Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search

Anja Becker[1], Nicolas Gama[2], and Antoine Joux[3,4]

[1] EPFL, Lausanne, Switzerland
[2] UVSQ, Versailles, France
[3] Laboratoire d'Informatique de Paris 6, UPMC Sorbonnes Universités, Paris
[4] CryptoExperts, France and Chaire de Cryptologie de la Fondation de l'UPMC
anja.becker@epfl.ch, nicolas.gama@prism.uvsq.fr, antoine.joux@m4x.org

Abstract. We give a simple heuristic sieving algorithm for the m-dimensional exact shortest vector problem (SVP) which runs in time $2^{0.3112m+o(m)}$. Unlike previous time-memory trade-offs, we do not increase the memory, which stays at its bare minimum $2^{0.2075m+o(m)}$. To achieve this complexity, we borrow a recent tool from coding theory, known as nearest neighbor search for binary code words. We simplify its analysis, and show that it can be adapted to solve this variant of the fixed-radius nearest neighbor search problem: Given a list of exponentially many unit vectors of $\mathbb{R}^m$, and an angle $\gamma\pi$, find all pairs of vectors whose angle $\leq \gamma\pi$. The complexity is sub-quadratic which leads to the improvement for lattice sieves.

Keywords: Nearest neighbor search, lattice, sieve

1 Introduction

Lattice-based cryptography is a promising area due to the simple additive, parallelizable structure of a lattice. Additionally, lattices serve as a tool to attack cryptosystems that are not necessarily lattice-based themselves. The two basic hard problems shortest vector problem (SVP) and closest vector problem (CVP) are known to be NP-hard[1] to solve exactly [1, 16] and also NP-hard to approximate [11, 20] within at least constant factors. Moreover, hard lattice problems have a long standing relationship to number theory and cryptology. In number theory, they can for example be used to find Diophantine approximations. Together with the worst-case reduction by Ajtai [1] this provides a very good starting point for lattice-based cryptography.

The time complexity of known algorithms that find the *exact* solution to SVP or CVP are at least exponential in the dimension of the lattice. These algorithms also serve as subroutines for strong polynomial time *approximation* algorithms which have been shown to be of great use as a cryptanalytic tool. In cryptology, they were used for a long time, first through a direct approach as in [14] and then more indirectly using Coppersmith's small roots algorithms [9, 10]. Algorithms for the exact problem enable us to choose appropriate parameters for cryptographic schemes.

[1] Under randomized reductions in the case of SVP.

A *shortest* vector can be found by enumeration [26, 15], sieving [2, 24, 22, 27, 6, 12, 28, 17, 18] or the Voronoi-cell algorithm [21]. Enumeration uses a negligible amount of memory and its running time is between $2^{\tilde{\mathcal{O}}(m)}$ and $2^{\tilde{\mathcal{O}}(m^2)}$ depending on the amount and quality of the preprocessing. Probabilistic sieving algorithms, as well as the deterministic Voronoi-cell algorithm are simply exponential in time and memory.

Previous work has shown how to reduce the time complexity by exponential factors while however increasing the memory requirement by exponential factors. As the memory requirement proves to represent the bottleneck in practice, researchers are looking for only slight or no increase in the memory requirement. Indeed, the most efficient sieve in practice is not the one of lowest asymptotic time complexity. It is a parallelized version of the Gauss sieve [24, 7, 13, 23, 19] for which the time complexity is unproven but postulated to be at most $2^{0.41m+o(m)}$. The complexity hence differs to less expensive sieves by exponential factors while at the same time the memory complexity is smaller by an exponential factor. We give details in the following paragraph and in Sect. 2. A provable variant of same asymptotic complexity is the Nguyen-Vidick sieve [24] (NV sieve) for which we propose a heuristic improvement.

Our contribution. The basic step in a lattice sieve algorithm is to search pairs of vectors for which the norm of the sum or difference is smaller than the one of the input vectors. The number of potential candidate vectors is exponential in the lattice dimension and the cost to perform a reduction is hence also exponential. It represents the dominating cost of the sieve and we are highly interested in a most efficient execution.

