ON FACTORING AND POWER DIVISOR PROBLEMS VIA RANK-3 LATTICES AND THE SECOND VECTOR

YIMING GAO, YANSONG FENG, HONGGANG HU, AND YANBIN PAN

Abstract. We propose a deterministic algorithm based on Coppersmith's method that employs a rank-3 lattice to address factoring-related problems. An interesting aspect of our approach is that we utilize the second vector in the LLL-reduced basis to avoid trivial collisions in the Baby-step Giant-step method, rather than the shortest vector as is commonly used in the literature. Our results are as follows:

- Compared to the result by Harvey and Hittmeir (Math. Comp. 91 (2022), 1367–1379), who achieved a complexity of $O\left(\frac{N^{1/5}\log^{16/5}N}{(\log\log N)^{3/5}}\right)$ for factoring a semiprime $N = pq$, we demonstrate that in the balanced p and q case, the complexity can be improved to $O\left(\frac{N^{1/5}\log^{13/5}N}{(\log\log N)^{3/5}}\right)$.

- For factoring sums and differences of powers, i.e., numbers of the form $N = a^n \pm b^n$, we improve Hittmeir's result (Math. Comp. 86 (2017), 2947–2954) from $O(N^{1/4}\log^{3/2}N)$ to $O\left(N^{1/5}\log^{13/5}N\right)$.

- For the problem of finding r-power divisors, i.e., finding all integers p such that $p^r \mid N$, Harvey and Hittmeir (Proceedings of ANTS XV, Res. Number Theory 8 (2022), no.4, Paper No. 94) recently directly applied Coppersmith's method and achieved a complexity of $O\left(\frac{N^{1/4r}\log^{10+\epsilon}N}{r^3}\right)$. By using faster LLL-type algorithm and sieving on small primes, we improve their result to $O\left(\frac{N^{1/4r}\log^{7+3\epsilon}N}{(\log\log N-\log 4r)r^{2+\epsilon}}\right)$. The worst case running time for their algorithm occurs when $N = p^r q$ with $q = \Theta(N^{1/2})$. By focusing on this case and employing our rank-3 lattice approach, we achieve a complexity of $O\left(\sqrt{r}N^{1/4r}\log^{5/2}N\right)$. In conclusion, we offer a new perspective on these problems, which we hope will provide further insights.

1. Introduction

This paper focuses on deterministic and rigorous integer factorization algorithms implemented on a Turing machine. It is worth noting that when these constraints are relaxed, superior complexity bounds have been achieved through various methods, including the Elliptic Curve Method (ECM) [LJ87], the General Number Field Sieve [CP05, §6.2], and Shor's Quantum factoring algorithm [Sho94].

In [Har21], Harvey demonstrated that a positive integer N can be rigorously and deterministically factored into primes in time $F(N) = O\left(N^{1/5}\log^{16/5}N\right)$, where time is measured in bit operations—specifically, the number of steps executed by a deterministic Turing machine with a fixed, finite number of linear tapes. Subsequently, Harvey and Hittmeir [HH22b] improved this bound to $F(N) = O\left(\frac{N^{1/5}\log^{16/5}N}{(\log\log N)^{3/5}}\right)$. Our work adopts the same computational model as [Har21] and [HH22b].

We demonstrate that in certain common scenarios, such as when $N = pq$ is a semiprime with $p, q = \Theta(N^{1/2})$, this complexity can be further refined using the Coppersmith method in conjunction with the Baby-step Giant-step framework. Our primary result is as follows:

Theorem 1.1 (Deterministic integer factorization)**.** *Let $N = pq$ be a semiprime with $p, q = \Theta(N^{1/2})$. Then there exists a deterministic algorithm to recover the factors p and q in time*

$$F(N) = O\left(\frac{N^{1/5}\log^{13/5} N}{(\log\log N)^{3/5}}\right).$$

In earlier work, Hittmeir [Hit17] presented an algorithm for factoring sums and differences of powers with time complexity $F(N) = O(N^{1/4}\log^{3/2} N)$. While this complexity can be improved to $F(N) = O\left(\frac{N^{1/5}\log^{16/5} N}{(\log\log N)^{3/5}}\right)$ using [HH22b], such an approach does not exploit the specific properties of sums and differences of powers. Our framework naturally leverages these properties, leading to our second main result:

Theorem 1.2 (Factoring sums and differences of powers)**.** *Let $a, b \in \mathbb{N}$ be fixed and coprime such that $a > b$, and define $P_{a,b} := \{a^n \pm b^n : n \in \mathbb{N}\}$. Then, one may compute the prime factorization of any $N \in P_{a,b}$ in time*

$$F(N) = O\left(N^{1/5}\log^{13/5} N\right).$$

We briefly introduce the ideas underpinning the algorithms presented in [Har21] and [HH22b]. The strategy in [Har21] leverages Fermat's Little Theorem. For an integer α coprime to $N = pq$, and integers a, b, we have $\alpha^{aq+bp} \equiv \alpha^{aN+b} \pmod{p}$. If a/b is chosen as a good rational approximation to the unknown ratio p/q, then the value $X = aq + bp$ is close to $K = \lfloor(4abN)^{1/2}\rfloor$. This leads to the congruence $\alpha^{X-K} \equiv \alpha^{aN+b-K} \pmod{p}$. Since $X - K$ is small under the approximation assumption, the core idea is to find a collision modulo p (and hence potentially modulo N) between "baby steps" of the form $\alpha^i \pmod{N}$ for small integers i, and "giant steps" of the form $\alpha^{aN+b-K} \pmod{N}$ as a/b varies over a suitable dense set of rational approximations. This collision-finding problem is efficiently addressed using techniques from fast polynomial arithmetic, resulting in the complexity bound $O(N^{1/5}\log^{16/5} N)$.

The subsequent work [HH22b] refines this approach by incorporating the crucial observation that the prime factors p and q of a large integer N cannot themselves be divisible by small primes. The algorithm is modified to restrict the search space by considering only candidates for p that are coprime to $m = p_1 p_2 \cdots p_d$, the product of the first d primes, for a suitably chosen d such that $m = N^{O(1)}$. The number of residue classes modulo m that need consideration is reduced by a factor of $m/\varphi(m)$, which, by Mertens' theorem, behaves asymptotically like $\log d \approx \log\log m$. When optimized, this yields a saving proportional to $\log\log N$. Technically, this refinement involves a reorganization where suitable pairs (a, b) for the giant steps are no longer generated by simple iteration over a range, but are instead computed using algorithms for finding short vectors in appropriately constructed lattices. This leads to the improved complexity bound $O\left(\frac{N^{1/5}\log^{16/5} N}{(\log\log N)^{3/5}}\right)$.

Our work builds upon the foundational Baby-step Giant-step framework and the sieving techniques introduced in [Har21, HH22b]. We retain the overall strategy of seeking collisions and incorporate the log-log optimization derived from restricting prime factor candidates modulo a product of small primes m.

The primary departure and main contribution of this paper lies in a novel method for constructing the "giant steps", applicable both to balanced semiprimes and cases with additional modular information. While [HH22b] employs lattice reduction to find pairs (a, b) such that a/b approximates p/q, our approach adapts Coppersmith's method more directly. We construct a specific rank-3 lattice based on known congruences of the target prime factor p (such as $p \pmod m$ or $p \pmod{mn}$) and its approximate size. By applying lattice basis reduction (LLL) to this lattice—albeit relaxing the strict determinant conditions typically required by the Howgrave-Graham lemma and utilizing information from the second vector—we derive coefficients for computing our giant steps.

In essence, by leveraging known properties of the prime factor p (either its approximate size or a specific congruence) through the Coppersmith-style lattice, our method performs a more targeted search compared to implicitly searching through rational approximations. When applied to balanced semiprimes where $p = \Theta(N^{1/2})$, this more efficient use of the factor's bit-length information contributes significantly to reducing the search space, ultimately saving a logarithmic factor in the complexity analysis and leading to the improved bound $O\left(\frac{N^{1/5}\log^{13/5} N}{(\log\log N)^{3/5}}\right)$ presented in Theorem 1.1. Furthermore, we extend this lattice-based technique to scenarios where partial information $p \equiv r \pmod n$ is available. This adaptation yields a deterministic factoring algorithm with complexity explicitly dependent on n, refining previous results [Hit17]. As a key application, this allows for factoring numbers of the form $N = a^n \pm b^n$ in time $O(N^{1/5}\log^{13/5} N)$ (Theorem 1.2), naturally utilizing the inherent information $(ab^{-1})^{2n} \equiv 1 \pmod N$ where $n = \Theta(\log N)$ arises.

We also propose some improved toolkits. We design a proper module choosing algorithm for the loglog speed-up. Furthermore, we present several improvements to existing toolkits in factorization algorithms. While previous works by [Hit18, Har21, HH22b] required finding an element α with order $\mathrm{ord}_N(\alpha) > N^{2/5}$ or $\mathrm{ord}_N(\alpha) > N^{2/(3+2r)}$, we demonstrate how to reduce these requirements to $\mathrm{ord}_N(\alpha) > N^{1/4+o(1)}$ and $\mathrm{ord}_N(\alpha) > N^{1/(4r)+o(1)}$ respectively. This advancement is achieved through our refined generalization of Harvey's deterministic factorization method for finding r-power divisors.

Finally, we extend our rank-3 lattice construction to address the r-power divisor problem of finding all integers p such that $p^r \mid N$. While recent work by Hales and Hiary [HH24] achieved complexities of $O(N^{1/(r+2)}(\log N)^2 \log\log N)$ and $O(N^{1/(3+2r)}(\log N)^{16/5})$ by extending Lehman's method, and Harvey and Hittmeir [HH22a] obtained $O(N^{1/4r}\log^{10+\epsilon} N/r^3)$ using Coppersmith's method, we present two improvements. First, by incorporating faster LLL-type lattice reduction algorithms and small prime sieving, we improve the complexity to $O\left(\frac{N^{1/4r}\log^{7+3\epsilon} N}{(\log\log N - \log 4r)r^{2+\epsilon}}\right)$. Then, focusing on the worst-case scenario where $N = p^r q$ with $p = \Theta(N^{1/2r})$ and $q = \Theta(N^{1/2})$, our rank-3 lattice construction further reduces the complexity to $O(\sqrt{r}N^{1/4r}\log^{5/2} N)$.

2. Preliminaries

2.1. **Notations.** Throughout this paper, we use $\log N$ to denote the binary logarithm $\log_2 N$. For asymptotic complexity analysis, we employ the standard Bachmann-Landau notation: For functions $f, g : \mathbb{N} \to \mathbb{R}^+$, we write $f = O(g)$ if there exist positive constants c and n_0 such that $f(n) \leqslant cg(n)$ for all $n \geqslant n_0$; we write $f = o(g)$ if $\lim_{n\to\infty} f(n)/g(n) = 0$; and we write $f = \Theta(g)$ if both $f = O(g)$ and $g = O(f)$ hold, or equivalently, if there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)$ for all $n \geqslant n_0$.

2.2. **Arithmetics.**

2.2.1. *Integer Arithmetic.* We recall some result about integer and modular arithmetic.

Let $n \in \mathbb{N}^+$, and suppose we are given integers $x, y \in \mathbb{Z}$ satisfying $|x|, |y| \leqslant 2^n$. The fundamental arithmetic operations exhibit the following computational complexities: The computations of $x + y$ and $x - y$ can be performed in $O(n)$ time. Let $\mathsf{M}(n)$ denote the time complexity of computing the product xy; as established in [HVDH21], we have $\mathsf{M}(n) = O(n \log n)$. For $y > 0$, both floor division $\lfloor x/y \rfloor$ and ceiling division $\lceil x/y \rceil$ can be computed in $O(\mathsf{M}(n))$ time, and consequently, the computation of $x \bmod y \in [0, y)$ also requires $O(\mathsf{M}(n))$ time. More generally, for any fixed rational number $u/v \in \mathbb{Q}^+$ and positive integers $x, y > 0$, both $\lfloor (x/y)^{u/v} \rfloor$ and $\lceil (x/y)^{u/v} \rceil$ can be computed in $O(\mathsf{M}(n))$ time. Using the half-GCD algorithm, for any $x, y \in \mathbb{Z}$, we can compute both $g = \gcd(x, y)$ and the Bézout coefficients $u, v \in \mathbb{Z}$ satisfying $ux + vy = g$ in time $O(\mathsf{M}(n) \log n) = O(n \log^2 n)$. In particular, when $\gcd(x, y) = 1$, the modular multiplicative inverse of x modulo y can be computed in $O(\mathsf{M}(n) \log n)$ time.

