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

0.1 Methodology

0.1.1 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)

0.1.2 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

0.1.3 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

0.2 T0-Framework Feasibility Demonstration

0.2.1 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.

0.2.2 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)
```

0.2.3 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

0.2.4 Resonance Calculation

Resonance evaluation is performed using exact rational arithmetic:

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

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

0.3 Experimental Results: Proof of Concept

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

0.3.1 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

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

0.4.1 Comparison of Period Finding Approaches

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

0.4.2 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 $\mathbb{Z}_n^*$

0.4.3 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**: $O(\log n) \times O((\log n)^2) = O((\log n)^3)$

0.4.4 The Shared Polynomial Advantage

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

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

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.

0.4.5 The Implementation Paradox

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

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.