Rounding LLL: Finding Faster Small Roots of Univariate Polynomial Congruences

Jingguo Bi[*] and Phong Q. Nguyen[†]

May 30, 2013

Abstract

In a seminal work at EUROCRYPT '96, Coppersmith showed how to find all small roots of a univariate polynomial congruence in polynomial time: this has found many applications in public-key cryptanalysis and in a few security proofs. However, the running time of the algorithm is a high-degree polynomial, which limits experiments: the bottleneck is an LLL reduction of a high-dimensional matrix with extra-large coefficients. We present in this paper a polynomial speedup over Coppersmith's algorithm. Our improvement is based on a special property of the matrices used by Coppersmith's algorithm, which allows us to speed up the LLL reduction by rounding. The exact speedup depends on the LLL algorithm used: for instance, the speedup is quadratic in the bit-size of the small-root bound if one uses the Nguyen-Stehlé L^2 algorithm.

Keywords: Coppersmith's Algorithm, Small Roots of Polynomial Equations, LLL, Complexity, Speedup, RSA.

1 Introduction

At EUROCRYPT '96, Coppersmith [6, 5, 7] showed how to find efficiently all small roots of polynomial equations (modulo an integer, or over the integers). The simplest (and perhaps most popular) result is the following: Given an integer N of unknown factorization and a monic polynomial $f(x) \in \mathbb{Z}[x]$ of degree δ, Coppersmith's lattice-based algorithm finds all integers $x_0 \in \mathbb{Z}$ such that $f(x_0) \equiv 0 \pmod{N}$ and $|x_0| \leq N^{1/\delta}$ in time polynomial in $\log N$ and δ. This has many applications in public-key cryptanalysis (*e.g.* attacking special cases of RSA and factoring with a hint), but also in a few security proofs (such as in RSA-OAEP [21]). Accordingly, Coppersmith's seminal work has been followed up by dozens of articles (see May's survey [13] for references), which introduced new variants, generalizations, simplifications and applications.

All these small-root algorithms are based on the same idea of finding new polynomial equations using lattice basis reduction: it reduces the problem of finding small roots to finding LLL-short vectors in a lattice. This can theoretically be done in polynomial time using the LLL algorithm [12], but is by no means trivial in practice: the asymptotical running time is a high-degree polynomial, because the lattice is huge. More precisely, May's recent survey [13] gives for Coppersmith's lattice-based algorithm the complexity upper bound $O(\delta^5 \log^9 N)$ using the Nguyen-Stehlé L^2 algorithm [16] as the reduction algorithm. A careful look gives a somewhat better upper bound: asymptotically, one may take a matrix of dimension $O(\log N)$, and bit-size $O((\log^2 N)/\delta)$, resulting in a complexity upper bound $O((\log^9 N)/\delta^2)$ using L^2. In typical applications, δ is small ≤ 9 but $\log N$ is the bit-size of an RSA modulus, *i.e.* at least 1024 bits, which makes the theoretical running time daunting: $\log^9 N$ is already at least 2^{90}. For more powerful variants of Coppersmith's algorithm, the running time is even worse, because the lattice dimension and/or the bit-size increase: for instance, Coron [8] gives the upper bound $O(\log^{11} W)$ for finding small roots over bivariate equations over the integers (W plays a role similar to N in the univariate congruence case), using L^2.

The bottleneck of all Coppersmith-type small-root algorithms is the LLL reduction. Despite considerable attention, no significant improvement on the running time has been found, except that LLL algorithms have improved since [7],

[*]Tsinghua University, Institute for Advanced Study, China. `jingguobi@mail.tsinghua.edu.cn`

[†]INRIA, France & Tsinghua University, Institute for Advanced Study, China. `http://www.di.ens.fr/~pnguyen/`.

with the appearance of L^2 [16] and $\tilde{L}^1$ [18]. And this issue is reflected in experiments (see [9]): in practice, one settles for sub-optimal parameters, which means that one can only find small roots up to a bound lower than the asymptotical bound. To illustrate this point, the celebrated Boneh-Durfee attack [1] on RSA with short secret exponent has the theoretical bound $d \leq N^{1-1/\sqrt{2}} \approx N^{0.292}$, but the largest d in the Boneh-Durfee experiments is only $d \approx N^{0.280}$ with a 1000-bit N, and much less for larger N, *e.g.* $d \approx N^{0.265}$ for 4000-bit N.

OUR RESULTS. We present a polynomial speedup over Coppersmith's algorithm for finding small roots of univariate polynomial congruences. The exact speedup depends on the LLL algorithm used: if one uses L^2 [16], the total bit-complexity is upper bounded by $O(\log^7 N)$, which gives a speedup $\Theta((\log^2 N)/\delta^2)$ quadratic in the bit-size of the small-root bound $N^{1/\delta}$; and if one uses $\tilde{L}^1$ [18], the total complexity is upper bounded by $O(\log^{6+\varepsilon} N)$ for any $\varepsilon > 0$ using fast integer arithmetic, which gives a speedup $O((\log N)/\delta)$ linear in the bit-size of the small-root bound $N^{1/\delta}$.

Our improvement comes from combining LLL reduction with rounding: instead of LLL-reducing directly a matrix with huge entries, we suitably round the coefficients before LLL reduction to make them much smaller, and show that the LLL output allows to derive sufficiently short vectors in the original lattice. In practice, this means that for any instantiation of Coppersmith's algorithm achieving a small-root bound X, we can drastically reduce the size of the coefficients of the matrix to be LLL-reduced and achieve essentially the same small-root bound: asymptotically, the bit-size is reduced by a factor $(\log N)/\delta$, which implies that the speedup is quadratic when using the popular L^2 algorithm, or quasi-linear using the more theoretical $\tilde{L}^1$ algorithm.

This rounding strategy is very natural, but it is folklore that it fails in the worst case: when an arbitrary non-singular matrix is rounded, it may even become singular, and the situation is worse for LLL reduction. However, we show that a well-chosen rounding strategy surprisingly works for the special matrices used by Coppersmith's algorithm: this is because the matrices to be reduced are triangular matrices whose diagonal entries are reasonably balanced, which can be exploited.

To the best of our knowledge, this is the first example of polynomial speedup for LLL reduction of special matrices. Previously, it was known that the complexity upper bound of the LLL algorithm could be improved by some polynomial factor for certain matrices, *e.g.* the so-called knapsack lattices (see [15]), but it should be stressed that only the analysis was improved, not the algorithm. Here, in Coppersmith's algorithm, it is not known how to improve the worst-case analysis of LLL, and the algorithm needs to be modified to improve the complexity upper bound.

Finally, our work helps to clarify the asymptotical complexity of Coppersmith's algorithm for univariate polynomial congruences. Despite the importance of the algorithm, it seems that the dependence on the polynomial degree δ was not well-understood: as previously mentioned, May's survey [13] gave an upper bound including a factor δ^5, and Coppersmith's journal article [7] gave an upper bound growing exponentially in δ. Our final complexity upper bound is independent of δ: it only depends on the bit-size of the modulus N.

ROADMAP. In Sect. 2, we recall background on lattices and Coppersmith's small-root algorithm. In Sect. 3, we present our speedup of Coppersmith's algorithm by rounding, together with a theoretical analysis and experimental results. Finally, we discuss the case of other small-root algorithms in Sect. 4.