For any integer $N \geqslant 2$, we consider the ring of integers modulo N, denoted by $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$. Elements of $\mathbb{Z}_N$ are canonically represented by their residues in $[0, N)$, requiring at most $\lceil \log_2 N \rceil$ bits for storage. Let $\mathbb{Z}_N^*$ denote the multiplicative group of units in $\mathbb{Z}_N$, that is, $\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N : \gcd(x, N) = 1\}$. For the ring $\mathbb{Z}_N$, we have the following computational results: Given $x, y \in \mathbb{Z}_N$, modular addition and subtraction $x \pm y \bmod N$ can be computed in $O(\log N)$ time, while modular multiplication $xy \bmod N$ requires $O(\mathsf{M}(\log N)) = O(\log N \log\log N)$ time. For $x \in \mathbb{Z}_N$ and $m \in \mathbb{N}_0$, the computation of $x^m \bmod N$ can be achieved in $O(\mathsf{M}(\log N) \log m)$ time using the repeated squaring algorithm. Furthermore, testing whether $x \in \mathbb{Z}_N^*$ can be performed in $O(\mathsf{M}(\log N) \log\log N)$ time by computing $\gcd(x, N)$.

2.2.2. *Polynomial Arithmetic.* The next few results are taken from the previous papers [Hit18],[Har21] and [HH22b].

Lemma 2.1 (Polynomial Construction)**.** *Let $n \geqslant 1$ with $n = O(N)$. Given as input $v_1, \ldots, v_n \in \mathbb{Z}_N$, we may compute the polynomial $f(x) = (x - v_1) \cdots (x - v_n) \in \mathbb{Z}_N[x]$ in time $O(n \lg^3 N)$.*

Lemma 2.2 (Polynomial Evaluation)**.** *Given as input an element $\alpha \in \mathbb{Z}_N^n$, positive integers $m, n = O(N)$, and $f \in \mathbb{Z}_N[x]$ of degree n, we may compute $f(1), f(\alpha), \ldots, f(\alpha^{m-1}) \in \mathbb{Z}_N$ in time*

$$O((n + m) \lg^2 N).$$

Proof. This is exactly in [Har21, HH22b]. The proof leverages Bluestein's trick to compute polynomial evaluations efficiently by transforming the problem into a Laurent polynomial multiplication. □

2.3. **Lattice.** We begin by recalling fundamental concepts of lattices and basis reduction. Although our work exclusively employs integer lattices, all definitions extend naturally to real-valued lattices.

Let $v_1, \ldots, v_n \in \mathbb{Z}^m$ with $m \geqslant n$ be linearly independent vectors. The lattice $\mathcal{L}$ generated by $\{v_1, \ldots, v_n\}$ is defined as

$$\mathcal{L} = \left\{ \sum_{i=1}^{n} a_i v_i \mid a_i \in \mathbb{Z} \right\} \subseteq \mathbb{Z}^m.$$

When $m = n$, the lattice is said to be of full rank. A basis $\mathbf{B} = \{v_1, \ldots, v_n\}$ spans $\mathcal{L}$, and we call $n = \dim(\mathcal{L})$ the lattice dimension.

Given a basis $\mathbf{B}$, let $v_1^*, \ldots, v_n^*$ denote its Gram-Schmidt orthogonalization. The lattice determinant is

$$\det(\mathcal{L}) := \prod_{i=1}^{n} \|v_i^*\|,$$

where $\|\cdot\|$ denotes Euclidean norm. Any lattice $\mathcal{L}$ admits infinitely many bases, yet all share the same determinant. For full-rank lattices, this equals the absolute value of the determinant of the basis matrix $[v_1 \cdots v_n]^\top$.

Let $\mathcal{B}_m(0, r) := \{\mathbf{x} \in \mathbb{R}^m : \|\mathbf{x}\| < r\}$ denote the m-dimensional open ball of radius r centered at the origin (subscript omitted when clear from context). The successive minima $\lambda_1, \ldots, \lambda_n$ of a rank n lattice satisfy that $\lambda_i(\mathcal{L})$ is the smallest radius of a ball centered at the origin containing i linearly independent lattice vectors.

Minkowski's Second theorem establishes a fundamental connection between a lattice's determinant and its successive minima.

Theorem 2.3 (Minkowski's Second Theorem)**.** *For any rank n lattice $\mathcal{L}$ with basis $\mathbf{B}$, the successive minima satisfy*

$$\prod_{i=1}^{n} \lambda_i(\mathcal{L}) < (\sqrt{n})^n \det(\mathbf{B}).$$

This bound is non-constructive. For the special case of dimension 2, the classical Gauss reduction algorithm efficiently computes a shortest lattice vector. In arbitrary dimensions, we employ the seminal lattice reduction technique by Lenstra, Lenstra, and Lovász [LLL82]:

Lemma 2.4 (LLL Reduction)**.** *Let $\mathcal{L}(\mathbf{B}) \subseteq \mathbb{Z}^m$ be a lattice with basis $\{b_1, \ldots, b_n\}$. The LLL algorithm outputs a reduced basis $\{v_1, \ldots, v_n\}$ satisfying*

$$\|v_i\| \leqslant 2^{(n-1)/2} \lambda_i(\mathcal{L}) \quad (1 \leqslant i \leqslant n),$$

with time complexity $O(n^4 m \beta^2 (\log n + \log \beta))$, where $\beta = \Theta(\log \|\mathbf{B}\|_{\max} + \log m)$ encodes the basis vector bit-length.

Remark 2.5. By Corollary 17.5.4 in Mathematics of Public Key Cryptography, Version 2.0 by Steven D. Galbraith (October 31, 2018) [Gal12], we know that the LLL algorithm requires $O(n^3 m \beta)$ arithmetic operations on integers of size $O(n\beta)$.

Here, by employing integer multiplication with complexity $O(n \log n)$ for two n-bit integers as described in [HVDH21], we obtain our LLL complexity result.

Remark 2.6. If we only require the first relative shortest vector, we can apply the lattice reduction algorithm proposed by Neumaier and Stehlé [NS16], which improves the time complexity to $O(n^{4+\epsilon}\beta^{1+\epsilon})$.

2.4. **Coppersmith's Method.** In Coppersmith's method, the first step is to construct a lattice, where each row corresponds to the coefficient vector of a polynomial sharing the same root modulo M. To find small roots of a modular univariate polynomial f, we require a polynomial that shares the same root with f not only modulo M but also over $\mathbb{Z}$. To achieve this, we use Howgrave-Graham's following result [HG97, HG01], which establishes that small modular roots of a polynomial h with small coefficients are indeed integer roots of h.

Lemma 2.7 (Howgrave-Graham [HG97])**.** *Let $h(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ be the sum of at most w monomials, and let $X_1, \ldots, X_n > 0$. For any $(y_1, \ldots, y_n) \in \mathbb{Z}^n$ satisfying the following two conditions:*

(1) $h(y_1, \ldots, y_n) \equiv 0 \pmod{M}$ where $|y_i| < X_i$ for $1 \leqslant i \leqslant n$,
(2) $\|h(x_1X_1, \ldots, x_nX_n)\| < \frac{1}{\sqrt{w}}M$,

then $h(y_1, \ldots, y_n) = 0$.

In 1996, Coppersmith proposed a result called *factoring with a hint* [Cop96], which factors RSA moduli $N = pq$ in polynomial time given only half of the bits of p.

Lemma 2.8 ([May03], Theorem 7)**.** *Let N be an integer of unknown factorization with a divisor $b \geqslant N^{\beta}$. Let $f_\delta(x)$ be a univariate, monic polynomial of degree δ. Then all solutions x_0 to the equation*

$$f_\delta(x) \equiv 0 \pmod{b} \quad \text{with} \quad |x_0| \leqslant cN^{\beta^2/\delta}$$

can be found in time $O\left(\lceil c \rceil \frac{\log^{6+3\epsilon} N}{\delta^{1+\epsilon}}\right)$.

Proof. We begin by defining

$$X \coloneqq \frac{1}{2}N^{\frac{\beta^2}{\delta} - \frac{1}{\log N}}, \quad n \coloneqq \lceil \log N + 1 \rceil, \quad m \coloneqq \left\lceil \frac{\beta n}{\delta} \right\rceil.$$

We construct a set G of polynomials where each polynomial has a root x_0 modulo b^m whenever $f_b(x)$ has the root x_0 modulo b. The set comprises:

$$\begin{array}{llll} N^m & xN^m & x^2N^m & \cdots & x^{\delta-1}N^m \\ N^{m-1}f & xN^{m-1}f & x^2N^{m-1}f & \cdots & x^{\delta-1}N^{m-1}f \\ N^{m-2}f^2 & xN^{m-2}f^2 & x^2N^{m-2}f^2 & \cdots & x^{\delta-1}N^{m-2}f^2 \\ \vdots & \vdots & \vdots & & \vdots \\ Nf^{m-1} & xNf^{m-1} & x^2Nf^{m-1} & \cdots & x^{\delta-1}Nf^{m-1}. \end{array}$$

Additionally, we include:

$$f^m,\ xf^m,\ x^2f^m,\ \ldots,\ x^{n-\delta m}f^m.$$

Let L be the lattice spanned by the coefficient vectors of $g_{i,j}(xX)$ and $h_i(xX)$. The basis matrix B of L is lower triangular, yielding:

$$\det(L) = N^{\frac{1}{2}\delta m(m+1)}X^{\frac{1}{2}n(n-1)}.$$

Applying Lemma 2.7, we require:

$$2^{\frac{n-1}{4}} \det(L)^{\frac{1}{n}} < \frac{b^m}{\sqrt{n}}.$$

Using $b \geqslant N^\beta$, this yields:

$$N^{\frac{\delta m(m+1)}{2n}} X^{\frac{n-1}{2}} \leqslant 2^{-\frac{n-1}{4}} n^{-\frac{1}{2}} N^{\beta m}.$$

For X, we obtain:

$$X \leqslant 2^{-\frac{1}{2}} n^{-\frac{1}{n-1}} N^{\frac{2\beta m}{n-1} - \frac{\delta m(m+1)}{n(n-1)}}.$$

For $n \geqslant 7$, $n^{-\frac{1}{n-1}} = 2^{-\frac{\log n}{n-1}} \geqslant 2^{-\frac{1}{2}}$, simplifying to:

$$X \leqslant \frac{1}{2} N^{\frac{2\beta m}{n-1} - \frac{\delta m(m+1)}{n(n-1)}}.$$

Given our choice of X, it suffices to show:

$$\frac{m(2\beta n - \delta(m+1))}{n(n-1)} \geqslant \frac{\beta^2}{\delta} - \frac{1}{\log N}$$

Substituting $m = \beta n/\delta$ and simplifying yields:

$$\frac{1}{\log N} \geqslant \frac{\beta(\delta - \beta)}{\delta(n-1)}.$$

This is equivalent to:

$$n - 1 \geqslant \frac{\beta(\delta - \beta)}{\delta} \log N.$$

Since $0 < \beta \leqslant 1$, we have $\frac{\beta(\delta-\beta)}{\delta} \log N < \log N$, which holds by our choice of n.

Note that our choice of X only covers solutions in $[-X, X]$. To find all solutions in $[-cN^{\frac{\beta^2}{\delta}}, cN^{\frac{\beta^2}{\delta}}]$, we solve the problem for at most $4\lceil c \rceil$ translated polynomials to cover the full range. At last we apply a root-finding algorithm on each polynomial. For a single polynomial, the total time complexity is dominated by the LLL algorithm. Note that we only require the first relative shortest vector, so we could apply the faster lattice reduction algorithm in Remark 2.6. With lattice dimension $n = O(\log N)$ and basis vector bit-length $\log(N^m) = m \log N$, the complexity is:

$$O(\lceil c \rceil \log^{4+\epsilon} N (m \log N)^{1+\epsilon}) = O\left(\lceil c \rceil \frac{\log^{6+3\epsilon} N}{\delta^{1+\epsilon}} \right).$$

□

2.5. **Prime Distribution.** Following the idea of [HH22b], we also need the prime distribution lemma for the loglog speedup.

