**: Both search for periods r where $a^r \equiv 1 \pmod{n}$
- **Maximum period**: The order of any element is at most $n-1$, but typically much smaller
- **Expected period length**: $O(\log n)$ for most elements due to Euler's theorem
- **Period testing**: Each period test requires $O((\log n)^2)$ operations for modular exponentiation
- **Total complexity**: $O(\log n) \times O((\log n)^2) = O((\log n)^3)$

The Shared Polynomial Advantage

Both T0 and Shor's algorithm achieve the same theoretical breakthrough:

$$\text{Classical exponential: } O(2^{\sqrt{\log n \log \log n}}) \rightarrow \text{Polynomial: } O((\log n)^3) \tag{3}$$

The key insight is that **both algorithms exploit the same mathematical structure**:

- Period finding in the group $\mathbb{Z}_n^*$
- Expected period length $O(\log n)$ due to smooth numbers
- Polynomial-time period verification
- Identical factor extraction method

The only difference: Shor uses quantum superposition to search periods in parallel, while T0 searches them deterministically in sequence - but both have the same $O((\log n)^3)$ complexity bound.

The Implementation Paradox

Both T0 and Shor's algorithm demonstrate a fundamental paradox in advanced algorithmic design:

Core Problem

Perfect Theory, Imperfect Implementation:
Both algorithms achieve the same theoretical breakthrough from exponential to polynomial complexity, but practical implementation overhead completely negates these theoretical advantages.

Shared Implementation Failures

- **Shor's quantum overhead**:
  - Quantum error correction requires $\sim 10^6$ physical qubits per logical qubit
  - Decoherence times limit algorithm execution
  - Current systems: 1000 qubits → Need: 10^9 qubits for RSA-2048
- **T0's classical overhead**:
  - Exact rational arithmetic: Fraction objects grow exponentially in size
  - Resonance evaluation: Complex mathematical operations per period
  - Adaptive parameter tuning: Multiple ξ-strategies increase computational cost

5 Philosophical Implications: Information and Determinism

Intrinsic Mathematical Information

A crucial insight emerges from this analysis that extends beyond computational complexity:

Fundamental Principle

No Superdeterminism Required:
All information that can be extracted from a number through factorization algorithms is intrinsically contained within the number itself. The algorithms merely reveal pre-existing mathematical relationships - they do not create information.

Vibrational Modes and Predictive Patterns

A deeper analysis reveals that number size constrains the possible "vibrational modes" in factorization:

Vibrational Constraint Principle

Size-Determined Mode Space:
The size of a number n predetermines the boundaries of possible oscillation modes. Within these boundaries, only specific resonance patterns are mathematically possible, and these follow predictable patterns that enable "looking into the future" of the factorization process.

Constrained Oscillation Space

For a number n with $k = \log_2(n)$ bits:

- **Maximum period:** $r_{\text{max}} = \lambda(n) \leq n-1$ (Carmichael function)
- **Typical period range:** $r_{typical} \in [1, O(\sqrt{n})]$ for most bases

- **Resonance frequencies:** $\omega = 2\pi/r$ constrained to discrete values
- **Vibrational modes**: Only $O(\sqrt{n})$ distinct oscillation patterns possible

The Bounded Universe of Oscillations

$$\Omega_n = \left\{\omega_r = \frac{2\pi}{r} : r \in \mathbb{Z}, 2 \leq r \leq \lambda(n)\right\} \quad (4)$$

This frequency space Ω_n is:

- **Finite**: Constrained by number size
- **Discrete**: Only integer periods allowed
- **Structured**: Follows mathematical patterns based on n's prime structure
- **Predictable**: Resonance peaks cluster in mathematically determined regions

Predictive Principle

Mathematical Foresight:
By analyzing the constrained oscillation space and recognizing structural patterns, it becomes possible to predict which periods will yield strong resonances without exhaustively testing all possibilities. This represents a form of mathematical "future sight" - not mystical, but based on deep pattern recognition in number-theoretic structures.

6 Neural Network Implications: Learning Mathematical Patterns

Machine Learning Potential

If mathematical patterns in oscillation modes are predictable through pattern recognition, then neural networks should inherently be capable of learning these patterns:

Neural Network Hypothesis

Learnable Mathematical Patterns:
Since the vibrational modes and resonance patterns follow mathematically deterministic rules within constrained spaces, neural networks should be able to learn to predict optimal factorization strategies without exhaustive search.

Training Data Structure

The experimental data provides perfect training material:

- **Input features**: Number size, bit length, mathematical type (twin prime, smooth, etc.)
- **Target predictions**: Optimal ξ-strategy, expected resonance periods, success probability
- **Pattern examples**: 37 test cases with documented success/-failure patterns
- **Feature engineering**: Extract mathematical invariants (prime gaps, smoothness, etc.)

Learning Mathematical Invariants

Neural networks could learn to recognize:

Table 5: Learnable Mathematical Patterns

Math Pattern	**NN Learning Target**
Twin prime struct.	Predict $\xi = 1/50$ strategy
Prime gap distrib.	Estimate reson. clustering
Smoothness indic.	Predict period distrib.
Math constants	ID multi-reson. patterns
Carmichael patterns	Estimate max period bounds
Factor size ratios	Predict optimal base select.

Bibliography

[1] Python Software Foundation. (2023). *fractions — Rational numbers*. Python 3.9 Documentation.

[2] Pollard, J. M. (1975). A Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15(3), 331–334.

[3] Fermat, P. de (1643). *Methodus ad disquirendam maximam et minimam*. Historical source.

[4] Knuth, D. E. (1997). *The art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley.

[5] Cohen, H. (2007). *Number theory volume I: Tools and diophantine equations*. Springer Science & Business Media.