2 Background and Notation

We use row representation for matrices: vectors are row vectors denoted by bold lowercase letters, matrices are denoted by uppercase letters, and their coefficients are denoted by lowercase letters. All logarithms are in base 2. Let $\|\|$ and $\langle , \rangle$ be the Euclidean norm and inner product of $\mathbb{R}^n$. The Euclidean norm is naturally extended to polynomials as follows: if $f(x) = \sum_{i=0}^{n} f_i x^i \in \mathbb{R}[x]$, then $\|f\| = (\sum_{0 \leq i \leq n} f_i^2)^{1/2}$. We use the following matrix norms: if $M = (m_{i,j})$ is an $n \times m$ matrix, then $\|M\|_2 = \max_{\|\mathbf{x}\| \neq 0} \frac{\|\mathbf{x}M\|}{\|\mathbf{x}\|}$, and $\|M\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{m} |m_{i,j}|$. Then: $\|M\|_2 \leq \sqrt{n}\|M\|_\infty$. If $x \in \mathbb{R}$, we denote by $\lceil x \rfloor$ a closest integer to x.

2.1 Lattices

LATTICES. A lattice L is a discrete subgroup of $\mathbb{R}^m$: there exist $n(\leq m)$ linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$ s.t. that L is the set $\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of all integral linear combinations of the $\mathbf{b}_i$'s, *i.e.*

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\}.$$

Then the matrix $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is called a *basis* of L and n is the *rank* (or *dimension*) of L. Here, we mostly consider full-rank lattices, *i.e.* $n = m$. The (co-)volume of L is $\mathrm{vol}(L) = \sqrt{\det(BB^t)}$ for any basis B of L, where B^t denotes B's transpose. If B is square, then $\mathrm{vol}(L) = |\det B|$, and if B is further triangular, then $\mathrm{vol}(L)$ is simply the product of the diagonal entries of B in absolute value.

GRAM-SCHMIDT ORTHOGONALIZATION. Let $\mathbf{b}_1, \cdots, \mathbf{b}_n \in \mathbb{R}^m$ be linearly independent vectors. The Gram-Schmidt orthogonalization (GSO) is the family $(\mathbf{b}_1^\star, \ldots, \mathbf{b}_n^\star)$ defined recursively as: $\mathbf{b}_1^\star = \mathbf{b}_1$ and for $i \geq 2$, $\mathbf{b}_i^\star$ is the component of the vector $\mathbf{b}_i$ which is orthogonal to the linear span of $\mathbf{b}_1, \cdots, \mathbf{b}_{i-1}$. Then $\mathbf{b}_i^\star = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^\star$, where $\mu_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j^\star \rangle / \|\mathbf{b}_j^\star\|^2$ for $1 \leq j < i \leq n$.

SIZE-REDUCTION. A basis $B = (\mathbf{b}_1, \cdots, \mathbf{b}_n)$ is *size-reduced* if its GSO satisfies $|\mu_{i,j}| \leq 1/2$, for all $1 \leq j < i \leq n$. There is a classical (elementary) algorithm which size-reduces a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of an integer lattice $L \subseteq \mathbb{Z}^m$, in polynomial time, without ever modifying the Gram-Schmidt vectors $\mathbf{b}_i^\star$: this algorithm is included in the original LLL algorithm [12]. In the special case that the input basis is (square) lower-triangular, the running-time of this size-reduction algorithm is $O(n^3 b^2)$ without fast integer arithmetic, and $n^3 \tilde{O}(b)$ using fast-integer arithmetic, where $b = \max_{1 \leq i \leq n} \log \|\mathbf{b}_i\|$.

LLL AND SHORT LATTICE VECTORS. Coppersmith's small-root method requires the ability to efficiently find reasonably short vectors in a lattice, namely a non-zero vector $\mathbf{v} \in L$ s.t. $\|\mathbf{v}\| \leq c^n \mathrm{vol}(L)^{1/n}$ where c is some constant and n is the lattice rank. This can be achieved by the celebrated LLL algorithm [12]: given a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of an integer lattice $L \subseteq \mathbb{Z}^m$, LLL outputs a non-zero $\mathbf{v} \in L$ s.t. $\|\mathbf{v}\| \leq 2^{\frac{n-1}{4}} \mathrm{vol}(L)^{1/n}$ in time $O(n^5 m b^3)$ (resp. $n^3 m b \tilde{O}(n) \tilde{O}(b)$) without (resp. with) fast integer arithmetic, where $b = \max_{1 \leq i \leq n} \log \|\mathbf{b}_i\|$: strictly speaking, this vector is actually the first vector of the basis output by the algorithm. Nguyen and Stehlé [16] introduced the L^2 algorithm, a faster variant of LLL which can output similarly short vectors in time $O(n^4 m(n+b)b)$ (resp. $n^2 m(n+b) b \tilde{O(n)}$) without (resp. with) fast integer arithmetic. The recent $\tilde{L}^1$ algorithm by Novocin *et al.* [18] can output similarly short vectors for a full-rank lattice in time $O(n^{5+\varepsilon} b + n^{\omega+1+\varepsilon} b^{1+\varepsilon})$ for any $\varepsilon > 0$ using fast integer arithmetic, where $\omega \leq 2.376$ is the matrix multiplication complexity constant. However, this algorithm is considered to be mostly of theoretical interest for now: $\tilde{L}^1$ is currently not implemented anywhere, as opposed to L^2. When assessing the complexity of LLL reduction, it is therefore meaningful to mention two complexities: one (closer to the real world) using L^2 without fast integer arithmetic, and another using $\tilde{L}^1$ using fast integer arithmetic and fast linear algebra.

The complexity upper bound of LLL reduction can sometimes be decreased by some polynomial factor, but not for Coppersmith's small-root method. For instance, it is folklore that the complexity of LLL and its variants include a multiplicative factor $n^2 b$, which can actually be replaced by the more precise term $\log \prod_{i=1}^{n} \|\mathbf{b}_i^\star\|^{n+1-i}$. This term is always less than $O(n^2 b)$, but for special input matrices, it can be much lower, therefore decreasing the LLL complexity upper bound: for instance, in the well-known case that the input matrix is lower-triangular with a single diagonal coefficient not equal to 1, then the term is $O(nb)$, which gains a linear factor $\Theta(n)$ (see [15] for more details). However, if the input matrix is lower-triangular with balanced diagonal coefficients, there is no improvement over the worst-case complexity upper bound: we will see that it is the case for Coppersmith's small-root method.

2.2 Coppersmith's method for finding small roots

At EUROCRYPT '96, Coppersmith [6, 5, 7] showed how to find efficiently all small roots of polynomial equations (modulo an integer, or multivariate over the integers), which is surveyed in [13, 14]. We now review the simplest result, following the classical Howgrave-Graham approach [10]: In Sect. 4, we will discuss the main variants of this result.

Theorem 2.1 (Coppersmith [6, 7]). *There is an algorithm (Alg. 1) which, given as input an integer N of unknown factorization and a monic polynomial $f(x) \in \mathbb{Z}[x]$ of degree δ and coefficients in $\{0, \ldots, N-1\}$, outputs all integers $x_0 \in \mathbb{Z}$ such that $f(x_0) \equiv 0 \pmod{N}$ and $|x_0| \leq N^{1/\delta}$ in time polynomial in $\log N$ and δ.*

In fact, Coppersmith's algorithm (Alg. 1) does not directly achieve the bound $N^{1/\delta}$: instead, it uses a subroutine (Alg. 2), which finds efficiently all roots up to some bound X ($< N^{1/\delta}$) depending on an integer parameter $h \geq 1$, chosen asymptotically to be $O((\log N)/\delta)$. When h is sufficiently large, X becomes sufficiently close to $N^{1/\delta}$ that one can find all roots up to $N^{1/\delta}$ by applying Alg. 2 a few times. We now explain the main algorithm: Alg. 2. The subroutine reduces