Lemma 2.9. *For $B \to \infty$ we have*

$$\sum_{\substack{2 \leqslant r \leqslant B \\ r \text{ prime}}} \log r = (1 + o(1))B, \tag{2.1}$$

$$\prod_{\substack{2 \leqslant r \leqslant B \\ r \text{ prime}}} \frac{r-1}{r} = \Theta\left(\frac{1}{\log B} \right). \tag{2.2}$$

To choose a proper module, we design the following lemma:

Lemma 2.10. *For sufficiently large input $x \in \mathbb{N}$, there exists a deterministic polynomial-time algorithm that outputs an integer $m = \prod_{p\in S} p$ where S is a set of distinct primes, satisfying:*

$$\frac{x}{2} < m < 2x, \quad \frac{\varphi(m)}{m} = \Theta\left(\frac{1}{\log\log x}\right)$$

Proof. Label the first n primes by $p_1, p_2, \ldots, p_n$. We compute the product $\prod_{i=1}^{n} p_i$ step by step, stopping at the first stage where the product exceeds x. Suppose this occurs at the k-th prime. Then

$$\prod_{i=1}^{k-1} p_i < x \leq \prod_{i=1}^{k} p_i.$$

Define $t := \left\lfloor \prod_{i=1}^{k} p_i / x \right\rfloor$. By Bertrand's postulate, there exists a prime p with $t < p < 2t$. We scan integers from $t+1$ to $2t-1$ to find p_s, the prime closest to t. We claim that $p_s \leqslant p_k$. Indeed,

$$p_k = \frac{\prod_{i=1}^{k} p_i}{\prod_{i=1}^{k-1} p_i} > \frac{\prod_{i=1}^{k} p_i}{x} \geqslant t.$$

Define $m := \prod_{i=1}^{k} p_i / p_s$, we have

$$\frac{x}{2} \leq \frac{\prod_{i=1}^{k} p_i}{2t} < m = \frac{\prod_{i=1}^{k} p_i}{p_s} < \frac{\prod_{i=1}^{k} p_i}{t} < 2x.$$

We now estimate k. From (2.1), we know this implies $p_k = \Theta(\log x)$. Consequently, by (2.2)

$$\frac{\varphi(m)}{m} = \frac{p_s}{p_s - 1} \prod_{\substack{2\leqslant r\leqslant p_k \\ r \text{ prime}}} \frac{r-1}{r} = \Theta\left(\frac{1}{\log p_k}\right) = \Theta\left(\frac{1}{\log\log x}\right).$$

Since the algorithm only needs to check primes up to p_k, the total time complexity is $(\log x)^{O(1)}$, which is a polynomial in $\log x$. □

3. Some Improved Toolkits

All related works involve finding an element α of large order. More precisely, the works [Hit18, Har21, HH22b] require an α with $\mathrm{ord}_N(\alpha) > N^{2/5}$, while [HH24] requires $\mathrm{ord}_N(\alpha) > N^{2/(3+2r)}$. In our work, we improve these requirements to $\mathrm{ord}_N(\alpha) > N^{1/4+o(1)}$ for the former setting, and $\mathrm{ord}_N(\alpha) > N^{1/(4r)+o(1)}$ for the latter.

First, we generalize the result of Theorem 1.1 in [HH22a] as the following Theorem.

Theorem 3.1. *Let N, s be a natural number and $m \in \mathbb{Z}_N^*$ such that $s, m < N$. Knowing s and m, one can compute finds all primes p such that $p \equiv s \pmod m, p^r | N$ in*

$$O\left(\left\lceil \frac{N^{1/4r}}{m} \right\rceil \frac{\log^{7+3\epsilon} N}{r^{2+\epsilon}}\right)$$

bit operations.

Proof. See Appendix A. □

Corollary 3.2. *Let N, s be a natural number and $m \in \mathbb{Z}_N^*$ such that $s, m < N$ and $p \equiv s \pmod{m}$ for every prime divisor p of N. Knowing s and m, one can factor N in*

$$O\left(\left\lceil \frac{N^{1/4}}{m} \right\rceil \log^{7+3\epsilon} N\right)$$

bit operations.

Proof. Set $r = 1$ in Theorem 3.1. □

Remark 3.3. When $r = 1$, Corollary 3.2 improves Theorem 3.1 in [Hit18], which obtains

$$O\left(\frac{N^{1/4}}{\sqrt{m}} \log^2 N\right)$$

when m is larger than $\log^{10} N$. Another advantage is that our algorithm only requires polynomial space.

Then we revisit the Order-finding algorithm (Lemma 4.1) in [HH24].

Lemma 3.4. *There is an algorithm taking as an input $N \geqslant 2$ and δ such that $N^{1/4r} \log^8 N \leqslant \delta \leqslant N$. It returns either some $\alpha \in \mathbb{Z}_N^*$ with $\mathrm{ord}_N(\alpha) > \delta$, or a nontrivial factor of N, or "N is r power free". Its runtime is bounded by*

$$O\left(\frac{\delta^{1/2} \log^2 N}{(\log\log\delta)^{1/2}}\right).$$

Proof. See Appendix B. □

Finally, we revisit the Order-finding algorithm in [Hit18, Har21, HH22b].

Lemma 3.5. *There is an algorithm with the following properties. It takes as input integers $N \geqslant 2$ and δ such that $N^{1/4} \log^8 N \leqslant \delta \leqslant N$. It returns either some $\alpha \in \mathbb{Z}_N^*$ with $\mathrm{ord}_N(\alpha) > \delta$, or a nontrivial factor of N, or "N is prime". Its runtime is bounded by*

$$O\left(\frac{\delta^{1/2} \log^2 N}{(\log\log\delta)^{1/2}}\right).$$

Proof. Setting $r = 1$ in Lemma 3.4. □

Remark 3.6. In [Hit18, Remark 6.4], the author conjectured that the restriction could potentially be relaxed to $\delta \geq N^{1/3+o(1)}$ for suitable $o(1)$. Through the application of Theorem 3.1, we successfully improve the lower bound of δ from $N^{2/5}$ to $N^{1/4+o(1)}$. Given that the algorithm's complexity is $\delta^{1/2+o(1)}$, this lemma remains applicable for potential future improvements in deterministic integer factorization algorithms targeting complexities of $N^{1/6+o(1)}$ or even $N^{1/8+o(1)}$.

4. Starting Point: Factoring $N = pq$

4.1. **Factoring Algorithm for Balance Semiprime.** For the convenience of the reader, we recall the following algorithm from [Har21], which forms a key subroutine of the main search algorithm presented afterwards.

Algorithm 4.1 (Finding collisions)**.**
Input:
- A positive semiprime $N = pq$.
- A positive integer κ, and an element $\alpha \in \mathbb{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) \geqslant \kappa$.

– Elements $v_1 \ldots, v_n \in \mathbb{Z}_N$ for some positive integer n, such that $v_h \neq \alpha^i$ for all $h \in \{1, \ldots, n\}$ and $i \in \{0, \ldots, \kappa - 1\}$.
– There exists $h \in \{1, \ldots, n\}$ such that $v_h \equiv \alpha^i \pmod{p}$ or $v_h \equiv \alpha^i \pmod{q}$ for some $i \in \{0, \ldots, \kappa - 1\}$

Output:

– p and q.

1: Using Lemma 2.1 (product tree), compute the polynomial

$$f(x) \coloneqq (x - v_1) \cdots (x - v_n) \in \mathbb{Z}_N[x].$$

2: Using Lemma 2.2 (Bluestein's algorithm), compute the values $f(\alpha^i) \in \mathbb{Z}_N$ for $i = 0, \ldots, \kappa - 1$.
3: **for** $i = 0, \ldots, \kappa - 1$ **do**
4: Compute $\gamma_i \coloneqq \gcd(N, f(\alpha^i))$.
5: **if** $\gamma_i \notin \{1, N\}$ **then** recover p and q and return.
6: **if** $\gamma_i = N$ **then**
7: **for** $h = 1, \ldots, n$ **do**
8: **if** $\gcd(N, v_h - \alpha^i) \neq 1$ **then** recover p and q and return.

Proposition 4.2. *Algorithm 4.1 is correct. Assuming that $\kappa, n = O(N)$, its running time is $O(n \log^3 N + \kappa \log^2 N)$.*

Proof. This is exactly Proposition 4.1 in [Har21] and Proposition 4.2 in [HH22b]. □

We now present the main search algorithm, the idea is to generalize Coppersmith's method by relaxing the determinant constraint $\det(L) < p^{md}$.

Algorithm 4.3 (The main search)**.**

Input:

– A semiprime $N = pq$ with $cN^\beta < p \leq N^\beta, 1/3 \leq \beta \leq 1/2$, where $c < 1$ is a constant.
– Positive integers $m, s \in \mathbb{Z}_N^*$ such that $72 < m < N^{(1-\beta)/2}/2, ms = tN + 1$, $X = \left\lfloor \frac{N^\beta}{m} \right\rfloor$.
– An element $\alpha \in \mathbb{Z}_N^*$ such that for all $i \in [k]$, $\gcd(\alpha^{m^2 i} - 1, N) = 1$ where

$$k \coloneqq \left\lceil \frac{2 \cdot 3^{5/4} N^{1/2}}{c m^{3/2}} \right\rceil.$$

Output: p and q.

1: **for** $i = 0, \ldots, k - 1$ **do** ▷ Computation of babysteps
2: Compute $\alpha^{m^2 i} \pmod{N}$.
3: **for** $j = 1, \ldots, m$ **do** ▷ Computation of giantsteps
4: **if** $\gcd(j, m) = 1$ **then**
5: Construct 3-rank lattice with basis

$$B = \begin{pmatrix} N & 0 & 0 \\ js & X & 0 \\ j^2 s^2 & 2jsX & X^2 \end{pmatrix} \tag{4.1}$$

6: Apply Lemma 2.4 (LLL Algorithm) and take the second vector

$$v_j = (c_j, b_j X, a_j X^2).$$

7: Compute

$$x_j = \alpha^{c_j m^2 + b_j m(1-j) + a_j (1-j)^2} \pmod N$$

8: Applying a sort-and-match algorithm to the babysteps and giantsteps computed in Steps 2 and 7, find all pairs (i, j) such that

$$\alpha^{m^2 i} \equiv x_j \pmod N. \tag{4.2}$$

For each such match, solve the quadratic equation

$$c_j + b_j \frac{(p-j)}{m} + a_j \frac{(p-j)^2}{m^2} = ip$$

If p is found, return p and N/p.

9: Let $x_1, \ldots, x_n$ be the list of giantsteps computed in Step 7, skipping those that were discovered in Step 8 to be equal to one of the babysteps. Apply Algorithm 4.1 (finding collisions) with N, $\kappa \coloneqq k$, $\alpha \coloneqq \alpha^{m^2} \pmod N$ and $x_1, \ldots, x_n$ as inputs.

10: Return p and q.

Before we prove the correctness of Algorithm 4.3, we would like to explain the core step of the main search.

The main idea is to use the Coppersmith method to compute the giantsteps, which is different with [HH22b]. We also gain the loglog speed up by sieving on small primes. In common Coppersmith method, suppose one knows x, m such that $p \equiv x \bmod m$, if $\log_N(p/m) \leq (\log_N p)^2$ then one could factor N in polynomial time. We use the same lattice in dimension 3. While the classical Coppersmith method requires strict adherence to the Howgrave-Graham lemma's inequality, we present a generalization that relaxes these constraints. Specifically, we remove the strict requirement that $\det < p^d$. Although the vectors obtained from our modified LLL approach may not be as short, they can be utilized to construct giant steps. We then employ Harvey's baby-step giant-step method [Har21] to find collisions and factor N.

The reason why we choose the second vector by LLL algorithm, is that the shortest vector is the second row of the basis itself. The collision by this shortest vector is trivial. We prove this in the following lemma.

Lemma 4.4. *Let $N, m \in \mathbb{N}, ms = tN + 1, j \in \mathbb{Z}_m^*$, $\tilde{p}_j = js$, $X = \lfloor N^\beta / m \rfloor$, $1/3 \leqslant \beta \leqslant 1/2, 72 < m < N^{(1-\beta)/2}/2$. Then the shortest vector in the lattice*

$$B = \begin{pmatrix} N & 0 & 0 \\ \tilde{p}_j & X & 0 \\ \tilde{p}_j^2 & 2\tilde{p}_j X & X^2 \end{pmatrix}$$

is

$$v_0 = -tj(N, 0, 0) + m(js, X, 0) = (j, mX, 0),$$

and the second vector v_2 obtained from the LLL algorithm has a non-zero third coordinate.

