Sieving for Shortest Vectors in Ideal Lattices

Michael Schneider

Technische Universität Darmstadt, Germany
mischnei@cdc.informatik.tu-darmstadt.de

Abstract. Lattice based cryptography is gaining more and more importance in the cryptographic community. It is a common approach to use a special class of lattices, so-called ideal lattices, as the basis of lattice based crypto systems. This speeds up computations and saves storage space for cryptographic keys. The most important underlying hard problem is the shortest vector problem. So far there is no algorithm known that solves the shortest vector problem in ideal lattices faster than in regular lattices. Therefore, crypto systems using ideal lattices are considered to be as secure as their regular counterparts.
In this paper we present IdealListSieve, a variant of the ListSieve algorithm, that is a randomized, exponential time sieving algorithm solving the shortest vector problem in lattices. Our variant makes use of the special structure of ideal lattices. We show that it is indeed possible to find a shortest vector in ideal lattices faster than in regular lattices without special structure. The practical speedup of our algorithm is linear in the degree of the field polynomial. We also propose an ideal lattice variant of the heuristic GaussSieve algorithm that allows for the same speedup.

Keywords. shortest vector problem, sieving algorithms, ideal lattices

1 Introduction

Lattices are discrete additive subgroups of $\mathbb{R}^m$. Their elements can be considered to be vectors in the Euclidean vector space. One of the most important computational problems in lattices is the shortest vector problem (SVP). Roughly speaking, given a representation of a lattice, it asks to output a shortest non-zero element of the lattice.

In 2001, Ajtai, Kumar, and Sivakumar presented a randomized algorithm to solve the shortest vector problem in lattices. Unfortunately, the space requirement of the AKS *sieving algorithm* is exponential in the lattice dimension, and therefore the algorithm was not practical. In 2010, Micciancio and Voulgaris presented two variants of this sieving algorithm that still require exponential storage, but with much smaller constants. Their algorithm is the first sieving approach considered to be competitive to enumeration algorithms that only require polynomial space.

Lattices are widely used in cryptography. There, in order to save storage for cryptographic keys, it is common to use structured lattices. The NTRU crypto system [HPS98] for example uses so-called cyclic lattices, where for each lattice vector $\mathbf{v}$, its rotations $\mathsf{rot}(\mathbf{v})$ consisting of the rotated lattice vectors are also elements of the lattice. Lattice of more general structure are used in the signature schemes [LM08,Lyu09], for the encryption systems [SSTX09,LPR10], the SWIFFTX hash function family [LMPR08,ADL+08], or the fully homomorphic encryption scheme of [Gen09], for example. They were called ideal lattices in [LM06]. The theory of ideal lattices is based on the work of [PR06,LM06]. Cyclic lattices are a special case of ideal lattices.

Micciancio proved a worst-case to average-case reduction for ideal lattices in [Mic07], where he showed that inverting his one-way function is as hard as solving ideal lattice problems in the worst case. Lattice problems are even considered unbroken in the presence of quantum computers, and so far they withstand sub-exponential attacks. Thus cryptographic schemes based in hard problems in ideal lattices are good candidates for future security requirements.

So far there is no SVP algorithm making use of the special structure of ideal lattices. It is widely believed that solving SVP (and all other lattice problems) in ideal lattices is as hard as in regular lattices. Our intention is to show how sieving algorithms can be strengthened in ideal lattices using their circular structure. The idea was already presented in [MV10b]. There, the authors assume that the amount of storage required by their algorithm decreases with a factor of n, where n is the degree of the field polynomial. We show that even more not only the storage but as well the running time of sieving algorithms decreases by a factor of n.

Related Work. There are basically three approaches to solve the SVP in lattices: Voronoi-cell based algorithms, enumeration algorithms, and probabilistic sieving algorithms. The Voronoi cell based algorithms were presented in [MV10a]. It is the first deterministic single exponential algorithm for the shortest vector problem. So far, this type of algorithms is more theoretical work. Enumeration algorithms solve the SVP deterministically in asymptotic time $2^{\mathcal{O}(m \log(m))}$, where m is the lattice dimension [HS07,PS08]. They perform exhaustive search by exploring all lattice vectors of a bounded search region. Enumeration algorithms can be rendered probabilistic using an extreme pruning strategy [GNR10], which allows for an exponential speedup and makes enumeration the fastest algorithm for solving SVP in practice. Enumeration algorithms only require polynomial space.

Sieving algorithms were first presented by Ajtai, Kumar, and Sivakumar [AKS01] in 2001. The runtime and space requirements were proven to be in $2^{\mathcal{O}(m)}$. Nguyen and Vidick [NV08] carefully analyzed this algorithm and presented the first competitive timings and results. They show that the runtime of AKS sieve is $2^{5.90m+o(m)}$ and the space required is $2^{2.95m+o(m)}$. The authors also presented a heuristic variant of AKS sieve without perturbations. Their runtime is $2^{0.41m+o(m)}$ and they require space $2^{0.21m+o(m)}$. In 2010, Micciancio and Voulgaris [MV10b] presented a provable sieving variant called ListSieve and a more practical, heuristic variant called GaussSieve. ListSieve runs in time $2^{3.199m+o(m)}$ and requires space $2^{1.325m+o(m)}$. For GaussSieve, the maximum list size can be bounded by the kissing number τ_m, whereas, due to collisions, a runtime bound can not be proven. The practical runtime is $2^{0.52m}$, the space requirements is expected to be less than $2^{0.21m}$ and turns out to be even smaller in practice.

Pujol and Stehlé [PS09] improve the theoretical bounds of ListSieve [MV10b] using the birthday paradox to runtime $2^{2.465m+o(m)}$ and space $2^{1.233m+o(m)}$. Wang et at. [WLTB10] present a heuristic variant of the Nguyen-Vidick sieve running in $2^{0.3836m+o(m)}$ with space complexity of $2^{0.2557m}$. The work of [BN09] deals with all ℓ_p norms, generalizing the AKS sieve. There is only one public implementation of a sieving algorithm, namely `gsieve` [Vou10], which implements the GaussSieve algorithm of [MV10b].

Using heuristics like extreme pruning [GNR10], enumeration algorithms outperform sieving algorithms again, as the SVP challenge [GS10] shows. We hope that it is possible to integrate heuristics such as extreme pruning to sieving algorithms, which would make them competitive to enumeration techniques again.