Algorithm 1 Coppersmith's algorithm [6, 7] for finding small roots of univariate polynomial congruences

```
Input: An integer $N \geq 1$ and a univariate degree-$\delta$ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0, \ldots, N-1\}$.
Output: All $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq N^{1/\delta}$ and $f(x_0) \equiv 0 \mod N$.
1:  if $\delta = 1$ then
2:      Return all $x_0 \in \mathbb{Z}$ s.t. $x_0 + f(0) \equiv 0 \pmod{N}$ and $|x_0| \leq N$.
3:  else
4:      if $\delta + 1 \geq (\log N)/2$ then
5:          Compute all $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq N^{1/\delta}$ and $f(x_0) \equiv 0 \mod N$ by exhaustive search.
6:      else
7:          Let $h = \lceil (\log N)/\delta \rceil$, $n = h\delta$, $X = \lfloor 2^{-1/2} N^{\frac{h-1}{n-1}} (n+1)^{-\frac{1}{n-1}} \rfloor$, and $t = \lfloor -N^{1/\delta} \rfloor + X$.
8:          while $t \leq N^{1/\delta}$ do
9:              Run Alg. 2 on $g(x) = f(x-t) \in \mathbb{Z}[x]$, $h$ and $N$.
10:             Output $x_0 + t$ for each root $x_0$ of $g(x)$ obtained.
11:             $t \leftarrow t + 2X$.
12:         end while
13:     end if
14: end if
```

Algorithm 2 Finding smaller roots of univariate polynomial congruences [6, 7]

```
Input: Two integers $N, h \geq 1$, and a monic polynomial $f(x) \in \mathbb{Z}[x]$ of degree $\delta \geq 2$ and coefficients in $\{0, \ldots, N-1\}$.
Output: All $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq X$ and $f(x_0) = 0 \mod N$, where $X = \lfloor 2^{-1/2} N^{\frac{h-1}{n-1}} (n+1)^{-\frac{1}{n-1}} \rfloor$ and $n = h\delta$.
1: if $\delta > \log N$ then
2:     Return 0 if $f(0) \equiv 0 \pmod{N}$.
3: else
4:     Build the $n \times n$ lower-triangular matrix $B$ whose rows are the $g_{i,j}(xX)$'s defined by (1) for $0 \leq i < h$ and
       $0 \leq j < \delta$.
5:     Run the $L^2$ algorithm [16] on the matrix $B$.
6:     The first vector of the reduced basis corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$.
7:     Compute all the roots $x_0$ of the polynomial $v(x) \in \mathbb{Z}[x]$ over $\mathbb{Z}$.
8:     Output all the roots $x_0$ which satisfy $f(x_0) \equiv 0 \pmod{N}$ and $|x_0| \leq X$.
9: end if
```

the problem to solving univariate polynomial equations over the integers, by transforming modular roots into integral roots. More precisely, it constructs a polynomial $g(x) \in \mathbb{Z}[x]$ such that: if $x_0 \in \mathbb{Z}$ is such that $f(x_0) \equiv 0 \pmod{N}$ and $|x_0| \leq X$, then $g(x_0) = 0$. To do so, it uses the following elementary criterion:

Lemma 2.2 (Howgrave-Graham [10]). *Let $g(x) \in \mathbb{Z}[x]$ be a polynomial with at most n non-zero coefficients. Let M be an integer ≥ 1. Assume that $\|g(xX)\| < \frac{M}{\sqrt{n}}$ for some $X \in \mathbb{R}$. If $x_0 \in \mathbb{Z}$ is such that $g(x_0) \equiv 0 \pmod{M}$ and $|x_0| \leq X$, then $g(x_0) = 0$.*

Lemma 2.2 will be used with $M = N^{h-1}$ and $g(x)$ found by lattice reduction. Let $h \geq 1$ be an integer and define the following family of $n = h\delta$ polynomials:

$$g_{i,j}(x) = x^j N^{h-1-i} f^i(x) \quad 0 \leq i < h, 0 \leq j < \delta \tag{1}$$

These n polynomials satisfy: if $f(x_0) \equiv 0 \pmod N$ for some $x_0 \in \mathbb{Z}$, then $g_{i,j}(x_0) \equiv 0 \pmod {N^{h-1}}$. In order to apply Lemma 2.2 for a bound $X \geq 1$ to be determined later, Coppersmith's algorithm constructs the n-dimensional lattice L spanned by the rows of the $n \times n$ matrix B formed by the n coefficient vectors of $g_{i,j}(xX)$, where the polynomials are ordered by increasing degree (*e.g.* in the order $(i, j) = (0, 0), (0, 1), \cdots, (0, \delta - 1), (1, 0), \cdots (h - 1, \delta - 1)$) and the coefficients are ordered by increasing monomial degree: the first coefficient is thus the constant term of the polynomial. The matrix B is lower triangular, and its n diagonal entries are:

$$\left(N^{h-1}, N^{h-1}X, \ldots, N^{h-1}X^{\delta-1}, \ldots, N^0 X^{\delta h-\delta}, \ldots, N^0 X^{\delta h-2}, N^0 X^{\delta h-1}\right), \tag{2}$$

because $f(x)$ is monic. In other words, the exponent of X increases by one at each row, while the exponent of N decreases by one every δ rows. It follows that:

$$\mathrm{vol}(L) = \det(B) = N^{\frac{1}{2}n(h-1)} X^{\frac{1}{2}n(n-1)}.$$

Alg. 2 applies the LLL algorithm to the matrix B, which provides a non-zero polynomial $v(x) \in \mathbb{Z}[x]$ such that:

$$\|v(xX)\| \leq 2^{\frac{n-1}{4}} \mathrm{vol}(L)^{\frac{1}{n}} = 2^{\frac{n-1}{4}} N^{\frac{h-1}{2}} X^{\frac{n-1}{2}}.$$

It follows that the polynomial $v(x)$ satisfies Lemma 2.2 with $M = N^{h-1}$ and $g(x) = v(x)$ if:

$$X \leq \frac{1}{\sqrt{2}} N^{\frac{h-1}{n-1}} (n+1)^{-\frac{1}{n-1}}. \tag{3}$$

The dimension of B is $n = h\delta$, and the entries of the matrix B have bit-size $O(h \log N)$, therefore the running time of L^2 in Step 5 is $O(\delta^6 h^7 \log N + \delta^5 h^7 \log^2 N)$, which is $O(\delta^5 h^7 \log^2 N)$ because $\delta \leq \log N$.

We obtain the following two concrete versions of Th. 2.1:

Theorem 2.3. *Given as input two integers $N \geq 1$ and $h \geq 1$, and a degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0, \ldots, N-1\}$, Alg. 2 outputs all $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq X$ and $f(x_0) = 0 \mod N$, where $X = \lfloor 2^{-1/2} N^{\frac{h-1}{n-1}} (n+1)^{-\frac{1}{n-1}} \rfloor$ and $n = h\delta$, in time $O(\delta^5 h^7 \log^2 N)$ without fast integer arithmetic, or $O(h^{6+\varepsilon} \delta^{5+\varepsilon} \log N + h^{\omega+2+2\varepsilon} \delta^{\omega+1+\varepsilon} \log^{1+\varepsilon} N)$ for any $\varepsilon > 0$ using fast integer arithmetic and $\tilde{L}^1$ in Step. 5, where $\omega \leq 2.376$ is the matrix multiplication complexity constant.*

Proof. If $\delta \leq \log N$, we already proved the correctness and the running time for L^2, and one can easily derive the improved running time using $\tilde{L}^1$. Otherwise, we have $N^{1/\delta} < 1$, which means that the only possible root is zero: this justifies Step 2, which takes time $O(\log N)$. □

Corollary 2.4. *Alg. 1 of Th. 2.1 runs in time $O((\log^9 N)/\delta^2)$ without fast integer arithmetic, or $O((\log^{7+\varepsilon} N)/\delta)$ for any $\varepsilon > 0$ using fast integer arithmetic and $\tilde{L}^1$ in Alg. 2.*