Proof. We first demonstrate that v_0 is indeed the shortest vector in the lattice. Suppose, for the sake of contradiction, that there exists a shorter vector. Let this vector be represented as

$$v = (aN + bjs, (b + cjs)X, cX^2).$$

By our assumption, the squared norm of v must satisfy

$$\|v\|^2 = (aN + bjs)^2 + (b + cjs)^2X^2 + c^2X^4 \leqslant \|v_0\|^2 = j^2 + m^2X^2.$$

Consider the expression

$$\begin{aligned}
|(am^2 + bjtm - cj^2t)N|^2 &= \left(m^2(aN + bjs) - jm(b + cjs) + cj^2\right)^2 \\
&\leqslant \left((aN + bjs)^2 + (b + cjs)^2X^2 + c^2X^4\right)\left(m^4 + \frac{j^2m^2}{X^2} + \frac{j^4}{X^4}\right) \\
&< (j^2 + m^2X^2)\left(m^4 + \frac{m^4}{X^2} + \frac{m^4}{X^4}\right) \\
&< 2N^{2\beta} \cdot 2m^4 < N^2,
\end{aligned}$$

which implies that $am^2 + bjtm - cj^2t = 0$. Since $\gcd(j, m) = 1$ and $\gcd(t, m) = 1$, we must have $m \mid c$. However, we know that $|c| < 2N^\beta/X^2 < m$, which forces $c = 0$.

Substituting this back into our equation yields $am + bjt = 0$. Again, using the facts that $\gcd(j, m) = 1$ and $\gcd(t, m) = 1$, there must exist an integer k such that $b = km$ and $a = -ktj$. This means that v is an integer multiple of v_0, contradicting our assumption that v is the shortest vector distinct from v_0.

We now prove that the second vector v_2 obtained from the LLL algorithm has a non-zero third coordinate. By Theorem 2.3 and the fact that $\lambda_3 \geqslant \lambda_2 \geqslant \lambda_1 > N^\beta$, we have

$$\lambda_1\lambda_2\lambda_3 \leqslant (\sqrt{3})^3 \det B \Rightarrow \lambda_2 \leqslant \left(\frac{3^{3/2}\det B}{\lambda_1}\right)^{1/2} < \frac{3^{3/4}N^{1/2+\beta}}{m^{3/2}}. \tag{4.3}$$

From the properties of the LLL algorithm (Lemma 2.4), we know that

$$\|v_2\| \leqslant 2\lambda_2 < \frac{2 \cdot 3^{3/4}N^{1/2+\beta}}{m^{3/2}}.$$

If the third coordinate of v_2 were zero, then using the same analysis as above, we could write $v_2 = (aN + bjs, bX, 0)$. This would give us

$$(aN + bjs)^2 + b^2X^2 = \|v_2\|^2 < \frac{4 \cdot 3^{3/2}N^{1+2\beta}}{m^3}.$$

Similarly,

$$\begin{aligned}
|(am + bjt)N|^2 &= \left(m(aN + bjs) - bj\right)^2 \\
&\leqslant \left((aN + bjs)^2 + b^2X^2\right)\left(m^2 + \frac{j^2}{X^2}\right) \\
&< \frac{4 \cdot 3^{3/2}N^{1+2\beta}}{m^3} \cdot 2m^2 \\
&= \frac{8 \cdot 3^{3/2}N^{1+2\beta}}{m} < N^2,
\end{aligned}$$

which implies $am + bjt = 0$. Since $\gcd(j, m) = 1$ and $\gcd(t, m) = 1$, we must have $b = km$ and $a = -ktj$ for some integer k. But this would make v_2 an integer multiple of the shortest vector v_0, contradicting the linear independence of the basis vectors. □

Now we will prove the following proposition of our main search.

Proposition 4.5. *Algorithm 4.3 is correct. It runs in time*

$$O\left(\phi(m)\log^3 N + \frac{N^{1/2}\log^2 N}{m^{3/2}}\right). \tag{4.4}$$

Proof. We first prove correctness. Let $p = mp_m + p_l$. Then for $j = p_l$, we have $p|js + p_m$ and $|p_m| < \lfloor p/m \rfloor \leqslant X$. Consider the vector $v_j = (c_j, b_j X, a_j X^2)$ obtained in Step 6. We know that

$$p|c_j + b_j p_m + a_j p_m^2$$

Let $c_j + b_j p_m + a_j p_m^2 = ip$. By the Cauchy-Schwarz inequality,

$$i^2p^2 = (c_j + b_j p_m + a_j p_m^2)^2 \leqslant 3(c_j^2 + b_j^2 p_m^2 + a_j^2 p_m^4) \leqslant 3\|v_j\|^2 \tag{4.5}$$

From (4.3), we know

$$\|v_j\| \leqslant 2\lambda_2 < \frac{2\cdot 3^{3/4}N^{1/2+\beta}}{m^{3/2}} \tag{4.6}$$

Combining (4.5) and (4.6), we get

$$i < \frac{2\cdot 3^{5/4}N^{1/2+\beta}}{m^{3/2}p} < \frac{2\cdot 3^{5/4}N^{1/2}}{cm^{3/2}} \leqslant k$$

Furthermore, we know

$$mp_m = p - p_l \equiv 1 - j \pmod{p-1}$$

Therefore,

$$\alpha^{im^2} \equiv \alpha^{ipm^2} \equiv \alpha^{m^2(c_j + b_j p_m + a_j p_m^2)} \equiv \alpha^{c_j m^2 + b_j m(1-j) + a_j(1-j)^2} = x_j \pmod{p}$$

So there must be a collision modulo p between the baby steps computed in Step 2 and the giant steps computed in Step 7.

If this collision is not only a collision mod$p$ but also a collision mod$N$, then Step 8 will find the pair (i, j) and solve the quadratic equation. By Lemma 4.4, we know that $a_j \neq 0$, so we are guaranteed to return p by solving the quadratic equation.

Otherwise, if this collision is merely a collision mod$p$, then the algorithm will find p in Step 9.

Now we analyse the time complexity. It is clear that the number of babysteps is k. So the total cost of computing babysteps (Step 1–2) is

$$O(k\log N(\log\log N)^2).$$

In Step 4 we compute m GCDs of integers bounded by $m = O(N)$, which costs

$$O(m\log N(\log\log N)^2).$$

Then consider the gaintsteps computation. It is also clear that the number of gaintsteps is $\phi(m)$. By Lemma 2.4, we know the LLL algorithm on our 3-dimensional lattice is $O(\log^{2+\epsilon} N)$. So the cost of Step 5–6 is

$$O(\phi(m)\log^{2+\epsilon} N)$$

The Step 7 can be computed in time

$$O(\phi(m)\log N\log\log N)$$

In Step 8, we construct a list of pairs $(\alpha^{m^2 i}, i)$ of length k, and a list of tuples (x_j, j) of length $O(\phi(m))$. From the bounds already mentioned, each item in these

lists occupies $O(\lg N)$ bits. We then use merge-sort to sort the lists by the first component of each tuple, which requires

$$O((k+\phi(m))\log^2 N)$$

bit operations. Each giantstep x_j is equal to at most one babystep $\alpha^{m^2 i}$, because our input $gcd(\alpha^{m^2 i}-1, N) = 1, \forall i \in [k]$ implies that the babysteps are all distinct. Matching the two sorted lists, we may hence find all matches in time

$$O((\kappa+\phi(m))\log N).$$

Since there are at most k such matches, the total cost for solving the quadratic equation in Step 8 is bounded by

$$O(k \log N \log\log N).$$

Finally, in Step 9 we apply Algorithm 4.1, whose complexity is

$$O(\phi(m)\log^3 N + k\lg^2 N) = O\left(\phi(m)\log^3 N + \frac{N^{1/2}\log^2 N}{m^{3/2}}\right).$$

Combining the cost time from all steps, we obtain the overall bound

$$O\left(\phi(m)\log^3 N + \frac{N^{1/2}\log^2 N}{m^{3/2}}\right).$$

□

Then we present an algorithm to find the element $\alpha \in \mathbb{Z}_N^*$ for Algorithm 4.3.

Algorithm 4.6 (Finding α)**.**
Input:
- A semiprime $N = pq$.
- A positive integer $k, m = O(N)$, where $m = \prod_{p_j \in S} p_j^{\alpha_j}$ and S is a known set of distinct primes.

Output:
- An element $\alpha \in \mathbb{Z}_N^*$ such that for all $i \in [k]$, $\gcd(\alpha^{m^2 i}-1, N) = 1$.
- Otherwise, the prime factors p and q.

1: Apply Lemma 3.5 with $D \coloneqq \lceil N^{1/3} \rceil$. If any factors of N are found, return. Otherwise, we obtain $\alpha \in \mathbb{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) > D$.
2: **for** $i = 0, \ldots, k-1$ **do**
3: Compute $\gcd(\alpha^{m^2 i}-1, N)$.
4: **if** $\gcd(N, \alpha^{m^2 i}-1) \notin \{1, N\}$ **then**
5: Recover p and q and return.
6: **if** $\gcd(N, \alpha^{m^2 i}-1) = N$ **then**
7: Initialize $r = m^2 i$ and factor $m^2 = \prod_{p_j \in S} p_j^{2\alpha_j}$.
8: **for** each prime $p_j \in S$ **do**
9: **while** $p_j | r$ and $\gcd(\alpha^{r/p_j}-1, N) = N$ **do**
10: $r \leftarrow r/p_j$
11: **for** each prime $p_j \in S$ where $p_j | r$ **do**
12: Compute $\gcd(\alpha^{r/p_j}-1, N)$.
13: **if** $\gcd(\alpha^{r/p_j}-1, N) \notin \{1, N\}$ **then**
14: Recover p and q and return.

15: Using Corollary 3.2 with $p \equiv 1 \pmod{r}$ to factor N and return p and q.

16: Return α.

Proposition 4.7. *Algorithm 4.6 is correct. It runs in time*

$$O\left(\frac{N^{1/6}\log^2 N}{(\log\log N)^{1/2}} + (k+\log m)\log N(\log\log N)^2\right)$$

Proof. We first prove correctness. Assume that the element β we obtain in Step 1 satisfies $\gcd(\alpha^{m^2 i}-1, N) = 1$ for all $i \in [k]$. Then the algorithm will return β in Step 16.

Otherwise, there exists some $i \in [k]$ such that $\gcd(\alpha^{m^2 i}-1, N) \neq 1$. Let i_0 be the smallest $i \in [k]$ such that $\gcd(\alpha^{m^2 i}-1, N) \neq 1$. If $\gcd(\alpha^{m^2 i_0}-1, N) \in \{p, q\}$, then the algorithm will return p and q in Step 5. The only remaining case is that $\gcd(\alpha^{m^2 i_0}-1, N) = N$.

Let $\mathrm{ord}_p(\beta) = r_p$ and $\mathrm{ord}_q(\beta) = r_q$. We first claim that $i_0 | r_p$. We prove this by contradiction. Suppose $\gcd(i_0, r_p) < i_0$. Since

$$r_p | i_0 m^2 \Longrightarrow \frac{r_p}{\gcd(i_0, r_p)} \Big| \frac{i_0}{\gcd(i_0, r_p)} m^2$$

Because $\gcd(r_p/\gcd(i_0, r_p), i_0/\gcd(i_0, r_p)) = 1$, we know that $(r_p/\gcd(i_0, r_p)) | m^2$. Thus

$$r_p = \gcd(i_0, r_p) \cdot \frac{r_p}{\gcd(i_0, r_p)} \Big| \gcd(i_0, r_p) m^2$$

This implies that $p | \gcd(\alpha^{\gcd(i_0, r_p) m^2}-1, N)$, which contradicts the minimality of i_0! By similar reasoning, we can show that $i_0 | r_q$.

At the beginning of Step 6, we initialize $r = m^2 i_0$. The algorithm then performs a series of divisions by the prime factors of m^2. For each prime factor p_j, we divide r by p_j as long as $\gcd(\alpha^{r/p_j}-1, N) = N$. This process continues until we find the smallest value r such that $\alpha^r \equiv 1 \pmod{N}$.

We claim that after this process, r equals $\mathrm{ord}_N(\alpha)$. This is because we start with a multiple of both r_p and r_q and gradually reduce it by removing unnecessary prime factors until we reach the minimum value that still satisfies $\alpha^r \equiv 1 \pmod{N}$.