Our Contribution. Micciancio and Voulgaris already mention the possibility to speed up the sieving for ideal lattices [MV10b]. They propose to use the cyclic rotations of each sampled vector to reduce the size of the vectors. For ideal lattices, the "rotation" of each lattice vector is still an element of the lattice. Therefore, it can be used in the sieving process. The authors of [MV10b] expect a reduction of the list size linear in the degree of the field polynomial, and a substantial impact on the practical behaviour of the Gauss sieving algorithm. In this paper, we present experimental results using this approach. We implement ListSieve and IdealListSieve without perturbations. Our experiments show that indeed the storage requirements decrease as expected. But even more, sieving in

ideal lattices can find a shortest lattice vector much faster, with a speedup factor linear in the degree of the field polynomial in practice. To explain the results, we use the assumption that the number of reductions used in the sieving process stays the same in both the original and the ideal cases. We will show that this assumptions conforms with our experiments.

To give an example, the measured and fitted runtime of IdealListSieve in cyclic lattices is $2^{0.52m-21.9}$ seconds, compared to $2^{0.67m-26.3}$ seconds for ListSieve. In dimension $m = 60$, the runtime difference is about 4 hours, which corresponds to a time advantage of 95%. The worst-case runtime of IdealListSieve remains the same as for ListSieve, since considering all rotations cancels out the factor n in theory.

To our knowledge, this is the first SVP or lattice reduction algorithm that can use the special structure of ideal lattices. It is often stated that solving problems in ideal lattices is as hard as in the general case, among others in [MR08,ADL+08,Lyu09]. Since the runtime of sieving algorithms is exponential, this linear speedup does not effect the asymptotic runtime of sieving algorithms. It only helps to speed up sieving in ideal lattices in practice noticeably. For the fully homomorphic encryption challenges for example n is bigger than 2^{10}, which would result in a speedup of more than 1000 for sieving. [Lyu09] uses $n \geq 512$. These numbers show that even if we only allow for a linear speedup this still might give huge speedups in practice.

Organization of this Paper. In Section 2 we present some basic notation and facts on (ideal) lattices. In Section 3 we show the IdealListSieve algorithm. In Section 4 we give some theoretical analysis, and Section 5 shows experimental results of our implementation. Finally we give a conclusion, including some ideas for enumeration in ideal lattices.

2 Preliminaries

Define the index set $[n] = \{0, 1, \ldots, n-1\}$. $\mathbf{I}_m$ is the identity matrix in dimension m, $\mathbf{0}_m$ and $\mathbf{1}_m$ are m-dimensional column vectors consisting of 0 and 1 entries only. The scalar product of two vector elements $\mathbf{x}$ and $\mathbf{y}$ is written $\langle \mathbf{x} \mid \mathbf{y} \rangle$. Throughout this paper, n denotes the degree of polynomials and m is the dimension of lattices.

Let $m, d \in \mathbb{N}$, $m \leq d$, and let $\mathbf{b}_1, \ldots, \mathbf{b}_m \in \mathbb{R}^d$ be linearly independent. Then the set $\mathcal{L}(\mathbf{B}) = \{\sum_{i=1}^{m} x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$ is the lattice spanned by the basis column matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_m] \in \mathbb{Z}^{d \times m}$. $\mathcal{L}(\mathbf{B})$ is called m-dimensional. The basis $\mathbf{B}$ is not unique, unimodular transformations

lead to a different basis of the same lattice. The first successive minimum $\lambda_1(\mathcal{L}(\mathbf{B}))$ is the length of a shortest vector of $\mathcal{L}(\mathbf{B})$. The (search) shortest vector problem (SVP) is formulated the following. Given a basis $\mathbf{B}$, compute a vector $\mathbf{v} \in \mathcal{L}(\mathbf{B}) \setminus \{\mathbf{0}\}$ subject to $\|\mathbf{v}\| = \lambda_1(\mathcal{L})$. It can be formulated in every norm, we will consider the Euclidean norm ($\|\cdot\| = \|\cdot\|_2$) in the remainder of this paper.

The lattice determinant $\det(\mathcal{L}(\mathbf{B}))$ is defined as $\sqrt{\det(\mathbf{B}^t\mathbf{B})}$. It is invariant under basis changes. For full-dimensional lattices, where $m = d$, there is $\det(\mathcal{L}(\mathbf{B})) = |\det(\mathbf{B})|$ for every basis $\mathbf{B}$. In the remainder of this paper we will only be concerned with full-dimensional lattices ($m = d$).

Ideal Lattices. Ideal lattices are lattices with special structure. Let $\mathbf{f} = x^n + \mathbf{f}_n x^{n-1} + \ldots + \mathbf{f}_1 \in \mathbb{Z}[x]$ be a monic polynomial of degree n, and consider the ring $\mathbf{R} = \mathbb{Z}[x]/\langle \mathbf{f}(x) \rangle$. Elements in $\mathbf{R}$ are polynomials of maximum degree $n-1$. If there exists an ideal $\mathbf{I} \subseteq \mathbf{R}$, each element $\mathbf{v} = \sum_{i=1}^{n} \mathbf{v}_i x^{i-1} \in \mathbf{I}$ naturally corresponds to its coefficient vector $(\mathbf{v}_1, \ldots, \mathbf{v}_n) \in \mathbb{Z}^n$. Since ideals are additive subgroups, the set of all coefficient vectors corresponding to the ideal $\mathbf{I}$ forms a so-called *ideal lattice*. For the sake of simplicity we can switch between the vector and the ideal notations and use the one that is more suitable in each case.

For each $\mathbf{v} \in \mathbf{R}$, the elements $x^i \cdot \mathbf{v}$ for $i \in [n]$ form a basis of an ideal lattice. We call this multiplication with x a *rotation*, since for special polynomials the vector $x \cdot \mathbf{v}$ consists of the rotated coefficients of $\mathbf{v}$. In vector notation, the multiplication of an element with x can be performed by multiplication with the matrix

$$\mathsf{rot} = \left(\begin{array}{c|c} \mathbf{0}_{n-1} & -\bar{\mathbf{f}} \\ \mathbf{I}_{n-1} & \end{array} \right),$$

where $\bar{\mathbf{f}}$ consists of the coefficients of the field polynomial $\mathbf{f}$. If $\mathbf{f} \in \mathbf{R}$ is a monic, irreducible polynomial of degree n, then for any element $\mathbf{v} \in \mathbf{R}$, the elements $\mathbf{v}, \mathbf{v}x, \ldots, \mathbf{v}x^{n-1}$ are linearly independent (see for example the proof of Lemma 2.12 in [Lyu08]). For $\mathbf{f}(x) = x^n - 1$, which is not irreducible over $\mathbb{Z}$, it is easy to see that the vectors $\mathbf{v}x^i$ are also linearly independent, unless the vector has very special form.