Proof. If $\delta + 1 \geq (\log N)/2$, we consider two cases: if $\delta > \log N$, Step 5 is the same as Step 2 in Alg. 2, which takes time $O(\log N)$; else $\delta \leq \log N$, so Step 5 takes time $O(\delta \log^2 N \times N^{1/\delta})$. In both cases, this is much less than $O((\log^9 N)/\delta^2)$. Otherwise, $\delta + 1 < (\log N)/2$. Now consider the bound $X = \lfloor 2^{-1/2} N^{\frac{h-1}{n-1}} (n+1)^{-\frac{1}{n-1}} \rfloor$ achieved by Alg. 2. By definition, Alg. 1 runs Alg. 2 at most $O(N^{1/\delta}/X)$ times. We have:

$$N^{1/\delta}/N^{\frac{h-1}{n-1}} = N^{1/\delta - \frac{h-1}{h\delta-1}} = N^{\frac{\delta-1}{\delta(h\delta-1)}} \leq N^{1/(h\delta-1)}$$

If $h = \lfloor \log N/\delta \rfloor$, then $h\delta - 1 \geq \log N - \delta - 1 \geq (\log N)/2$, therefore $N^{1/\delta}/N^{\frac{h-1}{n-1}} = O(1)$. □

Cor. 2.4 improves by δ^7 upon the upper bound $O(\delta^5 \log^9 N)$ given for Coppersmith's algorithm in May's survey [13]. Note that Coppersmith earlier announced in [7] that the running was polynomial in $\log N$ and 2^δ: we see that the dependence on δ is actually much better.

3 Speeding up Coppersmith's Algorithm by Rounding

Our main result is the following speedup over Coppersmith's algorithm (Corollary 2.4):

Theorem 3.1. *There is an algorithm (namely, Alg. 4) which, given as input an integer N of unknown factorization and a monic polynomial $f(x) \in \mathbb{Z}[x]$ of degree δ and coefficients in $\{0, \ldots, N-1\}$, outputs all integers $x_0 \in \mathbb{Z}$ such that $f(x_0) \equiv 0 \pmod{N}$ and $|x_0| \leq N^{1/\delta}$ in time $O(\log^7 N)$ without fast integer arithmetic using the L^2 algorithm [16], or $O(\log^{6+\varepsilon} N)$ for any $\varepsilon > 0$ using fast integer arithmetic and the $\tilde{L}^1$ algorithm [18] in Alg. 3, where $\omega \leq 2.376$ is the matrix multiplication complexity constant.*

3.1 Rounding for Coppersmith's Algorithm

The bottleneck of Coppersmith's algorithm (Algs. 1 and 2) is the LLL reduction of the matrix B, whose dimension is $n = h\delta$, and whose entries have bit-size $O(h \log N)$. Asymptotically, we have $h = O(\log N/\delta)$ so the dimension is $O(\log N)$ and the bit-size is $O((\log^2 N)/\delta)$. We will modify Coppersmith's algorithm in such a way that we only need to LLL-reduce a matrix of the same dimension but which much smaller entries, namely bit-length $O(\log N)$.

To explain the intuition behind our method, let us first take a closer look at the matrix B:

Lemma 3.2. *Let $X \leq N^{1/\delta}$. The maximal diagonal coefficient of B defined in Step. 4 of Alg. 2 is $N^{h-1}X^{\delta-1} < N^h$, the minimal diagonal coefficient is $X^{h\delta-\delta} \leq N^{h-1}$, and $\frac{N^{h-1}X^{\delta-1}}{X^{h\delta-\delta}} \geq N^{1-1/\delta}$ if $h \geq 2$. Furthermore, if $X \geq \Omega(N^{\frac{h-1}{n-1}})$, $h \geq 2$ and $h\delta = O(\log N)$ then:*

$$X^{h\delta-\delta} \geq N^{h-O(1)} \tag{4}$$

Proof. The $n = h\delta$ diagonal coefficients of B are naturally split into h blocks of δ coefficients: the i-th block is formed by the leading coefficients of the polynomials $g_{i,j}(xX)$ for $0 \leq j < \delta$. Since the leading coefficient of $g_{i,j}(xX)$ is $X^j N^{h-1-i} X^{\delta i}$, it follows that the maximal and minimal coefficients in the i-th block are located respectively at the end and at the beginning: their values are respectively $X^{\delta(i+1)-1}N^{h-1-i} = N^{h-1}(X^\delta/N)^i X^{\delta-1}$ and $N^{h-1-i}X^{\delta i} = N^{h-1}(X^\delta/N)^i$. If $X \leq N^{1/\delta}$, we obtain that the maximal diagonal coefficient is $N^{h-1}X^{\delta-1}$ reached in the 0-th block, and the minimal diagonal coefficient is $X^{(h-1)\delta}$ reached in the $(h-1)$-th block. And the ratio $\frac{N^{h-1}X^{\delta-1}}{X^{h\delta-\delta}}$ is exactly $N^{h-1}/X^{h\delta-2\delta+1}$ which is clearly $\geq N^{1-1/\delta}$ if $h \geq 2$.

Now, let $X_0 = N^{\frac{h-1}{n-1}}$ so that $X = \Omega(X_0)$. We have $N^{1/\delta}/N^{\frac{h-1}{n-1}} \leq N^{1/(h\delta-1)}$ by the proof of Cor. 2.4, therefore:

$$X_0 \geq N^{1/\delta - 1/(h\delta-1)} = N^{(h\delta-1-\delta)/(\delta(h\delta-1))}.$$

Hence:

$$X_0^\delta \geq N^{(h\delta-1-\delta)/(h\delta-1)} = N^{1-\delta/(h\delta-1)}.$$

Thus:

$$X_0^{h\delta-\delta} \geq N^{(h-1)-\delta(h-1)/(h\delta-1)} > N^{h-2}.$$

Since $X = \Omega(X_0)$ and $h\delta = O(\log N)$, we obtain (4). □

This implies that the diagonal coefficients of B are somewhat balanced: the matrix B is not far from being reduced. In fact, the first row of B has norm N^{h-1} which is extremely close to the bound $N^{h-1}/\sqrt{n}$ required by Lemma 2.2: intuitively, this means that it should not be too difficult to find a lattice vector shorter than $N^{h-1}/\sqrt{n}$. Still, the complexity upper bound of LLL is surprisingly unable to exploit the structure of B.

To take advantage of the structure of B, we first size-reduce B to make sure that the subdiagonal coefficients are smaller than the diagonal coefficients. Then we round the entries of B so that the smallest diagonal coefficient becomes $\lfloor c \rfloor$ where $c > 1$ is a parameter. More precisely, we create a new $n \times n$ triangular matrix $\tilde{B} = (\tilde{b}_{i,j})$ defined by:

$$\tilde{B} = \left\lfloor cB/X^{h\delta-\delta} \right\rfloor \tag{5}$$

By Lemma 3.2:

$$b_{i,i} \geq X^{h\delta-\delta} \text{ and } \tilde{b}_{i,i} \geq \lfloor c \rfloor \tag{6}$$

We LLL-reduce the rounded matrix $\tilde{B}$ instead of B: let $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$ be the first vector of the reduced basis obtained. If we applied to B the unimodular transformation that LLL-reduces $\tilde{B}$, we may not even obtain an LLL-reduced basis in general. However, because of the special structure of B, it turns out that $\mathbf{v} = \mathbf{x}B$ is still a short non-zero vector of L, as shown below:

Lemma 3.3. *Let $B = (b_{i,j})$ be an $n \times n$ lower-triangular matrix over $\mathbb{Z}$ with strictly positive diagonal. Let $c > 1$. If $\tilde{B} = \lfloor cB/\min_{i=1}^{n} b_{i,i} \rfloor$ and $\mathbf{x}\tilde{B}$ is the first vector of an LLL-reduced basis of $\tilde{B}$, then:*

$$0 < \|\mathbf{x}B\| < \left(n\|\tilde{B}^{-1}\|_2 + 1\right) 2^{\frac{n-1}{4}} \det(B)^{\frac{1}{n}}.$$