Now, two cases are possible:

Case 1: $r_p \neq r_q$. Without loss of generality, there must exist a prime factor p_j of r_q such that $p_j \nmid r_p$ or p_j appears with a higher exponent in r_q than in r_p. When we compute $\gcd(\alpha^{r/p_j}-1, N)$, we have $\alpha^{r/p_j} \equiv 1 \pmod{p}$ (since r/p_j is still a multiple of r_p) and $\alpha^{r/p_j} \not\equiv 1 \pmod{q}$ (since r/p_j is no longer a multiple of r_q). Therefore, $\gcd(\alpha^{r/p_j}-1, N) = p$, and the algorithm will return p and q in Step 14.

Case 2: $r_p = r_q = r$. In this case, we have $r = \mathrm{ord}_p(\alpha) = \mathrm{ord}_q(\alpha) = \mathrm{ord}_N(\alpha)$. Since $r > D > N^{1/4+o(1)}$ (from Step 1), we can apply Coppersmith's method as implemented in Corollary 3.2 to factor N in polynomial time, and return p and q in Step 15.

This completes the proof of correctness.

We now analyze time complexity. The cost of Step 1 from Lemma 3.5 is

$$O\left(\frac{N^{1/6}\log^2 N}{(\log\log N)^{1/2}}\right).$$

To prepare for the loop in Step 5, we first compute $\alpha^{m^2} \pmod N$; the cost of this step is

$$O(\log m \log N \log\log N).$$

The Step 5 itself computes at most k products modulo N and k GCDs of integers bounded by N, whose total cost is

$$O(k \log N (\log\log N)^2).$$

If we go into Step 10, for each prime $p_j \in S$, we compute at most $2\alpha_j$ products modulo N and $2\alpha_j$ GCDs of integers bounded by N. Since $\sum \alpha_j$ is at most $\log m$, and the time cost of Coppersmith's method is in polynomial time, so the total cost of Steps 10–15 is at most

$$O(\log m \log N (\log\log N)^2 + \mathrm{poly}(\log N)).$$

Combining together we complete the proof. □

Finally we present the main factoring algorithm. In this algorithm, N_0 is a constant that is chosen large enough to ensure that the proof of correctness works for all $N \geqslant N_0$.

Algorithm 4.8 (Factoring semiprimes)**.**
Input: A semiprime $N = pq \geq N_0$ with $cN^\beta < p \leq N^\beta, 1/3 \leq \beta \leq 1/2$, where $c < 1$ is a constant. *Output:* p and q.

1: Compute

$$x \coloneqq \left\lfloor \frac{N^{1/5}(\log\log N)^{2/5}}{\log^{2/5} N} \right\rfloor,$$

2: Apply Lemma 2.10 with x and get m satisfying:

$$\frac{x}{2} < m < 2x, \quad \frac{\varphi(m)}{m} = \Theta\left(\frac{1}{\log\log N}\right).$$

If $\gcd(N, m) \notin \{1, N\}$, recover p and q and return. Computing $s = m^{-1} \pmod N$ and

$$k \coloneqq \left\lceil \frac{2 \cdot 3^{5/4} N^{1/2}}{cm^{3/2}} \right\rceil = \Theta\left(\frac{N^{1/5} \log^{3/5} N}{(\log\log N)^{3/5}}\right).$$

3: Apply Algorithm 4.6 with N, k, m. If any factors of N are found, return. Otherwise, we obtain $\alpha \in \mathbb{Z}_N^*$ such that for all $i \in [k]$, $\gcd(\alpha^{m^2 i} - 1, N) = 1$.
4: Run Algorithm 4.3 (the main search) with the given N, m, s and α. Return p and q.

Proposition 4.9. *Algorithm 4.8 is correct (for suitable N_0), and it runs in time*

$$O\left(\frac{N^{1/5} \log^{13/5} N}{(\log\log N)^{3/5}}\right).$$

Proof. We first prove the correctness. Consider Step 2, we either get p and q or m, s satisfying

$$m = \Theta(x) = \Theta\left(\frac{N^{1/5}(\log\log N)^{2/5}}{\log^{2/5} N}\right).$$

In Step 3, we either get p and q or obtain $\alpha \in \mathbb{Z}_N^*$ such that for all $i \in [k]$, $\gcd(\alpha^{m^2 i} - 1, N) = 1$. Also, for large enough N, we have

$$72 < m < \frac{N^{1/4}}{2} \leqslant \frac{N^{(1-\beta)/2}}{2},$$

which satisfies the requirement of Algorithm 4.3. In Step 4, we are guaranteed to get p and q by the Proposition 4.5.

Then we calculate the time complexity. Step 1 and Step 2 (from Lemma 2.10) are only polynomial time, which is negligible.

From Proposition 4.7, Step 3 costs

$$O\left(\frac{N^{1/6}\log^2 N}{(\log\log N)^{1/2}} + (k + \log m)\log N(\log\log N)^2\right),$$

is also negligible.

From Proposition 4.5, Step 4 costs

$$O\left(\phi(m)\log^3 N + \frac{N^{1/2}\log^2 N}{m^{3/2}}\right) = O\left(\frac{N^{1/5}\log^{13/5} N}{(\log\log N)^{3/5}}\right).$$

□

Corollary 4.10. *Applying Algorithm 4.8 with $\beta = 1/2$, we prove Theorem 1.1.*

5. Better Factoring Sums and Differences of Powers

5.1. **Factoring Algorithm for Extra Modulo Information.** In the the previous section, we showed how the bit size of p can help us to improve factoring. Now we focus on how to exploit extra modulo information.

We employ the same fundamental approach as in Algorithm 4.3, with the primary distinction being the necessity to enumerate the possible bit sizes of p, resulting in a factor of $\log N$ increase in the number of giantsteps. Concurrently, the congruence constraint $p \equiv r \pmod n$ reduces the number of giantsteps by a factor of n.

Algorithm 5.1 (The main search with modulo)**.**
Input:

- A semiprime $N = pq, N^{1/3} < p < N^{1/2}$, r, n such that $p \equiv r \pmod n, (n, N) = 1$.
- Positive integers $m \in \mathbb{Z}_N^*$, such that $(m, n) = 1, 72 < mn < N^{1/4}/2, s = (mn)^{-1} \pmod N, m_{inv} = m^{-1} \pmod n, n_{inv} = n^{-1} \pmod m$.
- An element $\alpha \in \mathbb{Z}_N^*$ such that for all $i \in [k]$, $\gcd(\alpha^{(mn)^2 i} - 1, N) = 1$ where

$$k \coloneqq \left\lceil \frac{4 \cdot 3^{5/4} N^{1/2}}{(mn)^{3/2}} \right\rceil.$$

Output: p and q.

1: **for** $i = 0, \ldots, k-1$ **do** ▷ Computation of babysteps
2: Compute $\alpha^{(mn)^2 i} \pmod N$.
3: **for** $i = 1, \ldots, \left\lceil \frac{\log N}{6} \right\rceil$ **do** ▷ Computation of giantsteps
4: Compute $X_i = \left\lfloor \frac{2^i N^{0.3}}{mn} \right\rfloor$.
5: **for** $j = 1, \ldots, m$ **do**
6: **if** $\gcd(j, m) = 1$ **then**
7: Compute $m_j \equiv snm_{inv} + jmn_{inv} \pmod{mn}$.

8: Construct 3-rank lattice with basis

$$B = \begin{pmatrix} N & 0 & 0 \\ m_j s & X_i & 0 \\ m_j^2 s^2 & 2m_j s X_i & X_i^2 \end{pmatrix}. \tag{5.1}$$

9: Apply Lemma 2.4 (LLL Algorithm) and take the second vector

$$v_{ij} = (c_{ij}, b_{ij} X_i, a_{ij} X_i^2).$$

10: Compute

$$x_{ij} = \alpha^{c_{ij}(mn)^2 + b_{ij} mn(1-j) + a_{ij}(1-j)^2} \pmod N.$$

11: Applying a sort-and-match algorithm to the babysteps and giantsteps computed in Steps 2 and 10, find all pairs (i, j, σ) such that

$$\alpha^{(mn)^2 \sigma} \equiv x_{ij} \pmod N. \tag{5.2}$$

For each such match, solve the quadratic equation

$$c_{ij} + b_{ij} \frac{(p - m_j)}{mn} + a_{ij} \frac{(p - m_j)^2}{(mn)^2} = \sigma p.$$

If p is found, return p and N/p.

12: Let $\{x_{ij}\}$ be the list of giantsteps computed in Step 10, skipping those that were discovered in Step 11 to be equal to one of the babysteps. Apply Algorithm 4.1 (finding collisions) with N, $\kappa \coloneqq k$, $\alpha \coloneqq \alpha^{(mn)^2} \pmod N$ and $\{x_{ij}\}$ as inputs.

13: Return p and q.

Proposition 5.2. *Algorithm 5.1 is correct. It runs in time*

$$O\left(\phi(m) \log^4 N + \frac{N^{1/2} \log^2 N}{(mn)^{3/2}}\right). \tag{5.3}$$

Proof. The proof follows a similar structure to that of Proposition 4.5. We first establish the correctness of the algorithm.

Let $p = mnp_m + p_l$ where $0 < p_l < mn$. For $j_0 = (p \bmod m)$, the Chinese remainder theorem implies that $p_l = m_{j_0}$, which gives us $p | m_{j_0} s + p_m$. Setting $t = \lceil \log N/6 \rceil$, we observe that $X_0 \leqslant |p_m| < \lfloor p/(mn) \rfloor \leqslant \lfloor N^{0.5}/(mn) \rfloor = X_t$. Therefore, there exists an index i_0 such that $X_{i_0 - 1} \leqslant |p_m| \leqslant X_{i_0}$, which implies that $p \geqslant mnX_{i_0 - 1} = mnX_{i_0}/2$.

Consider the vector $v_{i_0 j_0} = (c_{i_0 j_0}, b_{i_0 j_0} X_{i_0}, a_{i_0 j_0} X_{i_0}^2)$ obtained in Step 6. We know that

$$p | c_{i_0 j_0} + b_{i_0 j_0} p_m + a_{i_0 j_0} p_m^2.$$

Let $c_{i_0 j_0} + b_{i_0 j_0} p_m + a_{i_0 j_0} p_m^2 = \sigma p$. By the Cauchy-Schwarz inequality, we have

$$\sigma^2 p^2 = (c_{i_0 j_0} + b_{i_0 j_0} p_m + a_{i_0 j_0} p_m^2)^2 \leqslant 3 \|v_{i_0 j_0}\|^2. \tag{5.4}$$

Working with modulo mn, we note that for $i = 0, \ldots, t$, we have $\beta_i = \log_N(X_i mn) \in [1/3, 1/2]$ and $mn < N^{1/4}/2 \leqslant N^{(1-\beta_i)/2}/2$, which satisfies the conditions of Lemma 4.4.

From equation (4.3), we derive

$$\|v_{i_0 j_0}\| \leqslant 2\lambda_2 < \frac{2 \cdot 3^{3/4} N^{1/2 + \beta_{i_0}}}{(mn)^{3/2}}. \tag{5.5}$$

Combining inequalities (5.4) and (5.5), we obtain

$$\sigma < \frac{2 \cdot 3^{5/4} N^{1/2+\beta_{i_0}}}{(mn)^{3/2} p} \leqslant \frac{4 \cdot 3^{5/4} N^{1/2}}{(mn)^{3/2}} \leqslant k.$$

Following the same reasoning as in Proposition 4.5, we have $\alpha^{\sigma(mn)^2} \equiv x_{i_0 j_0} \pmod{p}$. So there must exist a collision modulo p between the babysteps and the giantsteps.

If this collision also manifests as a collision modulo N, then Step 11 will identify the triplet (i_0, j_0, σ) and solve the corresponding quadratic equation. By Lemma 4.4, we know that $a_{i_0 j_0} \neq 0$, ensuring that we can successfully recover p by solving the quadratic equation.

Alternatively, if the collision only occurs modulo p but not modulo N, the algorithm will still determine p in Step 12.