The row matrices of the bases used in practice are of the form

$$\begin{pmatrix} q\mathbf{I}_n & \mathbf{0} \\ (\mathsf{rot}^i(\mathbf{v}))_{i \in [n]} & \mathbf{I}_n \end{pmatrix}. \tag{1}$$

Here the lower part consists of the n rotations of $\mathbf{v}$, which corresponds to the multiplications of the ring element $\mathbf{v}$ with x^i for $i \in [n]$. The upper

part is necessary to make sure that every element in the lattice can be reduced modulo q. Bases for higher dimensional lattices can be generated using multiple points $\mathbf{v}_i$ and their rotations. The dimension m of the lattice is then a multiple of the field polynomial degree n.

The usage of ideal lattices reduces the storage amount for a basis matrix from nm elements to m elements, because every block of the basis matrix is determined by its first row. In addition, for an ideal basis $\mathbf{B}$, the computation $\mathbf{B} \cdot \mathbf{y}$ can be sped up using Fast Fourier transformation from $\mathcal{O}(mn)$ to $\tilde{\mathcal{O}}(m)$.

In this paper we are concerned with three types of ideal lattices, defined by the choice of $\mathbf{f}$:

- *Cyclic lattices*: Let $\mathbf{f}_1(x) = x^n - 1$, i.e., $\overline{\mathbf{f}} = (-1, 0, \ldots, 0)$. We call the ideal lattices of the ring $\mathbf{R}_1 = \mathbb{Z}[x]/\langle \mathbf{f}_1(x) \rangle$ cyclic lattices. $\mathbf{f}_1(x)$ is never irreducible over $\mathbb{Z}$ ($x - 1$ is always a divisor), therefore cyclic lattices can not guarantee worst-case security. The rotation of $\mathbf{v}$ is $\mathsf{rot}(\mathbf{v}) = (\mathbf{v}_{n-1}, \mathbf{v}_0, \ldots, \mathbf{v}_{n-2})$.
- *Anti-cyclic lattices*: Let $\mathbf{f}_2(x) = x^n + 1$, i.e., $\overline{\mathbf{f}} = (1, 0, \ldots, 0)$. We call the ideal lattices of the ring $\mathbf{R}_2 = \mathbb{Z}[x]/\langle \mathbf{f}_2(x) \rangle$ anti-cyclic lattices. $\mathbf{f}_2(x)$ is irreducible over $\mathbb{Z}$ if n is a power of 2. The rotation of $\mathbf{v}$ is $\mathsf{rot}(\mathbf{v}) = (-\mathbf{v}_{n-1}, \mathbf{v}_0, \ldots, \mathbf{v}_{n-2})$. Anti-cyclic lattices are the ones used most in cryptography.
- *Prime cyclotomic lattices*: Let $\mathbf{f}_3(x) = x^n + x^{n-1} + \ldots + 1$, i.e., $\overline{\mathbf{f}} = (1, \ldots, 1)$. We call the ideal lattices of the ring $\mathbf{R}_3 = \mathbb{Z}[x]/\langle \mathbf{f}_3(x) \rangle$ prime cyclotomic lattices. $\mathbf{f}_3(x)$ is irreducible over $\mathbb{Z}$ if $n + 1$ is prime. The rotation of $\mathbf{v}$ is $\mathsf{rot}(\mathbf{v}) = (-\mathbf{v}_{n-1}, \mathbf{v}_0 - \mathbf{v}_{n-1}, \ldots, \mathbf{v}_{n-2} - \mathbf{v}_{n-1})$. We only consider cyclotomic polynomials of degree n where $n + 1$ is prime.[1]

A nice and more detailed overview about ideal lattices is shown in [Lyu08].

3 IdealListSieve Algorithm

In this section we will present the IdealSieve algorithm of [MV10b] and introduce the ideal lattice variant IdealListSieve. More details about the implementation will follow in Section 5.

ListSieve. Recall that the goal is to solve the shortest vector problem, i.e., given a basis $\mathbf{B}$ of a lattice find a non-zero vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ with

[1] Other cyclotomic polynomials, where $n + 1$ is not prime, have different structure, the rotations are hard to implement, and they are seldom used in practice.

norm equal to $\lambda_1(\mathcal{L}(\mathbf{B}))$. The idea of ListSieve is the following. A list L of lattice vectors is stored. In each iteration of the algorithm, a new random vector $\mathbf{p}$ is sampled uniformly at random from a set of bounded vectors. This vector $\mathbf{p}$ is then reduced using the list L in the following manner. If a list vector $\mathbf{l} \in L$ can be subtracted from $\mathbf{p}$ lowering its norm more than a factor $\delta < 1$, $\mathbf{p}$ is replaced by $\mathbf{p} - \mathbf{l}$. With this, $\mathbf{p}$ gets smaller every time. When the vector has passed the list it is appended to L. In the end, L will contain a vector of minimal length with high probability. When the sampled vector $\mathbf{p}$ is linearly independent of list vectors it will be reduced to $\mathbf{0}$ and not be appended. This rare case is called a *collision*. Collisions are important for runtime proofs (they avert a runtime proof for GaussSieve, for example). For practical issues, they are negligible, since they occur very seldom. Algorithm 1 shows a pseudo-code of ListSieve without perturbations.

Algorithm 1: `ListSieve(`$\mathbf{B}$`,` targetNorm`)`

```
1 List L ← LLL(B)                                  ▷ Pre-reduction with the LLL algorithm
2 while currentBestNorm > targetNorm do
3 |   p ← sampleRandomLatticeVector(B)              ▷ Sampling step
4 |   ListReduce(p, L, δ = 1 − 1/m)                 ▷ Reduction step
5 |   if p ≠ 0 then
6 |   |   L.append(p)                               ▷ Append step
7 |   end
8 end
9 return l_best
```

Originally, ListSieve does not work with lattice points $\mathbf{p}$, but with a perturbed point $\mathbf{p} + \mathbf{e}$ with a small error $\mathbf{e}$. The use of perturbations is necessary in order to upper bound the probability of collisions, which is essential for proving runtime bounds for the algorithm. Since in practice collisions play a minor role we will skip perturbations in our implementation. For the sampling of random vectors in Line 3 [MV10b] use Kleins randomized rounding algorithm [Kle00], which we will also apply for all our implementations.