Proof. Let $\alpha = \min_{i=1}^{n} b_{i,i}/c$, so that $\tilde{B} = \lfloor B/\alpha \rfloor$. Define the matrix $\bar{B} = \alpha\tilde{B}$ whose entries are $\bar{b}_{i,j} = \alpha\tilde{b}_{i,j}$. Then $0 \leq b_{i,j} - \bar{b}_{i.j} < \alpha$, therefore $\|B - \bar{B}\|_2 < n\alpha$. We have:

$$\|\mathbf{x}B\| \leq \|\mathbf{x}(B - \bar{B})\| + \|\mathbf{x}\bar{B}\| \leq \|\mathbf{x}\| \times \|B - \bar{B}\|_2 + \alpha\|\mathbf{x}\tilde{B}\| < n\|\mathbf{x}\|\alpha + \alpha\|\mathbf{x}\tilde{B}\|.$$

Let $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$. Then $\|\mathbf{x}\| \leq \|\tilde{\mathbf{v}}\|\|\tilde{B}^{-1}\|_2$, and we obtain:

$$\|\mathbf{x}B\| < \left(n\|\tilde{B}^{-1}\|_2 + 1\right) \alpha\|\tilde{\mathbf{v}}\|.$$

The matrix $\tilde{B}$ is lower-triangular with all diagonal coefficients strictly positive because $c > 1$. Since $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$ is the first vector of an LLL-reduced basis of $\tilde{B}$, and $\tilde{B}$ is non-singular, $\mathbf{x}B \neq 0$ and we have:

$$\alpha\|\tilde{\mathbf{v}}\| \leq \alpha 2^{\frac{n-1}{4}} \det(\tilde{B})^{\frac{1}{n}} = 2^{\frac{n-1}{4}} \det(\bar{B})^{\frac{1}{n}} \leq 2^{\frac{n-1}{4}} \det(B)^{\frac{1}{n}},$$

where we used the fact that the matrices $\tilde{B}$, $\bar{B}$ and B are lower-triangular. The result follows by combining both inequalities. □

If $\mathbf{x}B$ is sufficiently short, then it corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$ satisfying Lemma 2.2, and the rest proceeds as in Alg. 2. The whole rounding algorithm is given in Alg. 3, which is supposed to be a faster variant of Alg. 2. Alg. 3 naturally gives rise to Alg. 4, a faster algorithm than Alg. 1 to compute all roots up to $N^{1/\delta}$. We now justify the bound X given in Alg. 3. In order for Lemma 3.3 to be useful, we need to upper bound $\|\tilde{B}^{-1}\|_2$. An upper bound can be derived from the following elementary lemma on inverses of triangular matrices.

Lemma 3.4. *Let $t > 0$ and $T = (t_{i,j})$ be an $n \times n$ lower-triangular matrix over $\mathbb{R}$, with unit diagonal (*i.e. *$t_{i,i} = 1$ for $1 \leq i \leq n$), and such that $|t_{i,j}| \leq t$ for $1 \leq j < i \leq n$. Then $\|T^{-1}\|_\infty \leq (1+t)^{n-1}$.*

Proof. Let $S = T^{-1}$. Then for $1 \leq i, j \leq n$: $\sum_{k=j}^{n} s_{i,k}t_{k,j} = \delta_{i,j}$, where $\delta_{i,j}$ is Kronecker's symbol. Therefore $s_{i,j} = \delta_{i,j} - \sum_{k=j+1}^{n} s_{i,k}t_{k,j}$, which implies that S is lower-triangular and for $1 \leq j < i \leq n$:

$$|s_{i,j}| \leq t\left(1 + \sum_{k=j+1}^{i-1} |s_{i,k}|\right). \tag{7}$$

Let us prove that for all $j < i$, $|s_{i,j}| \leq t(1+t)^{i-j-1}$, by induction over $i - j$. Since $|t_{i,j}| \leq t$ for $1 \leq j < i \leq n$: $|s_{i,i-1}| \leq t|s_{i,i}| = t$, which starts the induction for $i - j = 1$. Now, assume by induction that $|s_{i,k}| \leq t(1+t)^{i-k-1}$ for all k s.t. $i - k < i - j$ for some $1 \leq j < i \leq n$. Then (7) implies:

$$|s_{i,j}| \leq t\left(1 + \sum_{k=j+1}^{i-1} t(1+t)^{i-k-1}\right) = t\left(1 + t\sum_{k=0}^{i-j-2}(1+t)^k\right) = t\left(1 + t\frac{(1+t)^{i-j-1} - 1}{1+t-1}\right) = t(1+t)^{i-j-1}$$

Algorithm 3 Finding faster smaller roots of univariate polynomial congruences, by rounding

```
Input: Two integers $N, h \geq 2$, a parameter $c > 1$ and a monic polynomial $f(x) \in \mathbb{Z}[x]$ of degree $\delta \geq 2$ and coefficients in $\{0, \ldots, N-1\}$.
Output: All $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq X$ and $f(x_0) = 0 \bmod N$, where $X = \left\lfloor N^{(h-1)/(n-1)} 2^{-1/2} n^{-1/(n-1)} \left( n^{3/2} \left(\frac{3c-2}{2c-2}\right)^{n-1} \lfloor c \rfloor^{-1} + 1 \right)^{-2/(n-1)} \right\rfloor$ and $n = h\delta$.
1:  if $\delta > \log N$ then
2:      Return 0 if $f(0) \equiv 0 \pmod N$.
3:  else
4:      Build the $n \times n$ matrix $B$ whose rows are the $g_{i,j}(xX)$'s defined by (1).
5:      Size-reduce $B$ without modifying its diagonal coefficients.
6:      Compute the matrix $\tilde{B} = \lfloor cB/X^{h\delta-\delta} \rfloor$ obtained by rounding $B$.
7:      Run the $L^2$ algorithm [16] on the matrix $\tilde{B}$.
8:      Let $\tilde{\mathbf{v}} = \mathbf{x}\tilde{B}$ be the first vector of the reduced basis obtained.
9:      The vector $\mathbf{v} = \mathbf{x}B$ corresponds to a polynomial of the form $v(xX)$ for some $v(x) \in \mathbb{Z}[x]$.
10:     Compute all the roots $x_0$ of the polynomial $v(x) \in \mathbb{Z}[x]$ over $\mathbb{Z}$.
11:     Output all the roots $x_0$ which satisfy $f(x_0) \equiv 0 \pmod N$ and $|x_0| \leq X$.
12: end if
```

Algorithm 4 Speeding up Coppersmith's algorithm [6, 7] for finding small roots of univariate polynomial congruences

```
Input: An integer $N \geq 1$ and a univariate degree-$\delta$ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0, \ldots, N-1\}$.
Output: All $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq N^{1/\delta}$ and $f(x_0) \equiv 0 \mod N$.
1:  if $\delta = 1$ then
2:      Return all $x_0 \in \mathbb{Z}$ s.t. $x_0 + f(0) \equiv 0 \pmod N$ and $|x_0| \leq N$.
3:  else
4:      if $\delta + 1 \geq (\log N)/2$ then
5:          Compute all $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq N^{1/\delta}$ and $f(x_0) \equiv 0 \mod N$ by exhaustive search.
6:      else
7:          Let $h = \lceil (\log N)/\delta \rceil$, $n = h\delta$, $X = \left\lfloor N^{(h-1)/(n-1)} 2^{-1/2} n^{-1/(n-1)} \left( n^{3/2} \left(\frac{3c-2}{2c-2}\right)^{n-1} \lfloor c \rfloor^{-1} + 1 \right)^{-2/(n-1)} \right\rfloor$,
            $c = (3/2)^n$ and $t = \lfloor -N^{1/\delta} \rfloor + X$.
8:          while $t \leq N^{1/\delta}$ do
9:              Run Alg. 3 on $g(x) = f(x-t) \in \mathbb{Z}[x]$, $h$ and $N$.
10:             Output $x_0 + t$ for each root $x_0$ of $g(x)$ obtained.
11:             $t \leftarrow t + 2X$.
12:         end while
13:     end if
14: end if
```

which completes the induction. Hence:

$$\|S\|_\infty \leq 1 + \sum_{j=1}^{n-1} t(1+t)^{n-j-1} = 1 + t\sum_{j=0}^{n-2}(1+t)^j = (1+t)^{n-1}.$$