We now analyze the time complexity of the algorithm. The number of babysteps is k, and the number of giantsteps is $t\phi(m)$. Therefore, the computational cost of finding collisions between babysteps and giantsteps is

$$O(k \log^2 N + t\phi(m) \log^3 N) = O\left(\phi(m) \log^4 N + \frac{N^{1/2} \log^2 N}{(mn)^{3/2}}\right).$$

The time complexity of the remaining steps is negligible compared to this dominant term, analogous to the analysis in Proposition 4.5, which completes our proof. □

Now we revisit Theorem 2.8 in [Hit17] with our three dimensional-lattice technique.

Theorem 5.3. *Let N, r, n be a natural number such that $r = p \mod n$ for every prime divisor p of N. Knowing r and n, one can compute the prime factorization of N in*

$$O\left(\frac{N^{1/5} \log^{16/5} N}{n^{3/5}} + N^{1/6} \log^4 N\right).$$

Proof. We first note that if N is prime, we can identify this using the AKS algorithm in polynomial time.

If N has three or more prime factors (counting repetitions), then at least one factor is bounded above by $N^{1/3}$, and such a factor may be found in time $O(N^{1/6} \log^3 N)$ (see [Har21, Prop. 2.5]). By repeating this process at most $\log N$ times, we reduce to the case where N is a semiprime with $p, q > N^{1/3}$. The cost of this reduction step is at most

$$O\left(N^{1/6} \log^4 N\right).$$

Now we assume $n < N^{1/4}$; otherwise, we could apply Corollary 3.2 to factor N in polynomial time.

Next, we determine the parameter m. We begin by computing $m = \left\lceil \frac{N^{1/5}}{\log^{4/5} N \cdot n^{3/5}} \right\rceil$. If $\gcd(m, n) > 1$, we set $m = m + 1$ and iterate until we find a value such that $\gcd(m, n) = 1$. We claim that this process examines at most $O(\log^2 n)$ values of m. This follows from a result by Iwaniec [Iwa78], which states that $j(n) = O(\log^2 n)$, where $j(n)$ is defined as the smallest positive integer y such that

every sequence of y consecutive integers contains an integer coprime to n. Since $\log^2 n < \log^2 N \ll \frac{N^{1/5}}{\log^{4/5} N \cdot n^{3/5}}$, the value of m we obtain satisfies

$$m = \Theta\left(\frac{N^{1/5}}{\log^{4/5} N \cdot n^{3/5}}\right), \quad \gcd(m, n) = 1,$$

and the computational complexity of this step is only polynomial in $\log N$, which is negligible in the overall complexity analysis.

We proceed to compute $s = (mn)^{-1} \pmod N$, $m_{inv} = m^{-1} \pmod n$, $n_{inv} = n^{-1} \pmod m$, and the factorization of mn. These computations can be performed in polynomial time, and the factorization costs at most $(mn)^{1/4} = O((N^{3/10})^{1/4}) = O(N^{3/40})$, which is also negligible.

Let

$$k \coloneqq \left\lceil \frac{4 \cdot 3^{5/4} N^{1/2}}{(mn)^{3/2}} \right\rceil = \Theta\left(\frac{N^{1/5} \log^{6/5} N}{n^{3/5}}\right).$$

We apply Algorithm 4.6 with parameters N, k, and mn, which either yields p and q directly or provides an element $\alpha \in \mathbb{Z}_N^*$ such that for all $i \in [k]$, $\gcd(\alpha^{m^2 i} - 1, N) = 1$. This step incurs a cost of

$$O\left(\frac{N^{1/6} \log^2 N}{(\log \log N)^{1/2}} + (k + \log(mn)) \log N (\log \log N)^2\right),$$

which is negligible in the overall complexity.

Finally, we employ Algorithm 5.1 with the aforementioned parameters to factor the semiprime N. We can verify that all conditions required by Algorithm 5.1 are satisfied. The complexity of this algorithm is

$$O\left(\phi(m) \log^4 N + \frac{N^{1/2} \log^2 N}{(mn)^{3/2}}\right) = O\left(\frac{N^{1/5} \log^{16/5} N}{n^{3/5}}\right).$$

Combining all the above analyses, the total complexity of the algorithm is

$$O\left(\frac{N^{1/5} \log^{16/5} N}{n^{3/5}} + N^{1/6} \log^4 N\right).$$

□

Remark 5.4. When n is not a product of small primes, by selecting an appropriate m as a product of small primes, we can obtain the same loglog speedup as Algorithm 4.8, yielding an improved complexity of $O\left(\frac{N^{1/5} \log^{16/5} N}{(n \log \log N)^{3/5}}\right)$ in the first term of Theorem 5.3.

5.2. **Factoring Sums and Differences of Powers.** While maintaining the original algorithmic framework proposed by Hittmeir (Algorithm 3.1 in [Hit17]), we present an improved version of his Theorem 2.8 as Theorem 5.3 for $n = \Theta(\log N)$, which yields better bounds in the final results.

Algorithm 5.5 (Factoring sums and differences of powers)**.**
Input:
- Coprime integers $a, b \in \mathbb{N}$ with $a > b$.
- A number $N \in P_{a,b}$ where either $N = a^n + b^n \in P_{a,b}^+$ or $N = a^n - b^n \in P_{a,b}^-$ for some $n \in \mathbb{N}$.

Output:

– The prime factorization of N.

Set $N_1 := N, v := 1$.
Apply trial division to compute all divisors of n.
if $N \in P^+_{a,b}$ **then**
    Define $\mathcal{D} := \{2d : d \mid n\}$.
else
    Define $\mathcal{D} := \{d : d \mid n\}$.
Let $d_1 < d_2 < \cdots < d_l$ be the ordered list of all elements in $\mathcal{D}$ where $l \geqslant 2$.
while $N_j \neq 1$ **do**
    Set $j = v$.
    Compute $G_j = \gcd((ab^{-1})^{d_j} - 1 \bmod N_j, N_j)$.
    if $G_j = 1$ **then**
        Set $N_{j+1} = N_j$.
    else if $1 < G_j \leqslant N$ **then**
        Apply Theorem 5.3 with $n = d_j$ and $r = 1$ to compute prime factorization of G_j.
        Remove all prime factors dividing G_j from N_j to obtain N_{j+1}.
    Set $v = j + 1$.
Return the prime factorization of N.

Proposition 5.6. *Algorithm 5.5 is correct and it runs in time*

$$O\left(N^{1/5}\log^{13/5} N\right).$$

Proof. The correctness of Algorithm 5.5 has already been proved in proof of Theorem 1.1 [Hit17]. We now focus on the time complexity.

Following the analysis in [Hit17], we have $n = O(\log N)$, and for $j < l$, we obtain $G_j \leqslant N^{1/2}$ when $N \in P^-_{a,b}$ and $G_j \leqslant N^{2/3}$ when $N \in P^+_{a,b}$. For $j = l$, note that $n = \Theta(\log N)$. In the worst case where $G_l = N$, applying Theorem 5.3 yields a runtime of

$$O\left(\frac{N^{1/5}\log^{16/5} N}{n^{3/5}} + N^{1/6}\log^4 N\right) = O(N^{1/5}\log^{13/5} N),$$

which dominates all other operations and gives the claimed complexity bound. This also proves Theorem 1.2. □

Remark 5.7. For the case where $j = l$, we have $n = \Theta(\log N)$. In this scenario, one can achieve a logarithmic speedup by carefully selecting small prime numbers that are coprime to n, thus obtaining a modulu m that satisfies both $\gcd(m, n) = 1$ and $\varphi(m)/m = \Theta(1/\log\log N)$. This leads to the complexity bound $O\left(\frac{N^{1/5}\log^{13/5} N}{(\log\log N)^{3/5}}\right)$.

Notably, this matches the complexity in Theorem 1.1 for balanced semiprimes, where the bit-length information of p saves a factor of $\log N$ in the exhaustive search, while here the additional information from $n = \Theta(\log N)$ combined with the loglog speedup achieves the same effect.

6. Speedup for $p^r q$

Finally, we generalize our rank-3 lattice construction to solve the r-power divisor problem. This problem—finding all integers p such that $p^r \mid N$—has recently seen notable progress in the development of provably deterministic algorithms. Hales

and Hiary [HH24] extended Lehman's method [Leh74] and obtained two algorithms with complexities

$$O\left(N^{1/(r+2)}(\log N)^2 \log\log N\right) \quad \text{and} \quad O\left(N^{1/(3+2r)}(\log N)^{16/5}\right).$$

Around the same time, Harvey and Hittmeir [HH22a] (Proceedings of ANTS XV, *Res. Number Theory* 8 (2022), no. 4, Paper No. 94) applied Coppersmith's method directly and achieved a complexity of

$$O\left(\frac{N^{1/4r}\log^{10+\epsilon} N}{r^3}\right).$$

By incorporating faster LLL-type lattice reduction algorithms and sieving on small primes, we improve this to

$$O\left(\frac{N^{1/4r}\log^{7+3\epsilon} N}{(\log\log N - \log 4r)r^{2+\epsilon}}\right).$$

According to Remark 3.5 in [HH22a], the worst-case running time of their algorithm occurs when $p = \Theta(N^{1/2r})$ and $q = \Theta(N^{1/2})$, that is, when $N = p^r q$. By focusing on this case and employing our rank-3 lattice construction, we further reduce the complexity to

$$O\left(\sqrt{r}N^{1/4r}\log^{5/2} N\right).$$

6.1. **More Refined Complexity Analysis of** [HH22a]**.** Set $m = 1$ in Theorem 3.1, we first demonstrate how the complexity bound in [HH22a] can be improved from

$$O\left(\frac{N^{1/4r}\log^{10+\epsilon} N}{r^3}\right) \quad \text{to} \quad O\left(\frac{N^{1/4r}\log^{7+3\epsilon} N}{r^{2+\epsilon}}\right).$$

Theorem 6.1. *There exists an explicit deterministic algorithm with the following properties. Given as input an integer $N > 2$ and a positive integer r, the algorithm outputs a list of all positive integers p such that $p^r \mid N$. Its running time is*

$$O\left(\frac{N^{1/4r}\log^{7+3\epsilon} N}{r^{2+\epsilon}}\right).$$

Proof. Set $s = 0$ and $m = 1$ in Theorem 3.1. □

Remark 6.2. We emphasize that this improvement does not arise from a novel algorithmic idea, but rather from a more refined complexity analysis combined with the use of alternative lattice basis reduction algorithms [NS16].

We could also get a faster algorithm using the same idea of sieving on the small primes.

Theorem 6.3. *Let N be a natural number, one can compute all primes p such that $p^r|N$ in time*

$$F(N) = O\left(\frac{N^{1/4r}}{\log\log N - \log 4r}\frac{\log^{7+3\epsilon} N}{r^{2+\epsilon}}\right).$$

Proof. Set $x = \lceil N^{1/4r} \rceil$ and apply Lemma 2.10 to obtain m satisfying

$$m = \prod_{p\in S} p, \quad \frac{x}{2} < m < 2x, \quad \frac{\varphi(m)}{m} = \Theta\left(\frac{1}{\log\log x}\right)$$

This step requires only polynomial time in $\log N$.

We first compute $G = \gcd(m, N)$, then identify all primes $p \in S$ such that $p|G$, and verify for each whether $p^r|N$. We remove all prime factors p of G from N to obtain N_1.

Since $|S| = O(\log N)$, this step also requires only polynomial time in $\log N$.

Next, we iterate through all elements $i \in \mathbb{Z}_m^*$. For each i, we apply Theorem 3.1 with parameters $N_1, r, s = i$, and m to find all primes p satisfying $p \equiv i \pmod m$ and $p^r|N_1$. We claim this will identify all primes p such that $p^r|N_1$. This is because if a prime p divides N_1, then $\gcd(p, m) = 1$, which implies $(p \bmod m) \in \mathbb{Z}_m^*$. Therefore, when $i = (p \bmod m)$, this prime p will be output by Theorem 3.1. Combined with our earlier discussion, we prove that we output all primes p such that $p^r|N$.

The time complexity for each i is

$$O\left(\left\lceil \frac{N_1^{1/4r}}{m} \right\rceil \frac{\log^{7+3\epsilon} N}{r^{2+\epsilon}}\right) = O\left(\frac{\log^{7+3\epsilon} N}{r^{2+\epsilon}}\right)$$

Since there are $\varphi(m) = O(m/\log\log x) = O(N^{1/4r}/(\log\log N - \log 4r))$ values of i, the total time complexity is

$$O\left(\frac{N^{1/4r}}{\log\log N - \log 4r} \cdot \frac{\log^{7+3\epsilon} N}{r^{2+\epsilon}}\right)$$

□

Remark 6.4. Compared to Theorem 6.1, Theorem 6.3 does not always yield a genuine $\log\log N$-speedup. When $r = O(1)$, the improvement $\log\log N - \log 4r$ can indeed be interpreted as a $\log\log N$-speedup. However, in the case where $r = \Theta(\log N/\log\log N)$, the difference becomes $\Theta(\log\log\log N)$, which is asymptotically weaker. A similar issue arises in Proposition 4.4 of [HH24], where the authors claim that "a more sophisticated choice of m can give a $\log\log N$ speedup." This assertion is not entirely accurate in general.