IdealListSieve. One of the properties of ideal lattices is that for each lattice vector $\mathbf{v}$, rotations of this vector are also contained in the lattice. This is due to the property of the ideal $\mathbf{I}$ corresponding to the ideal lattice. Ideals in $\mathbf{R}$ are closed under multiplication with elements from $\mathbf{R}$, and since vectors in ideal lattices are the same as elements of the ideal, multiplications of these vectors are also elements of the lattice.

To compute the rotation of a vector $\mathbf{v}$ one has to rotate each block of length n of $\mathbf{v}$. If $m = 2n$, the first half of $\mathbf{v}$ (which belongs to the $q\mathbf{I}_n$ part in the first rows of the basis matrix (1)) is rotated and so is the second half. So when ListSieve tries to reduce the sample $\mathbf{p}$ with a vector $\mathbf{l} = (\mathbf{l}_1, \ldots, \mathbf{l}_n, \mathbf{l}_{n+1}, \ldots, \mathbf{l}_m)$, we can also use the vectors

$$\mathbf{l}^{(j)} = (\mathsf{rot}^j((\mathbf{l}_1, \ldots, \mathbf{l}_n)), \mathsf{rot}^j((\mathbf{l}_{n+1}, \ldots, \mathbf{l}_m))), \text{ for } j = 1 \ldots n-1\,,$$

where the first and the second half of the vector is rotated. Therefore, the sample $\mathbf{p}$ can be more reduced in each iteration. Instead of reducing with one single vector $\mathbf{l}$ per entry in the list L, n vectors can be used.

Function `IdealListReduce` shows a pseudo-code of the function that is responsible for the reduction part. Compared to the ListSieve algorithm of [MV10b], this function replaces the `ListReduce` function. Unfortunately, only the case where m is a multiple of n allows the usage of rotations of lattice vectors. For the case where $n \nmid m$, it is not possible to apply the rotation to the last block of a lattice vector $\mathbf{v}$.

Func. `ListReduce`$(\mathbf{p}, L, \delta)$

```
1 while (∃l' ∈ L : ‖p − l'‖ ≤ δ ‖p‖) do
2 |   p ← p − round(⟨p | l'⟩ / ⟨l' | l'⟩) · l'
3 end
4 return p
```

Func. `IdealListReduce`$(\mathbf{p}, L, \delta)$

```
1 while (∃j ∈ [n], l ∈ L, l' = rot^j(l) :
    ‖p − l'‖ ≤ δ ‖p‖) do
2 |   p ← p − round(⟨p | l'⟩ / ⟨l' | l'⟩) · l'
3 end
4 return p
```

The while loop condition in Line 1 introduces the rotation step (for all types of ideal lattices). The reduction step in Line 2 differs from the original ListSieve description in [MV10b]. It uses the reduction step known from the Gauss (respectively Lagrange) algorithm (an orthogonal projection), that is also used in the LLL algorithm [LLL82]. The step is not explained in [MV10b], whereas their implementation [Vou10] already uses this improvement. The slackness parameter $\delta = 1 - 1/m$ is used to ensure that the norm decrease is sufficient for each reduction in order to guarantee polynomial runtime in the list size.

IdealGaussSieve. For ListSieve, when a vector joined the list once it remains unchanged forever. GaussSieve introduces the possibility to remove vectors from the list if they can be more reduced by a newly sampled vector. The reduction process is twofold in GaussSieve. First, the new vector $\mathbf{p}$ is reduced as in ListSieve. Second, all list vectors are reduced using $\mathbf{p}$. If a vector from the list is shortened, it will leave the list and pass it again in one of the next iterations. Therefore the list will consist of less and shorter vectors than in the ListSieve case.

It is easy to include the rotations into GaussSieve in the same manner as for ListSieve. We can replace the function `GaussReduce` of [MV10b] by a new function `IdealGaussReduce`, which uses the rotations twice. First it is used for the reduction of $\mathbf{p}$, second for the reduction of list vectors. The rest of GaussSieve remains unchanged. IdealGaussSieve is also included in our implementation.

4 Predicted Advantage of IdealListSieve

In this section we theoretically analyze the IdealSieve algorithm and try to predict the results concerning number of iterations I, the total number of reductions R, and the maximum size L of the list L. For comparison of an algorithm and its ideal lattice variant we will always use the quotient of a characteristic of the non-ideal variant divided by the ideal variant. We will always denote it with *speedup*. For example, the speedup in terms of reductions is $\mathsf{R}_{orig}/\mathsf{R}_{ideal}$.

Recall that the only change we made in the algorithm is that in the reduction step, all rotations $\mathsf{rot}^j(\mathbf{l})$ (for $j \in [n]$) of $\mathbf{l} \in L$ are considered instead of only considering $\mathbf{l}$. The runtime proof for ListSieve in [MV10b] uses the fact that the number of vectors of bounded norm can be bounded exponentially in the lattice dimension. Therefore, the list size L cannot grow unregulated. All list vectors have norm less than or equal $m\,\|\mathbf{B}\|$. For cyclic and anti-cyclic lattices, the norm of a vector remains unchanged when rotated. Therefore each list vector corresponds to n vectors of the same size, which results in a proven list size of factor n smaller. For prime cyclotomic lattices, the norm might increase when rotated (the expansion factor is > 1 in that case), therefore it is a bit harder to prove bounds on the size of the list.

We assume that for finding a shortest vector in the lattice, the total number of reductions is the same. Our experiments show that this assumption is reasonable (cf. Section 5). In this case we predict the number of iterations of IdealListSieve compared to ListSieve. When ListSieve performs t iterations (sampling - reducing - appending), we assume that IdealListSieve would take t/n iterations, since in t/n steps it can use the same number of list vectors for reduction, namely $n \cdot t/n$. Therefore, we expect the number of iterations for IdealListSieve to be an n-th fraction of ListSieve.

Since in every iteration one single vector is sampled and appended to the list, the maximum list size will be in the order of magnitude as the number of iterations. The runtime of the whole algorithm is depending on

the number of iterations itself. Since we are performing n possible reductions instead of one in each iteration, the time needed for one iteration is supposed to increase a bit. So we expect the total runtime to increase a bit less than the number of iterations. Figure 1 shows the expected speedup factors of IdealListSieve compared to ListSieve.