□

Corollary 3.5. *Let $B = (b_{i,j})$ be an $n \times n$ lower-triangular matrix over $\mathbb{Z}$ with strictly positive diagonal. Let $c > 1$. If $\tilde{B} = \lfloor cB / \min_{i=1}^n b_{i,i} \rfloor$, then:*

$$\|\tilde{B}^{-1}\|_\infty \leq \left(\frac{3c-2}{2c-2}\right)^{n-1} / \lfloor c \rfloor.$$

Proof. The matrix $\tilde{B}$ is lower-triangular like B. Because B is size-reduced, the entries of $\tilde{B}$ satisfy, for $1 \leq j < i \leq n$:

$$\frac{\tilde{b}_{i,j}}{\tilde{b}_{j,j}} < \frac{b_{i,j}/2^k}{b_{j,j}/2^k - 1} \leq \frac{1}{2} \times \frac{1}{1 - 2^k/b_{j,j}} \leq \frac{1}{2} \times \frac{1}{1 - 1/c}$$

This means that $\tilde{B}$ is almost size-reduced. Let Δ be the $n \times n$ diagonal matrix whose i-th diagonal entry is $1/\tilde{b}_{i,i}$. Then $T = \Delta\tilde{B}$ satisfies the conditions of Lemma 3.4 with $t = 1/(2(1-1/c))$, therefore:

$$\|T^{-1}\|_\infty \leq \left(\frac{1}{2} \times \frac{1}{1-1/c} + 1\right)^{n-1} = \left(\frac{3c-2}{2c-2}\right)^{n-1}.$$

Hence,

$$\|\tilde{B}^{-1}\|_\infty \leq \|T^{-1}\|_\infty \|\Delta\|_\infty \leq \left(\frac{3c-2}{2c-2}\right)^{n-1} \times \frac{1}{\min_{1 \leq i \leq n} \tilde{b}_{ii}} \leq \left(\frac{3c-2}{2c-2}\right)^{n-1} / \lfloor c \rfloor.$$

□

By combining Lemma 3.3 and Cor. 3.5, we obtain the following small-root bound X for Alg. 3:

Theorem 3.6. *Given as input two integers $N \geq 1$ and $h \geq 2$, a rational $c > 1$, and a univariate degree-δ monic polynomial $f(x) \in \mathbb{Z}[x]$ with coefficients in $\{0, \ldots, N-1\}$, Alg. 3 outputs all $x_0 \in \mathbb{Z}$ s.t. $|x_0| \leq X$ and $f(x_0) = 0 \bmod N$, where $X = \left\lfloor N^{(h-1)/(n-1)} 2^{-1/2} n^{-1/(n-1)} \left(n^{3/2}\left(\frac{3c-2}{2c-2}\right)^{n-1} \lfloor c \rfloor^{-1} + 1\right)^{-2/(n-1)} \right\rfloor$ and $n = h\delta$.*

Proof. By Lemma 3.3, we have:

$$0 < \|\mathbf{x}B\| < \left(n\|\tilde{B}^{-1}\|_2 + 1\right) 2^{\frac{n-1}{4}} \det(B)^{1/n},$$

where $\det(B)^{1/n} = N^{\frac{h-1}{2}} X^{\frac{n-1}{2}}$ and by Cor. 3.5,

$$\|\tilde{B}^{-1}\|_2 \leq \sqrt{n}\|\tilde{B}^{-1}\|_\infty \leq \sqrt{n}\left(\frac{3c-2}{2c-2}\right)^{n-1} / \lfloor c \rfloor.$$

Hence:

$$0 < \|\mathbf{x}B\| < \left(n^{3/2}\left(\frac{3c-2}{2c-2}\right)^{n-1} / \lfloor c \rfloor + 1\right) 2^{\frac{n-1}{4}} N^{\frac{h-1}{2}} X^{\frac{n-1}{2}}.$$

It follows that Lemma 2.2 is satisfied with $M = N^{h-1}$ and $v(xX)$ corresponding to $\mathbf{x}B$ if:

$$\left(n^{3/2}\left(\frac{3c-2}{2c-2}\right)^{n-1} / \lfloor c \rfloor + 1\right) 2^{\frac{n-1}{4}} N^{\frac{h-1}{2}} X^{\frac{n-1}{2}} \leq N^{h-1}/\sqrt{n},$$

which can be rewritten as

$$X \leq N^{(h-1)/(n-1)} 2^{-1/2} n^{-1/(n-1)} \left(n^{3/2} \left(\frac{3c-2}{2c-2} \right)^{n-1} \lfloor c \rfloor^{-1} + 1 \right)^{-2/(n-1)} .$$

□

The bound X of Th. 3.6 is never larger than that of Th. 2.3. However, if one selects $c \geq (3/2)^n$, then the two bounds are asymptotically equivalent. This is why Alg. 4 uses $c = (3/2)^n$.

3.2 Running time

The original matrix B had entries whose bit-size was $O(h \log N)$. Let β be the ratio between the maximal diagonal coefficient and the minimal diagonal coefficient of $\tilde{B}$:

$$\beta = \frac{N^h X^{\delta-1}}{X^{h\delta-\delta}} \tag{8}$$

If B is size-reduced, the entries of the new matrix $\tilde{B} = \lfloor cB/X^{h\delta-\delta} \rfloor$ are upper bounded by $c\beta$.

By Lemma 3.2. we know that if $h \geq 2$, then $\beta \geq N^{1-1/\delta}$, and if further $X \geq \Omega(N^{\frac{h-1}{n-1}})$ and $h\delta = O(\log N)$, then $\beta = N^{O(1)}$. Hence, the bit-size of $\tilde{B}$'s entries is $\leq \log c + O(\log N)$. And the dimension of $\tilde{B}$ is the same as B, *i.e.* $h\delta$. It follows that the running time of L^2 in Step 7 is $O(\delta^6 h^6 (\log c + \log N) + \delta^5 h^5 (\log c + \log N)^2)$ without fast integer arithmetic, which is $O(\delta^5 h^5 (\log c + \log N)^2$ because $\delta \leq \log N$.

Let ℓ be the maximal bit-size of the coefficients of $v(x) \in \mathbb{Z}[x]$ in Step 10: we know that $\ell \leq h \log N$, and the degree of $v(x)$ is $\leq n$. Then Step 10 can be performed in time $O(n^3(\ell + \log n)) = O(h \log^4 N) = O((\log^5 N)/\delta)$ using Schönhage's root isolation algorithm [19, Sec. 5.2]. Hence, the cost of Step 10 is less than Step 7. We note that in previous work on Coppersmith's method, different algorithms were proposed for Step 10, *e.g.* polynomial factorization, whose complexity upper bound might be higher than Step 7.