6.2. **Factor $N = p^r q$ with $q = \Theta(N^{1/2})$.** Next, we focus on this specific scenario and demonstrate how to use our rank-3 lattice to solve the r-power divisor problem. For $r > \frac{\log N}{32\log\log N}$, then $p < \log^{1/64} N$, which means we can find p in poly$(\log N)$ time by enumeration. Then we consider $r < \frac{\log N}{32\log\log N}$.

We now present the main search algorithm:

Algorithm 6.5 (The main search)**.**

Input:

- A integer $N = p^r q$ with $N^{1/2} < q \le N^{1/2}/c$, where $r < \frac{\log N}{32\log\log N}$ and $c < 1$ is a constant.
- Positive integers $X = \lfloor N^{1/4r} \log^{\gamma} N \rfloor$, where $\gamma = 1/2 - 1/2\log r$.
- An element $\alpha \in \mathbb{Z}_N^*$ whose order larger than $k = \lfloor \frac{2erX}{c} \rfloor$.

Output: p and q.

1: **for** $i = 0, \ldots, k-1$ **do** ▷ Computation of babysteps
2: Compute $\alpha^i \pmod N$ and $\gcd(\alpha^i - 1, N)$.
3: **if** $\gcd(\alpha^i - 1, N) > 1$ **then**
4: Compute p and q and return p and q.
5: **for** $j = \left\lfloor \frac{c^{1/r} N^{1/2r}}{X} \right\rfloor, \ldots, \left\lfloor \frac{N^{1/2r}}{X} \right\rfloor$ **do** ▷ Computation of giantsteps

6: $M_j \coloneqq jX$

7: Construct 3-rank lattice with basis

$$B_j = \begin{pmatrix} N & 0 & \dots & 0 & 0 \\ M_j^r & rXM_j^{r-1} & \dots & X^r & 0 \\ M_j^{r+1} & (r+1)M_j^r X & \dots & (r+1)M_j X^r & X^{r+1} \end{pmatrix} \tag{6.1}$$

8: Apply Lemma 2.4 (LLL Algorithm) and take the second vector

$$v_j = (v_j^0, v_j^1 X, \dots, v_j^{r+1} X^{r+1}).$$

9: Compute

$$x_j = \alpha^{\sum_{\ell=0}^{r+1} v_j^\ell (1-j)^\ell} \pmod N$$

10: Applying a sort-and-match algorithm to the babysteps and giantsteps computed in Steps 1 and 5, find all pairs (i, j) such that

$$\alpha^i \equiv x_j \pmod N. \tag{6.2}$$

For each such match, solve the quadratic equation

$$\sum_{\ell=0}^{r+1} v_j^\ell (p-j)^\ell = ip^r$$

If p is found, return p and N/p^r.

11: Let $x_1, \dots, x_n$ be the list of giantsteps computed in Step 5, skipping those that were discovered in Step 10 to be equal to one of the babysteps. Apply Algorithm 4.1 (finding collisions) with N, $\kappa \coloneqq k$, $\alpha \coloneqq \alpha \pmod N$ and $x_1, \dots, x_n$ as inputs.

12: Return p and q.

Now we will prove the following proposition of our main search.

Proposition 6.6. *Algorithm 4.3 is correct. It runs in time*

$$O(\sqrt{r} N^{1/4r} \log^{5/2} N). \tag{6.3}$$

Proof. The correctness and most parts of the complexity analysis follow similarly to the proof of Proposition 4.5. Analogous to Lemma 4.4, we establish that the second vector obtained from the LLL algorithm has a non-zero final coordinate. Since $v_j^{r+1} \neq 0$, we avoid trivial collisions. Our analysis will focus primarily on the giant-step baby-step component.

We can construct two linearly independent lattice vectors: $(M_j^r, rXM_j^{r-1}, \dots, X^r, 0)$ and

$$(M_j^{r+1}, (r+1)M_j^r X, \dots, (r+1)M_j X^r, X^{r+1}) - M_j(M_j^r, rXM_j^{r-1}, \dots, X^r, 0),$$

which simplifies to $(0, M_j^r X, \dots, rM_j X^r, X^{r+1})$. Computing its norm yields:

$$\begin{aligned}\left\|(0, M_j^r X, \ldots, rM_j X^r, X^{r+1})\right\| &= \sqrt{\sum_{i=0}^{r} \left(\binom{r}{i} M_j^i X^{r+1-i}\right)^2} \\ &\leqslant \sqrt{\left(\sum_{i=0}^{r} \binom{r}{i} M_j^i X^{r+1-i}\right)^2} \\ &= X(X + M_j)^r \\ &= XM_j^r \left(1 + \frac{X}{M_j}\right)^r \\ &= XM_j^r \left(1 + \frac{\log^\gamma N}{N^{1/4r}}\right)^r.\end{aligned}$$

Since

$$4r \log r + 4r \log\log N < \frac{1}{2} \log N + \frac{1}{2} \log N = \log N,$$

we obtain

$$1 + \frac{\log^\gamma N}{N^{1/4r}} < 1 + \frac{\log N}{N^{1/4r}} < 1 + \frac{1}{r}.$$

Therefore,

$$\left\|(0, M_j^r X, \ldots, rM_j X^r, X^{r+1})\right\| < XM_j^r \left(1 + \frac{\log^\gamma N}{N^{1/4r}}\right)^r < eXN^{1/2}.$$

Let the second vector obtained from the LLL algorithm in $\mathcal{L}(B_j)$ be $v_j = (v_j^0, v_j^1 X, \ldots, v_j^{r+1} X^{r+1})$. Then $\|v_j\| < 2\lambda_2(\mathcal{L}(B_j)) < 2eXN^{1/2}$.

Write $p = \lfloor p/X \rfloor X + (p - \lfloor p/X \rfloor X)$ and define $j_0 = \lfloor p/X \rfloor$ and $x_0 = p - j_0 X$. Then:

$$p^r \mid v_{j_0}^0 + v_{j_0}^1 x_0 + \cdots + v_{j_0}^{r+1} x_0^{r+1}. \tag{6.4}$$

By the Cauchy–Schwarz inequality and our bound on $\|v_j\|$:

$$\left|v_{j_0}^0 + v_{j_0}^1 x_0 + \cdots + v_{j_0}^{r+1} x_0^{r+1}\right| \leqslant \|v_{j_0}\| \cdot \sqrt{(r+2)X^2} < 2eXN^{1/2} \cdot \sqrt{r} X < 2erXN^{1/2}.$$

Since $N = p^r q$ and $q > N^{1/2}/c$, we have $p^r < cN^{1/2}$. Thus:

$$v_{j_0}^0 + v_{j_0}^1 x_0 + \cdots + v_{j_0}^{r+1} x_0^{r+1} = kp^r \quad \text{for some } |k| \leqslant \frac{2erX}{c}.$$

Using the congruence $x_0 \equiv p - j_0 X \equiv p - M_{j_0} \equiv 1 - M_{j_0} \pmod{p}$, we obtain:

$$\alpha^k \equiv \alpha^{(v_{j_0}^0 + v_{j_0}^1 (1 - M_{j_0}) + \cdots + v_{j_0}^{r+1} (1 - M_{j_0})^{r+1})/p^r} \pmod{p}. \tag{6.5}$$

Define

$$w_i = \alpha^{v_i^0 + v_i^1 (1 - M_i) + \cdots + v_i^{r+1} (1 - M_i)^{r+1}},$$

and construct the polynomial

$$f(x) = \prod_{i=1}^{M} (x - w_i) \in \mathbb{Z}_N[x]. \tag{6.6}$$

By Proposition 4.2, evaluating f at $\alpha^0, \ldots, \alpha^{k-1}$ takes time

$$O(N^{1/4r} \log^{3-\gamma} N + rN^{1/4r} \log^{2+\gamma} N).$$

At least one evaluation $f(\alpha^{k_0})$ will satisfy $\gcd(f(\alpha^{k_0}), N) > 1$, thus yielding a factor of N. With $\gamma = \frac{1}{2} - \frac{1}{2\log r}$, the overall complexity of our algorithm is $O(\sqrt{r}N^{1/4r}\log^{5/2} N)$. □

Remark 6.7. When $r = O\left(\frac{\log N}{\log\log N}\right)$, we have $p = \mathrm{poly}(\log N)$, so p can be found in $\mathrm{poly}(\log N)$ time by brute-force enumeration. Now consider the case where $r = o\left(\frac{\log N}{\log\log N}\right)$. In this regime, the complexity of our algorithm is $O\left(\sqrt{r}N^{1/(4r)}\log^{5/2} N\right)$. Compared to Strassen's method, which has complexity $O\left(N^{1/(4r)}\log^3 N\right)$ [Har21, Proposition 2.5], our method is strictly better by a factor of $\sqrt{r}/\log^{1/2} N$.

Remark 6.8. The LLL step takes time

$$O\left(\frac{N^{1/(2r)}}{X}\cdot r\log^2 N\log\log N\right) = O\left(rN^{1/(4r)}\log^{2-\gamma} N\log\log N\right),$$

which is negligible compared to the overall complexity. Moreover, the LLL step can be omitted entirely, as the vector $(0, M_j^r X, \ldots, rM_j X^r, X^{r+1})$ can be directly used to construct the giant step.

Finally we present the main factoring algorithm. In this algorithm, N_0 is a constant that is chosen large enough to ensure that the proof of correctness works for all $N \geqslant N_0$.

Algorithm 6.9 (Factoring r-powers)**.**
Input: A integer $N = p^r q \geq N_0$ with $N^{1/2} < q \leq N^{1/2}/c$. *Output:* p and q.

1: **if** $r > \frac{\log N}{32\log\log N}$ **then**
2: **for** $i = 0, \ldots, \log N$ **do**
3: **if** $i^r | N$ **then**
4: Recover p and q and return.
5: Apply Lemma 3.4 with N, r and $\delta = N^{1/4r}\log^8 N$. If any factors of N are found, return. Otherwise, we obtain $\alpha \in \mathbb{Z}_N^*$ with $\mathrm{ord}_N(\alpha) > \delta$.
6: Run Algorithm 6.5 (the main search) with the given N, r and α. Return p and q.

Proposition 6.10. *Algorithm 6.9 is correct (for suitable N_0), and it runs in time*

$$O(\sqrt{r}N^{1/4r}\log^{5/2} N).$$

Proof. For correctness, we first handle the case $r > \frac{\log N}{32\log\log N}$ by enumerating all $p < \log N$. For the case $r < \frac{\log N}{32\log\log N}$, we can recover p and q via the main search step. Moreover, in this case, we have $rX < N^{1/(4r)}\log^8 N$, which satisfies the requirement on the order size needed by the main search.

Regarding the algorithm's complexity: when $r > \frac{\log N}{32\log\log N}$, the enumeration of $p < \log N$ is negligible. For the case $r < \frac{\log N}{32\log\log N}$, finding α takes time less than $N^{1/(8r)}\log^4 N$, and the order checking is bounded by $N^{1/(4r)}$. Therefore, the most expensive step is the main search, which takes time

$$O(\sqrt{r}N^{1/4r}\log^{5/2} N).$$

□

7. Conclusion

In this work, we present a novel deterministic integer factorization approach that merges Coppersmith's method with Harvey and Hittmeir's Baby-step Giant-step framework through a specialized rank-3 lattice construction. The key insight of our work is utilizing the second vector from LLL-reduced lattice bases—rather than the traditional shortest vector—to generate giant steps that avoid trivial collisions, resulting in more efficient factorization algorithms with improved asymptotic complexity bounds.

Our contributions span multiple factorization problems: For semiprimes with known bit size of p or q, we achieve a logarithmic improvement in complexity compared to previous results. For numbers representing sums and differences of powers, our algorithm also provides significant logarithmic improvements. .