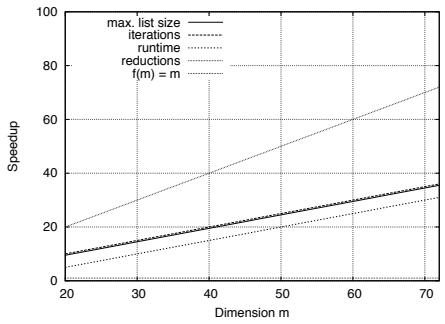


Fig. 1. Predicted speedups for IdealListSieve compared to ListSieve. The values for the number of iterations I and the list size L are computed with $f(m) = m/2$. The number of reductions R is expected to be close to 1, the total runtime a bit less than the I.

Recall that the speedups predicted in this section are asymptotical. They do not necessarily hold in practice, since we can only run experiments in small dimensions $m \leq 80$. In the next section, we present experimental results comparing the two algorithms to show if our predictions hold in practice.

5 Experiments

The public implementation of [Vou10] (called `gsieve`) implements the GaussSieve algorithm. Based on this, we implemented ListSieve, IdealListSieve, and IdealGaussSieve. ListSieve is essentially the `gsieve` implementation without stack functionality. IdealListSieve uses the function `IdealListReduce` of Section 3 in addition. Both algorithms do not use perturbations. IdealGaussSieve implements GaussSieve with the additional function. All three implemented algorithms is published online.[2]

Since we are using the NTL-library [Sho], it would be possible to implement a generic function `IdealReduce` for all polynomials f. However, specializing on a special class of polynomials allows some code

[2] `http://www.cdc.informatik.tu-darmstadt.de/mitarbeiter/mischnei.html`

improvements and leads to a huge speed-up in practice. Therefore, we have implemented three different functions, namely `AntiCyclicReduce`, `CyclicReduce`, and `CyclotomicReduce`. These functions can be used for sieving in anti-cyclic, cyclic, or prime cyclotomic lattices, respectively. Here we present experimental results for cyclic and prime cyclotomic lattices.

All experiments were performed on an AMD Opteron (2.3GHz) quad core processor, using one single core. We only apply LLL as pre-reduction, not BKZ. This is due to the fact that BKZ-reduction is too strong in small dimensions, and the sieving algorithms are not doing any work if BKZ already finds the shortest vector. Interestingly, we encountered in our experiments that the effect of pre-reduction for sieving is much less noticeable as in the enumeration case.

The results shown in this section are average values of 10 randomly generated lattices in each dimension. Since we do not know the length of a shortest vector in these lattices, we ran an SVP algorithm first to assess the norm. So we can stop our sieving algorithms as soon as we have found a vector below that bound. For cyclic and prime cyclotomic lattice we chose $n \in \{10, 12, 16, 18, 20, 22, 28, 30, 32\}$ and $m = 2n$. These are the values where $n + 1$ is prime, which is important for prime cyclotomic lattices. We chose these values for cyclic lattices as well in order to have results for both lattice types in the same dimensions. The generator of the ideal lattices is included in Sage [S+10] since version 4.5.2. The modulus q was fixed as 257. Naturally, the determinant of the lattices is q^n, i.e., 257^n. For a second series of experiments, we generate cyclic and prime cyclotomic lattices with $m = 4n$. We choose $n \in \{6 \dots 15\}$ (cyclic) and $n \in \{6, 10, 12, 16\}$ (prime cyclotomic), q is again 257.

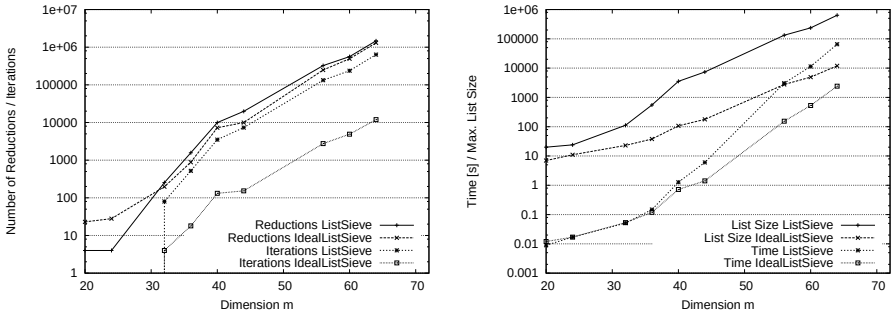
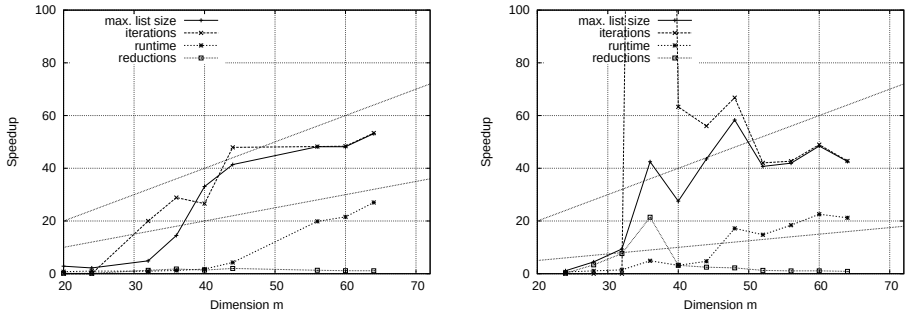


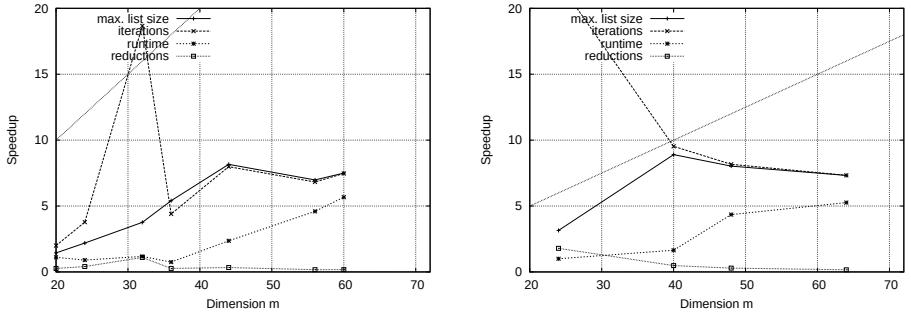
Fig. 2. Results for cyclic lattices. Left: The number of reductions is comparable for ListSieve and IdealListSieve, whereas the number of iterations differs. Right: The maximum list size as well as the runtime decrease for IdealListSieve.