Hence, we proved the following speedup over Th. 2.3:

Theorem 3.7. *Alg. 3 runs in time $O(\delta^5 h^5 (\log c + \log N)^2)$ without fast integer arithmetic using the L^2 algorithm, or $O((h\delta)^{5+\varepsilon}(\log c + \log N) + (h\delta)^{\omega+1+\varepsilon}(\log c + \log N)^{1+\varepsilon})$ for any $\varepsilon > 0$ using fast integer arithmetic and $\tilde{L}^1$ in Step. 7, where $\omega \leq 2.376$ is the matrix multiplication complexity constant.*

Our main result (Th. 3.1), a variant of Coppersmith's algorithm (Corollary 2.4) with improved complexity upper bound, is then a simple corollary of Th. 3.7. More precisely, we proved that Alg. 1 called Alg. 2 at most a constant times: similarly, one can easily prove that Alg. 4 calls Alg. 3 at most a constant times. Indeed, when $c = (3/2)^n$, then $\left(n^{3/2} \left(\frac{3c-2}{2c-2} \right)^{n-1} \lfloor c \rfloor^{-1} + 1 \right)^{-2/(n-1)}$ converges to 1. This means that the bound X achieved by Th. 3.6 is asymptotically equivalent to the one achieved by Th. 2.3, which completes the proof of Th. 3.1, because $\log c = O(\log N)$ when $c = (3/2)^n$.

3.3 Experiments

We implemented Coppersmith's algorithm and our rounding improvement using Shoup's NTL library [20]. However, for the LLL reduction, we used the fplll implementation [3] by Cadé *et al.*, which includes the L^2 algorithm [16]: fplll is much faster than NTL for Coppersmith's matrices. It should be stressed that fplll is a wrapper which actually implements several variants of LLL, together with several heuristics: L^2 is only used as a last resort when heuristic variants fail. This means that there might be a discrepancy between the practical running time and the theoretical complexity upper bound of LLL routines.

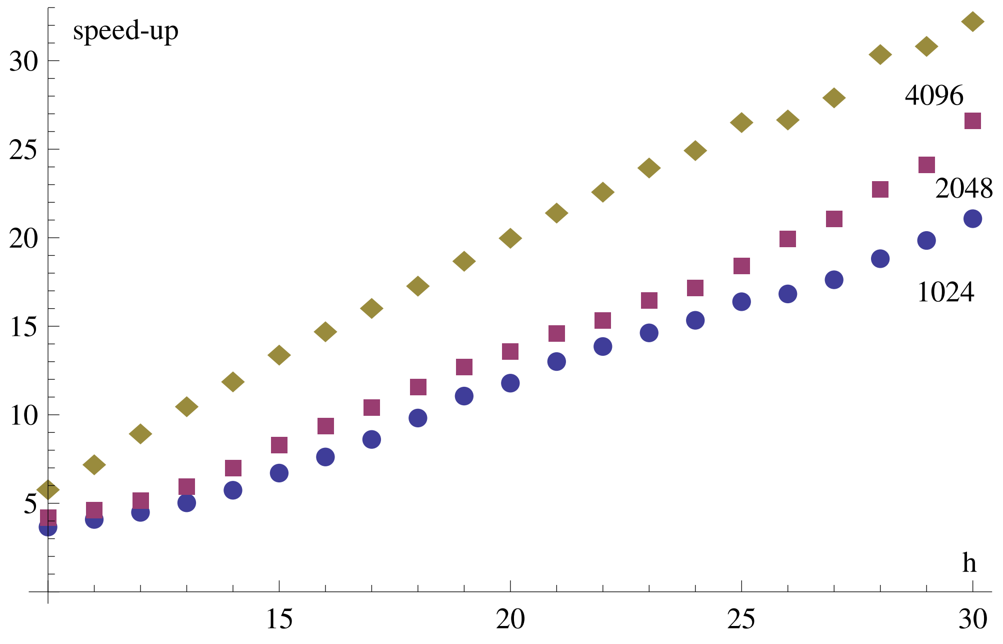
Our test machine is a 2.93-GHz Intel Core 2 Duo processor E7500 running on Fedora. Running times are given in seconds.

Table 1: Bounds and running time for cubic congruences

Size of N	Data type	Parameter h				
		10	15	20	25	30
1024	Size of X	318	324	328	331	332
	$T_{original}$	2.54	30.48	216.27	793.38	3720.81
	$T_{rounded}$	0.68	4.49	18.22	48.17	175.86
	Speed-up	3.74	6.79	11.87	16.47	21.16
2048	Size of X	634	650	658	663	666
	$T_{original}$	13.47	150.7	865.67	3078.01	10146.71
	$T_{rounded}$	3.14	17.79	63.3	166.36	379.8
	Speed-up	4.29	8.40	13.67	18.50	26.72
4096	Size of X	1270	1302	1318	1327	1333
	$T_{original}$	41.45	582.58	3161.99	11967.8	42053.4
	$T_{rounded}$	7.07	43.25	157.54	449.81	1301.51
	Speed-up	5.86	13.47	20.07	26.61	32.31

Like in [9], we used the case $\delta = 3$, and N an RSA-type modulus: the exact polynomial congruence is derived from RSA encryption with public exponent δ. Then $n = 3h$ and Alg. 2 of Th. 2.3 can find all the roots x_0 as long as $|x_0| \leq X = \lfloor 2^{-1/2} N^{\frac{h-1}{n-1}} n^{-\frac{1}{n-1}} \rfloor$. For a fixed h, the rounding strategy (Alg. 3) gives a worse bound than X, but the difference can be made arbitrarily small by increasing the parameter c: in our experiments, we therefore chose the smallest value of c such that $\left(n^{3/2} \left(\frac{3c-2}{2c-2} \right)^{n-1} \lfloor c \rfloor^{-1} + 1 \right)^{-2/(n-1)} \geq 0.90$, so that the new bound is never less than the old bound X by more than 10%, which is essentially the same. However, we note that our choice of c is pessimistic in practice: our theoretical analysis was a worst-case analysis, and the constants are better in practice. For instance, it has been proved in [22] that if T is a random $n \times n$ lower-triangular matrix with unit diagonal and subdiagonal coefficients normally distributed, then $(\||T^{-1}\|_2)^{1/n}$ converges to $1.3057\ldots$ And experimentally, if T is a random $n \times n$ lower-triangular matrix with unit diagonal and subdiagonal coefficients uniformly distributed over $[-1/2, +1/2]$, then $(\||T^{-1}\|_\infty)^{1/n}$ is with high probability less than 1.1. This means that the constants of Lemma 3.4 (and therefore the implicit $3/2$ in the formula for c) are pessimistic in practice.

Table 1 summarizes our limited experiments comparing Algs. 2 and 3 in practice: it provides the bit-length of X and the corresponding running times of Algs. 2 and 3. The running time only measures the lattice reduction time, because the cost of solving a univariate equation over $\mathbb{Z}$ turns out to be much less in practice. Running times are given as averages over 5 samples.



We see that we already get significant speedups (say, larger than 10) even for small values of h and typical sizes of N. The speedup grows when $\log N$ or h grows: for fixed N, the speedup grows roughly a bit less than quadratically in h, whereas the theoretical analysis gives a speedup quadratic in h. Hence, our improvement is practical and allows to get much closer to the asymptotical small-root bound.

4 Other Small-Root Algorithms

We now discuss whether our rounding method can similarly speed up other small-root algorithms (see the surveys [13, 14]), which are based on the same main ideas where LLL reduction plays a crucial role. In theory, the rounding method provides a speedup for any triangular matrix whose diagonal coefficients are all large. However, in order to have a large speedup, we need the minimal diagonal coefficient to be much larger than the ratio between the maximal diagonal coefficient and the minimal diagonal coefficient. In Coppersmith's algorithm, the smallest diagonal coefficient was $N^{h-O(1)}$, while the gap was $N^{O(1)}$, which translated into a polynomial speedup. It turns out that other small-root algorithms do not share the same features: we only get a (small) constant speedup. We leave it as an open problem to obtain polynomial (non-constant) speedups for these other small-root algorithms: this might be useful to make practical attacks on certain fully-homomorphic encryption schemes (see [4]).