We also introduce several technical innovations that may be of independent interest: an improved approach to finding elements of high multiplicative order; a refined generalization of Harvey's deterministic factorization method for identifying r-power divisors; and an extension of Coppersmith's method that relaxes determinant constraints while preserving practical utility.

An open question remains whether our approach could directly improve the general bound established by Harvey and Hittmeir [HH22b]. Without additional information about the factors p or q, our algorithm currently yields the same complexity bound $O\left(\frac{N^{1/5}\log^{16/5}N}{(\log\log N)^{3/5}}\right)$ as their original work.

References

[Cop96] Don Coppersmith, *Finding a small root of a univariate modular equation*, International Conference on the Theory and Applications of Cryptographic Techniques(EUROCRYPT'96), Springer, 1996, pp. 155–165.

[CP05] Richard E Crandall and Carl Pomerance, *Prime numbers: a computational perspective*, vol. 2, Springer, 2005.

[Gal12] Steven D Galbraith, *Mathematics of public key cryptography*, Cambridge University Press, 2012.

[Har21] David Harvey, *An exponent one-fifth algorithm for deterministic integer factorisation*, Math. Comp. **90** (2021), 2937–2950.

[HG97] Nicholas Howgrave-Graham, *Finding small roots of univariate modular equations revisited*, IMA International Conference on Cryptography and Coding, Springer, 1997, pp. 131–142.

[HG01] Nick Howgrave-Graham, *Approximate integer common divisors*, International cryptography and lattices conference, Springer, 2001, pp. 51–66.

[HH22a] David Harvey and Markus Hittmeir, *A deterministic algorithm for finding r-power divisors*, Research in Number Theory **8** (2022), no. 4, 94.

[HH22b] David Harvey and Markus Hittmeir, *A log-log speedup for exponent one-fifth deterministic integer factorisation*, Math. Comp. **91** (2022), 1367–1379.

[HH24] Jonathon Hales and Ghaith Hiary, *A generalization of lehman's method*, The Ramanujan Journal (2024), 1–18.

[Hit17] Markus Hittmeir, *Deterministic factorization of sums and differences of powers*, Math. Comp. **86** (2017), no. 308, 2947–2954.

[Hit18] M. Hittmeir, *A babystep-giantstep method for faster deterministic integer factorization*, Math. Comp. **87** (2018), no. 314, 2915–2935.

[HVDH21] David Harvey and Joris Van Der Hoeven, *Integer multiplication in time $O(n\log n)$*, Annals of Mathematics **193** (2021), no. 2, 563–617.

[Iwa78] Henryk Iwaniec, *On the problem of jacobsthal*, Demonstratio Mathematica **11** (1978), no. 1, 225–232.

[Leh74] R Sherman Lehman, *Factoring large integers*, Math. Comp. **28** (1974), no. 126, 637–646.

[LJ87] Hendrik W Lenstra Jr, *Factoring integers with elliptic curves*, Annals of Mathematics (1987), 649–673.

[LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. **261** (1982), 515–534.

[May03] Alexander May, *New rsa vulnerabilities using lattice reduction methods.*, Ph.D. thesis, Citeseer, 2003.

[NS16] A. Neumaier and D. Stehlé, *Faster LLL-type reduction of lattice bases*, Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Association for Computing Machinery, 2016, pp. 373–380.

[Sho94] Peter W Shor, *Algorithms for quantum computation: discrete logarithms and factoring*, Proceedings 35th Annual Symposium on Foundations of Computer Science(FOCS'94), IEEE, 1994, pp. 124–134.

Appendix A. Proof of Theorem 3.1

Proof. Without loss of generality, we may assume $s \leqslant m-1$. Let $p = mx_0 + s$ where $p^r | N$. This implies $p \leqslant N^{1/r}$ and $|x_0| \leqslant p/m$. Let $t \equiv m^{-1} \pmod N$. We observe that $p|(st + x_0)$.

Define a sequence $\{X_i\}$ where $X_0 = N^{1/r}$, $X_1 = X_0/2, \ldots, X_k = X_0/2^k$ with $k = \lfloor \log N/r \rfloor$. Let $X_i = N^{\beta_i}$ where $\beta_{i+1} = \beta_i - \frac{1}{\log N}$.

For each interval $[X_{i+1}, X_i]$, consider the polynomial $f_r(x) = (st + x)^r$. When $p \in [X_{i+1}, X_i]$, we have:

$$f_r(x_0) \equiv 0 \pmod{p^r}, \quad |x_0| \leqslant X_i/m = N^{\beta_i}/m$$

Applying Lemma 2.8 with $\delta = r, b = p^r$, $b \geqslant N^\beta = X_{i+1}^r$, $\beta = r\beta_{i+1}$, and $c = X_i/m/N^{\beta^2/r}$, we obtain an upper bound for c:

$$c = \frac{X_i}{mN^{\beta^2/r}} = \frac{N^{\beta_i - r\beta_{i+1}^2}}{m} = \frac{2N^{\beta_{i+1} - r\beta_{i+1}^2}}{m} \leqslant \frac{2N^{1/4r}}{m}$$

Thus, for each interval $[X_{i+1}, X_i]$ where $i \in [k-1]$, we can find all p satisfying $p \in [X_{i+1}, X_i]$ and $p^r | N$ in time:

$$O\left(\lceil c \rceil \frac{\log^{6+3\epsilon} N}{\delta^{1+\epsilon}}\right) = O\left(\left\lceil \frac{N^{1/4r}}{m} \right\rceil \frac{\log^{6+3\epsilon} N}{r^{1+\epsilon}}\right)$$

We repeat this process for intervals $[X_k, X_{k-1}], \ldots, [X_2, X_1], [X_1, X_0]$. For $[0, X_k]$, we perform exhaustive search for $p \equiv s \pmod m$. With $k = \lfloor \log N/r \rfloor$ intervals, the total time complexity is:

$$\left\lfloor \frac{\log N}{r} \right\rfloor O\left(\left\lceil \frac{N^{1/4r}}{m} \right\rceil \frac{\log^{6+3\epsilon} N}{r^{1+\epsilon}}\right) = O\left(\left\lceil \frac{N^{1/4r}}{m} \right\rceil \frac{\log^{7+3\epsilon} N}{r^{2+\epsilon}}\right)$$

Since $X_k = N^{1/r}/2^k = O(1)$, the exhaustive search complexity is $O(\log N)$, which is negligible in the overall complexity. □

Appendix B. Order-finding algorithms for r-powers

Lemma B.1 ([Hit18], Theorem 6.1)**.** *There exists an algorithm with the following properties:*

- ***Input:*** $N \in \mathbb{N}$, $T \leqslant N$ *and* $a \in \mathbb{Z}_N^*$.
- ***Output:*** *If* $\mathrm{ord}_N(a) \leqslant T$, *then the output is* $\mathrm{ord}_N(a)$*; otherwise, the output is '*$ord_N(a) > T$*'.*

The runtime complexity is bounded by

$$O\left(\frac{T^{1/2}}{\sqrt{\log\log T}}\cdot\log^2 N\right).$$

Algorithm B.2 (Order-finding algorithms)**.**
Input:
– $N\in\mathbb{N}$ and $\delta\leqslant N$.
Output:
– Either some $a\in\mathbb{Z}_N^*$ such that $\mathrm{ord}_N(a)>\delta$, or a nontrivial factor of N, or "N is r power free".

1: Set $M_1=1$ and $a=2$
2: **for** $e=1,2,\ldots$ **do**
3: **while** $a\nmid N$ **and** $a^{M_e}\equiv 1 \pmod N$ **do**
4: $a\leftarrow a+1$
5: **if** $a\mid N$ **then**
6: **return** a as a nontrivial factor of N, or if $a=N$, return 'N is prime'.
7: Apply Lemma B.1 with $T=\delta^{1/2}/\log^2 N$.
8: **if** $\mathrm{ord}_N(a)$ is not found **then**
9: Apply Lemma B.1 with $T=\delta$.
10: **if** $\mathrm{ord}_N(a)$ is not found **then**
11: **return** a as an element with $\mathrm{ord}_N(a)>\delta$.
12: Set $m_e=\mathrm{ord}_N(a)$ and compute the prime factorization of m_e.
13: **for** each prime p dividing m_e **do**
14: **if** $\gcd(N,a^{m_e/p}-1)\neq 1$ **then**
15: **return** $\gcd(N,a^{m_e/p}-1)$ as a nontrivial factor of N.
16: Set $M_{e+1}\leftarrow\mathrm{lcm}(M_e,m_e)$
17: **if** $M_{e+1}\geqslant\delta^{1/2}/\log^2 N$ **then**
18: Apply Theorem 3.1 with $r,s=1,m=M_{e+1}$.
19: **return** some nontrivial factor of N or "N is r power free".
20: $a\leftarrow a+1$

Theorem B.3. *Algorithm B.2 is correct. Assuming $N^{1/4r}\log^8 N\leqslant\delta$, the runtime complexity of Algorithm B.2 is bounded by*

$$O\left(\frac{\delta^{1/2}\log^2 N}{(\log\log\delta)^{1/2}}\right).$$

Proof. Our algorithm differs from [Hit18, Algorithm 6.2] in two aspects: we set $T=\delta^{1/2}/\log^2 N$ instead of $T=\delta^{1/3}$ in Steps 7 and 17, and we apply Theorem 3.1 for detecting r-powers of N in Step 18. Since Theorem 3.1 identifies all primes p satisfying $p\equiv s \pmod m$ and $p^r|N$, these modifications preserve the correctness established in the proof of [Hit18, Theorem 6.3]. We now analyze the time complexity.

Let us first examine the running time of Step 18. When the algorithm reaches this step, we have $M_e\geqslant\delta^{1/2}/\log^2 N$. Given our assumption that $N^{1/4r}\log^8 N\leqslant\delta$, Theorem 3.1 bounds the computational cost by:

$$O\left(\left\lceil\frac{N^{1/4r}}{M_e}\right\rceil\frac{\log^{7+3\epsilon}N}{r^{2+\epsilon}}\right)=O\left(\frac{\delta^{1/2}\log^{1+3\epsilon}N}{r^{2+\epsilon}}\right).$$

This shows that the cost of Step 18 is asymptotically negligible.

When the algorithm reaches Step 9, it terminates within the claimed running time. By Lemma B.1, Step 9 requires

$$O\left(\frac{\delta^{1/2}}{\sqrt{\log\log\delta}}\cdot\log^2 N\right).$$

If $\mathrm{ord}_N(a)$ is not found, the algorithm terminates. Otherwise, assuming $\mathrm{ord}_N(a)\leqslant\delta$, we note that $m_e>\delta^{1/2}/\log^2 N$ since the algorithm reached Step 9. This implies $M_e\geqslant m_e>\delta^{1/2}/\log^2 N$, leading to Step 18, whose negligible runtime was analyzed above.

For the e-th iteration of the main loop, each while loop execution in Steps 3-4 takes $O(\mathsf{M}(\log N)\log M_e)=O(\log^2 N\log\log N)$ operations, with at most $M_e<\delta^{1/2}/\log^2 N$ iterations. Thus, the total cost is bounded by

$$O\left(\delta^{1/2}\log\log N\right).$$

Step 7 requires $O\left(\delta^{1/4+o(1)}\right)$ bit operations by Lemma B.1. If $\mathrm{ord}_N(a)$ is found and since $m_e<\delta^{1/2}/\log^2 N$, factoring m_e in Step 12 via trial division costs $O\left(\delta^{1/4+o(1)}\right)$ bit operations. All remaining computations are polynomial-time.

Therefore, one complete main loop iteration requires $O\left(\delta^{1/2}\log\log N\right)$ bit operations. Since the value of M_{e+1} is at least twice as large as M_e, it iterates at most $O(\log\delta)$ until M_e reaches $\delta^{1/2}/\log^2 N$. The total time until reaching either Step 9 or Step 18 is $O\left(\delta^{1/2}\log N\log\log N\right)$, which is asymptotically negligible. This completes the proof. □

School of Cyber Science and Technology, University of Science and Technology of China, Hefei, China
Email address: qw1234567@mail.ustc.edu.cn

State Key Laboratory of Mathematical Sciences, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China and School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing, China
Email address: fengyansong@amss.ac.cn

School of Cyber Science and Technology, University of Science and Technology of China, Hefei, China
Email address: hghu2005@ustc.edu.cn

State Key Laboratory of Mathematical Sciences, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China and School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing, China
Email address: panyanbin@amss.ac.cn