Figure 2 shows the results concerning R, I, L, and the runtime for cyclic lattices. The speedups for cyclic lattices are shown in Figure 3 and for prime cyclotomic lattices in Figure 4. Figure 3(a) shows the speedups of IdealListSieve compared to ListSieve. More exactly it shows the values for the number of iterations I, the maximum list size L, the runtime, and the total number of reductions R of ListSieve divided by the same values for IdealListSieve in the same lattices. Figure 3(b) shows the same data for $m = 4n$. Figures 4(a) and (b) show the same data using cyclotomic lattices. All graphs contain a line for the identity function $f(m) = m$, and a line for $f(m) = m/2$ or $f(m) = m/4$, in order to ease comparison with the prediction of Figure 1.



(a) ListSieve, cyclic lattices $m = 2n$ (b) ListSieve, cyclic lattices $m = 4n$

Fig. 3. Speedup of IdealListSieve compared to ListSieve, for cyclic lattices.



(a) ListSieve, cyclotomic lattices $m = 2n$ (b) ListSieve, cyclotomic lattices $m = 4n$

Fig. 4. Speedup of IdealListSieve compared to ListSieve, for cyclotomic lattices.

Interpretation. In small dimensions, the results are kind of abnormal. Sometimes one of the ideal lattice variant of the algorithms finds shortest

vectors very fast, which results in speedups of more than 100, e.g. in dimension $m = 36$ in Figure 3(b). Therefore, small dimensions of (say) less than 40 should be taken into account only carefully. Testing higher dimensions failed due to time reasons. Neither better pre-reduction nor searching for longer vectors helped decreasing the runtime noticeably.

A first thing that is apparent is that the number of reductions R stays nearly the same in all cases. With increasing dimension the speedup tends to 1 in all cases. Therefore our assumption was reasonable, namely that the number of reductions required to find a shortest vector is the same for the ideal and the non-ideal variant of ListSieve.

The higher the dimension gets, the closer the list size L and the iteration counter I get. Again this is how we expected the algorithms to behave. The runtime grows slower than the number of iterations. In dimension $m = 64$ for example, IdealListSieve finds a shortest vector 53 times faster than ListSieve.

Considering the number of iterations I, we see that our prediction was not perfect. For cyclic lattices, the speedups of IdealListSieve are higher than the predicted factor n; the factor is between n and m (for both $m = 2n$ and $m = 4n$). This implies that compared to the non-ideal variant, the same number of reductions is reached in less iterations. In other words, rotating a list vector $\mathbf{l}$ is better than sampling new vectors, for cyclic lattices. Unfortunately, it is not possible from our experiments to predict the asymptotic behaviour. Testing higher dimension is not possible due to time restrictions.

In case of prime cyclotomic lattices, the situation is different. The speedup of iterations is much smaller than for cyclic lattices (≤ 10 in all dimensions). The only difference between both experiments is the type of lattices. The rotations of prime cyclotomic lattices are less useful than those of cyclic lattices. A possible explanation for this is that rotating a vector of a cyclic lattice does not change the norm of the vector, whereas the rotations of prime cyclotomic lattice vectors have increased norms. The expansion factor of a ring $\mathbf{R}$ denotes the maximum "blow up" factor of a ring element when multiplied with a second one. More exactly, the expansion factor $\theta_2(\mathbf{f})$ of a polynomial $\mathbf{f} \in \mathbf{R}$ in the Euclidean norm is defined

$$\theta_2(\mathbf{f}) = \min \left\{ c : \left\| \mathbf{a} x^i \right\|_2 \leq c \left\| \mathbf{a} \right\|_2 \forall \mathbf{a} \in \mathbb{Z}[x]/\langle \mathbf{f} \rangle \text{ for } 0 \leq i \leq n-1 \right\} .$$

The expansion factor in the Euclidean norm is considered in [SSTX09]. For cyclic (and anti-cyclic) lattices it is easy to see that this factor is 1.

For prime cyclotomic lattices, it is $\sqrt{\frac{n+1}{2} + \sqrt{\left(\frac{n+1}{2}\right)^2 - 1}} \approx \sqrt{n}$ (for a proof see Appendix B). So when the norm of the rotated list vectors $\mathbf{l}$ increases, this lowers the probability of a vector to be useful for reduction of the new sample. Therefore, compared to cyclic lattices, the speedup for iterations decreases. But still, sieving in prime cyclotomic lattices using the IdealListSieve is up to 10 times faster than in the original case.

IdealGaussSieve. We also performed experiments using the GaussSieve implementation of Voulgaris and our IdealGaussSieve version. These experiments are shown in Appendix A. The speedup factor are comparable to those of IdealListSieve.

Anti-Cyclic Lattices. Lattices corresponding to ideals in the ring factored with $\mathbf{f}(x) = x^n + 1$ behave exactly as cyclic lattices. The algebra of both rings differs, but the algorithmic behaviour is exactly the same. In order to have the polynomial f irreducible, we choose $n \in \{2, 4, 8, 16, 32\}$ and $m = 2n$.

6 Conclusion and Further Work

We have shown that it is indeed possible to make use of the special structure of ideal lattices when searching for shortest vectors. The linear speedup does not affect the asymptotic runtime of the SVP algorithms, but it allows for some improvements in practice. Our experiments show that runtime speedups of up to 60 are possible in suitable lattice dimensions. With this we also propose the fastest sieving implementation for ideal lattices.

Unfortunately, the projection of ideal lattices is no more ideal. This prevents the usage of Ideal-Sieve in block-wise reduction algorithms like the BKZ algorithm.

Ideal Enumeration. The enumeration algorithm for exhaustive search for shortest lattice vectors can also exploit the special structure of cyclic lattices. In the enumeration tree, linear combinations $\sum_{i=1}^{n} x_i \mathbf{b}_i$ in a specified search region are considered. For cyclic (and also anti-cyclic) lattices, a coefficient vector $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and its rotations $\mathsf{rot}^i \cdot \mathbf{x}$ for $i \in [n]$ specify the same vector. Therefore it is sufficient to enumerate the subtree predefined by one of the rotations. It is for example possible to choose

only the coefficient vectors where the top coordinate $\mathbf{x}_n$ is the biggest entry, i.e., $\mathbf{x}_n = \max_i(\mathbf{x}_i)$. This would decrease the number of subtrees in the enumeration tree with a factor of up to n.