4.1 Gcd Generalization

Coppersmith's algorithm (Alg. 1) has been generalized by essentially Howgrave-Graham [11] and Boneh *et al.* [2] (see the surveys [13, 14]) as follows:

Theorem 4.1. *There is an algorithm which, given as input an integer N of unknown factorization, a rational α s.t. $0 < \alpha \leq 1$ and a monic polynomial $f(x) \in \mathbb{Z}[x]$ of degree δ and coefficients in $\{0, \dots, N-1\}$, outputs all integers $x_0 \in \mathbb{Z}$ such that $\gcd(f(x_0), N) \geq N^{\alpha}$ and $|x_0| \leq N^{\alpha^2/\delta}$ in time polynomial in $\log N$, δ and the bit-size of α.*

Th. 2.1 is then the special case $\alpha = 1$ of Th. 4.1. The algorithm underlying Th. 4.1 is in fact very similar to Alg. 2: instead of applying Lemma 2.2 with $M = N^{h-1}$, one uses $M = p^{h-1}$ where $p \geq N^{\alpha}$ is some unknown divisor of N. And one considers the same family of polynomials $g_{i,j}(x) = x^j N^{h-1-i} f^i(x)$ but over different indices. Alg. 2 used $0 \leq i < h$ and $0 \leq j < \delta$. This time, we use two sets of indices: one with $0 \leq i < h-1$ and $0 \leq j < \delta$, and another with $i = h-1$ and $0 \leq j < \gamma$, where γ is chosen asymptotically to be $\lfloor \delta(h-1)(1/\alpha - 1) \rfloor$. Then the dimension is $n = (h-1)\delta + \gamma$. The maximal diagonal coefficient is still $N^{h-1}X^{\delta-1}$, and the minimal diagonal coefficient is still

$X^{h\delta-\delta}$, like in Lemma 3.2. However, the balance between these two coefficients has changed, because the bound X is much smaller than in Coppersmith's algorithm. Before, X was essentially $N^{(h-1)/(n-1)}$ whose order of magnitude is the same as $N^{1/\delta}$, but now, it is close to $N^{\alpha^2/\delta}$, so $X^{h\delta-\delta}$ is close to $N^{(h-1)\alpha^2}$. In other words, the ratio between the maximal and minimal diagonal coefficient is about $N^{(1-\alpha^2)(h-1}$ which is no longer $N^{O(1)}$. We are thus trading an LLL reduction of a matrix with bit-size $\approx (h-1)\log N$, with one with bit-size $\approx (1-\alpha^2)(h-1)\log N$, which can only provide a small constant speedup at best, namely $1/(1-\alpha^2)^2$ for L^2 or close to $1/(1-\alpha^2)$ for $\tilde{L}^1$. In the gcd generalization, the input basis is much less reduced than in Coppersmith's algorithm.

4.2 Multivariate Equations

Coppersmith [6, 7] showed that his algorithm for finding small roots of univariate polynomial congruences can heuristically be extended to multivariate polynomial congruences: the most famous example is the Boneh-Durfee attack [1] on RSA with short secret exponent.

Not all these multivariate variants use triangular matrices, though they sometimes can be tweaked: some rely on lattices which are not full-rank, including the Boneh-Durfee attack [1]. However, when the matrix is triangular, there is a similar problem than for the gcd generalization: the diagonal coefficients are much more unbalanced than in the univariate congruence case, which means that the speedup of the rounding method is at most a small constant. And in the Boneh-Durfee attack, the coefficients which play the role of the diagonal coefficients are also unbalanced.

For instance, assume that one would like to find all small roots of $f(x,y) \equiv 0 \pmod{N}$ with $|x| \leq X$ and $|y| \leq Y$, where $f(x,y)$ has total degree δ and has at least one monic monomial $x^\alpha y^{\delta-\alpha}$ of maximal total degree. Then, for a given parameter h, the lower-triangular matrix has dimension $n = (h\delta+1)(h\delta+2)/2$ and diagonal coefficients $N^{h-v}X^{u_1+v\delta}Y^{u_2+v(\delta-\alpha)}$, where $u_1+u_2+\delta v \leq h\delta$ and $u_1, u_2, v \geq 0$ with $u_1 < \alpha$ or $u_2 < \delta - \alpha$. For typical choices of X and Y such that $XY < N^{1/\delta-\varepsilon}$, the ratio between the largest and smallest diagonal coefficient is no longer $N^{O(1)}$.

Acknowledgements

During the preparation of this paper, J. Bi was supported in part by China's 973 Program, Grant 2013CB834205, and NSFC Grants 61133013 and 61272035, and P. Q. Nguyen was supported in part by China's 973 Program, Grant 2013CB834205, and NSFC's Key Project, Grant 61133013.

References

[1] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, 46(4):1339, 2000.

[2] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $n = p^r q$ for large r. In *Advances in Cryptology - Proc. CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 1999.

[3] D. Cadé, X. Pujol, and D. Stehlé. FPLLL library, version 3.0. Available from `http://perso.ens-lyon.fr/damien.stehle`, Sep 2008.

[4] H. Cohn and N. Heninger. Approximate common divisors via lattices. *IACR Cryptology ePrint Archive*, 2011:437, 2011.

[5] D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *Advances in Cryptology - Proc. EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996.

[6] D. Coppersmith. Finding a small root of a univariate modular equation. In *Advances in Cryptology - Proc. EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1996.

[7] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997. Journal version of [6, 5].

[8] J.-S. Coron. Finding small roots of bivariate integer polynomial equations: A direct approach. In *Advances in Cryptology – Proc. CRYPTO '07*, volume 4622 of *Lecture Notes in Computer Science*, pages 379–394. Springer, 2007.

[9] C. Coupé, P. Q. Nguyen, and J. Stern. The effectiveness of lattice attacks against low-exponent RSA. In *Public Key Cryptography – Proc. PKC '99*, volume 1560 of *Lecture Notes in Computer Science*, pages 204–218. Springer, 1999.

[10] N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Cryptography and Coding – Proc. IMA '97*, volume 1355 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 1997.

[11] N. Howgrave-Graham. Approximate integer common divisors. In *Proc. CaLC '01*, volume 2146 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2001.

[12] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Ann.*, 261:513–534, 1982.

[13] A. May. Using LLL-reduction for solving RSA and factorization problems: A survey. 2010. In [17].

[14] P. Q. Nguyen. Public-key cryptanalysis. In I. Luengo, editor, *Recent Trends in Cryptography*, volume 477 of *Contemporary Mathematics*. AMS–RSME, 2009.

[15] P. Q. Nguyen and D. Stehlé. LLL on the average. In *Algorithmic Number Theory – Proc. ANTS*, LNCS, pages 238–256. Springer, 2006.

[16] P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. of Computing*, 39(3):874–903, 2009.

[17] P. Q. Nguyen and B. Vallée, editors. *The LLL Algorithm: Survey and Applications*. Information Security and Cryptography. Springer, 2010.

[18] A. Novocin, D. Stehlé, and G. Villard. An LLL-reduction algorithm with quasi-linear time complexity: extended abstract. In *Proc. STOC '11*, pages 403–412. ACM, 2011.

[19] A. Schönhage. *The fundamental theorem of algebra in terms of computational complexity - preliminary report*. Universität Tübingen, 1982.

[20] V. Shoup. Number Theory C++ Library (NTL) version 5.4.1. Available at `http://www.shoup.net/ntl/`.

[21] V. Shoup. OAEP reconsidered. *J. Cryptology*, 15(4):223–249, 2002.

[22] D. Viswanath and L. N. Trefethen. Condition numbers of random triangular matrices. *SIAM J. Matrix Anal. Appl.*, 19(2):564–581 (electronic), 1998.