Unfortunately, this approach is only applicable if the input matrix has circular structure. When LLL-reducing the basis, usually the special structure of the matrix is destroyed. Therefore, when applying enumeration for ideal lattices one would loose the possibility of pre-reducing the lattice. Even the symplectic LLL [GHGN06] does not maintain the circulant structure of the basis.

A second flaw of the ideal enumeration is that it is not applicable to ideal, non-cyclic lattices. For cyclic lattices it is easy to specify which rotations predefine the same vector, which does not work in the non-cyclic case. As a conclusion we state that ideal sieving is much more practical than ideal enumeration would be.

References

[ADL+08] Yuriy Arbitman, Gil Dogon, Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFTX: A proposal for the SHA-3 standard, 2008. In The First SHA-3 Candidate Conference.

[AKS01] Miklos Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610. ACM, 2001.

[BN09] Johannes Blömer and Stefanie Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. *Theor. Comput. Sci.*, 410(18):1648–1665, 2009.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

[GHGN06] Nicolas Gama, Nick Howgrave-Graham, and Phong Q. Nguyen. Symplectic lattice reduction and NTRU. In *EUROCRYPT*, volume 4004 of *LNCS*, pages 233–253. Springer, 2006.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In *EUROCRYPT*, volume 6110 of *LNCS*, pages 257–278. Springer, 2010.

[GS10] Nicolas Gama and Michael Schneider. SVP Challenge, 2010. available at `http://www.latticechallenge.org/svp-challenge`.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, volume 1423 of *LNCS*, pages 267–288. Springer, 1998.

[HS07] Guillaume Hanrot and Damien Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm. In *CRYPTO*, volume 4622 of *LNCS*, pages 170–186. Springer, 2007.

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In *SODA 2000*, pages 937–941. ACM, 2000.

[LLL82] Arjen Lenstra, Hendrik Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 4:515–534, 1982.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP*, volume 4052 of *LNCS*, pages 144–155. Springer, 2006.

[LM08] Vadim Lyubashevsky and Daniele Micciancio. Asymptotically efficient lattice-based digital signatures. In *TCC*, volume 4948 of *LNCS*, pages 37–54. Springer, 2008.

[LMPR08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. Swifft: A modest proposal for FFT hashing. In *FSE*, volume 5086 of *LNCS*, pages 54–72. Springer, 2008.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, volume 6110 of *LNCS*, pages 1–23. Springer, 2010.

[Lyu08] Vadim Lyubashevsky. Towards practical lattice-based cryptography. Phd thesis, University of California, San Diego, 2008.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT*, volume 5912 of *LNCS*, pages 598–616. Springer, 2009.

[Mic07] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007. Preliminary version in FOCS 2002.

[MR08] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes A. Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer, 2008.

[MV10a] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In *STOC*, pages 351–358. ACM, 2010.

[MV10b] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *SODA*, pages 1468–1480. ACM/SIAM, 2010.

[NV08] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *J. of Mathematical Cryptology*, 2(2), 2008.

[PR06] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *TCC*, volume 3876 of *LNCS*, pages 145–166. Springer, 2006.

[PS08] Xavier Pujol and Damien Stehlé. Rigorous and efficient short lattice vectors enumeration. In *Asiacrypt 2008*, volume 5350 of *LNCS*, pages 390–405. Springer, 2008.

[PS09] Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. Cryptology ePrint Archive, Report 2009/605, 2009.

[S$^+$10] W. A. Stein et al. *Sage Mathematics Software (Version 4.5.2)*. The Sage Development Team, 2010. `http://www.sagemath.org`.

[Sho] Victor Shoup. Number theory library (NTL) for C++. `http://www.shoup.net/ntl/`.

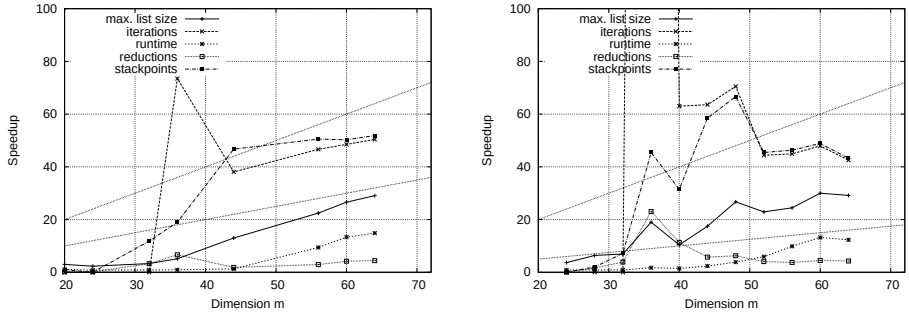
[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, volume 5912 of *LNCS*. Springer, 2009.

[Vou10] Panagiotis Voulgaris. Gauss Sieve alpha V. 0.1, 2010. Available at `http://cseweb.ucsd.edu/~pvoulgar/impl.html`.

[WLTB10] Xiaoyun Wang, Mingjie Liu, Chengliang Tian, and Jingguo Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. Cryptology ePrint Archive, Report 2010/647, 2010. `http://eprint.iacr.org/`.

A GaussSieve Results

Here we present our results comparing GaussSieve to IdealGaussSieve. Figure 5 shows the experimental data.



(a) GaussSieve, cyclic lattices $m = 2n$ (b) GaussSieve, cyclic lattices $m = 4n$

Fig. 5. Speedup of IdealGaussSieve compared to GaussSieve, for cyclic lattices.

B Expansion Factor in the Euclidean Norm

The expansion factor $\theta_2(\mathbf{f})$ of a polynomial $\mathbf{f} \in \mathbf{R} = \mathbb{Z}[x]/\langle \mathbf{f} \rangle$ in the Euclidean norm is defined as

$$\theta_2(\mathbf{f}) = \min \left\{ c : \left\| \mathbf{a} x^i \right\|_2 \leq c \left\| \mathbf{a} \right\|_2 \forall \mathbf{a} \in \mathbb{Z}[x]/\langle \mathbf{f} \rangle \text{ for } i \in [n] \right\} .$$

It is easy to see that $\theta_2(x^n - 1) = \theta_2(x^n + 1) = 1$. It is an easy adaption of the proof of [Lyu08, Lemma 2.6], already mentioned in [SSTX09]. Here, we will present the expansion factor of $\mathbf{f} = x^{n-1} + x^{n-2} + \ldots + x + 1$ in the Euclidean norm.

Let λ_i be the eigenvalues of the matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$, for $i \in [n]$. The spectral radius of $\mathbf{M}$ is the maximum of the absolute values of the eigenvalues of the matrix $\mathbf{M}$, i.e. $\rho(\mathbf{M}) = \max_{i \in [n]} \{|\lambda_i|\}$.

The rotation matrix is $\mathbf{A} = \left(\begin{array}{c|c} \mathbf{0}_{n-1}^T & -\mathbf{1}_n \\ \mathbf{I}_{n-1} & \end{array} \right)$. Lemma 1 will help us later in the proof of the expansion factor of $\mathbf{f}$.

Lemma 1. *For every $i \in [m]$, the spectral radius of the matrix $(\mathbf{A}^i)^T(\mathbf{A}^i)$ is $\rho = \frac{m+1}{2} + \sqrt{\left(\frac{m+1}{2}\right)^2 - 1}$.*

Proof. We are looking for the eigenvalues of the matrix $(\mathbf{A}^i)^T(\mathbf{A}^i)$, i.e. the values λ where $(\mathbf{A}^i)^T(\mathbf{A}^i)\cdot\mathbf{x} = \lambda\mathbf{x}$ for an $\mathbf{x}\in\mathbb{R}^n$. It is

$$\mathbf{A}^i = \left(\begin{array}{c|c|c} \mathbf{0} & -\mathbf{1}_n & \mathbf{I}_{i-1} \\ \mathbf{I}_{n-i} & & \mathbf{0} \end{array}\right) \quad \Rightarrow \quad (\mathbf{A}^i)^T\cdot(\mathbf{A}^i) = \left(\begin{array}{c|c|c} \mathbf{I}_{n-i} & -\mathbf{1}_{n-i} & \mathbf{0} \\ -\mathbf{1}_{n-i}^T & n & -\mathbf{1}_{i-1}^T \\ \mathbf{0} & -\mathbf{1}_{i-1} & \mathbf{I}_{i-1} \end{array}\right).$$

The equation system $\left((\mathbf{A}^i)^T(\mathbf{A}^i) - \lambda\mathbf{I}_n\right)\cdot\mathbf{x} = 0$ is equal to

$$\left(\begin{array}{c|c|c} (1-\lambda)\mathbf{I}_{n-i} & -\mathbf{1}_{n-i} & \mathbf{0} \\ -\mathbf{1}_{n-i}^T & n-\lambda & -\mathbf{1}_{i-1}^T \\ \mathbf{0} & -\mathbf{1}_{i-1} & (1-\lambda)\mathbf{I}_{i-1} \end{array}\right)\cdot\mathbf{x} = \mathbf{0}\,.$$

We consider two different cases for the value of λ.
Case 1 ($\lambda \neq 1$): Dividing the first $n-1$ and the last $i-1$ rows by $1-\lambda$ leads to

$$\left(\begin{array}{c|c|c} \mathbf{I}_{n-i} & -\mathbf{1}_{n-i}/(1-\lambda) & \mathbf{0} \\ -\mathbf{1}_{n-i}^T & n-\lambda & -\mathbf{1}_{i-1}^T \\ \mathbf{0} & -\mathbf{1}_{i-1}/(1-\lambda) & \mathbf{I}_{i-1} \end{array}\right)\cdot\mathbf{x} = \mathbf{0}\,.$$

The sum of all rows leads to 0 in the first $n-i$ and in the last $i-1$ columns. In the $(n-i+1)$th column, we compute

$$\left(\frac{(n-1)\cdot(-1)}{1-\lambda} + n - \lambda\right)\cdot\mathbf{x}_{n-i+1} = 0 \quad \Leftrightarrow \quad \frac{\lambda^2-(n+1)\lambda+1}{1-\lambda} = 0.$$

Therefore we derive the first two eigenvalues $\lambda_{1,2} = \frac{n+1}{2} \pm \sqrt{\left(\frac{n+1}{2}\right)^2 - 1}$.
Case 2 ($\lambda = 1$): The equation system is in this case

$$\left(\begin{array}{c|c|c} & -\mathbf{1}_{n-i} & \\ -\mathbf{1}_{n-i}^T & n-1 & -\mathbf{1}_{i-1}^T \\ & -\mathbf{1}_{i-1} & \end{array}\right)\cdot\mathbf{x} = \mathbf{0}$$

which is equivalent to $x_{n-i+1} = 0 \quad \wedge \quad \sum_{j=1,j\neq n-i+1}^{n} x_j = 0$. This defines an eigenspace of dimension $n-2$. Therefore, the eigenvalue 1 has geometric multiplicity $n-2$. Since the geometric multiplicity is smaller or equal to the algebraic multiplicity, and we have already found 2 eigenvalues, the algebraic multiplicity of the eigenvalue 1 is also $n-2$. Since we have found all eigenvalues we can compute the maximum $\max(1,\lambda_1,\lambda_2)$, which is always $\lambda_1 = \frac{n+1}{2} + \sqrt{\left(\frac{n+1}{2}\right)^2 - 1}$. □

Now we can proof the expansion factor of prime cyclotomic polynomials in the Euclidean norm.

Lemma 2. *If* $\mathbf{f} = x^{n-1} + x^{n-2} + \ldots + x + 1$*, then the expansion factor of* $\mathbf{f}$ *is* $\theta_2(\mathbf{f}) = \sqrt{\frac{n+1}{2} + \sqrt{\left(\frac{n+1}{2}\right)^2 - 1}}$.

Proof. The operator norm of an operator between normed vector spaces X and Y over the same base field, $A : X \to Y$, is defined as

$$\|A\| = \sup\left\{\frac{\|Ax\|}{\|x\|} : x \in X, \|x\| \neq 0\right\} = \inf\left\{c : \|Ax\| \leq c\,\|x\| \,\forall x \in X\right\}.$$

This can be defined for every norm of the base field of the vector spaces. The expansion factor defined for the ring $\mathbb{Z}/\langle\mathbf{f}\rangle$ is the maximum of the operator norms of the operators that perform multiplication with x^i, for $i \in [n]$. Using the usual embedding from the ring to a vector space, the operators can be described by the matrices $\mathbf{A}^i$.

For the Euclidean norm the operator norm of a matrix $\mathbf{M}$ is equal to the square root of the spectral radius of the matrix $\mathbf{M}^T\mathbf{M}$. In our case, we have to compute the spectral radius of the matrix $(\mathbf{A}^i)^T(\mathbf{A}^i)$. Therefore, the expansion factor equals the square root of the value computed in Lemma 1. □