Here, we propose a new and faster method that searches candidate vectors for reduction. This reduces the currently used method which performs a trivial exhaustive search over all vectors in the list. The new technique stems from a new method for nearest neighbor search in the domain of coding theory [3]. We develop a variant, presented in Sect. 3, that applies to the domain of lattices and can replace the trivial reduction search in a lattice sieve by our method. The complexity of an NV sieve [24] (cf. Sect. 2) is hence reduced from $2^{0.41m+o(m)}$ to $2^{0.3112m+o(m)}$ without changing the required memory of $2^{0.208m+o(m)}$ in its exponential factors[5] as illustrated in Fig. 1. This stands in contrast to currently proposed methods that reduce the time while increasing the memory by exponential factors.

Organization of the paper. In Section 2 we provide a brief background on Euclidean lattices and sieving algorithms (Sect. 2.1). We also summarize the nearest neighbor search algorithm by May and Ozerov in Sect. 2.2 and discuss its limitations. The following Sect. 3 presents our new method to find neighbor vectors which is based on the method by May and Ozerov. We provide an analysis of the complexity and finally show how to apply it to the NV-sieve in Section 4.

[5] The references in the figure refer to following publications in the bibliography: [NV08] = [24], [MV10] = [21], [WLTB11] = [27], [BGJ14]= [6], [La14] = [17] and [LdW15] = [18]

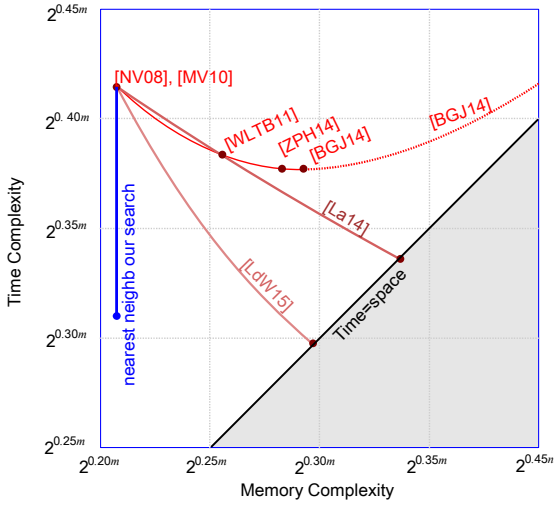

Fig. 1. New sieve vs. other time-memory trade-offs.

2 Required basic background

We here briefly summarize essential facts about Euclidean lattices, lattice problems for cryptographic use, important lattice algorithms and a nearest neighbor search algorithm.

2.1 Euclidean lattices and lattice sieve algorithms

Euclidean lattices. A lattice $\mathcal{L}$ of dimension $s \leq m$ is a discrete subgroup of $\mathbb{R}^m$ that spans an s-dimensional linear subspace. A lattice can be described as the set of all integer combinations $\{\sum_{i=1}^{s} \alpha_i \boldsymbol{b}_i \,|\, \alpha_i \in \mathbb{Z}\}$ of s linearly independent vectors $\boldsymbol{b}_i$ of $\mathbb{R}^s$. Such vectors $\boldsymbol{b}_1, .., \boldsymbol{b}_m$ are called a basis of $\mathcal{L}$. The volume of the lattice $\mathcal{L}$ is the volume of $\text{span}(\mathcal{L})/\mathcal{L}$, and can be computed as $\sqrt{\det(BB^t)}$, for any basis B. The volume and the dimension are invariant under change of basis. Any lattice has a shortest non-zero vector of Euclidean length $\lambda_1(\mathcal{L})$ which can be upper bounded by Minkowski's theorem as $\lambda_1(\mathcal{L}) \leq \sqrt{m}\,\text{vol}\,(\mathcal{L})^{1/m}$. The two basic hard problems shortest vector problem (SVP) and closest vector problem (CVP) are known to be NP-hard[1] to solve exactly [1, 16] and also NP-hard to approximate [11, 20] within at least constant factors. The problems can be solved exactly or approximately by finding a shortest or closest vector up to a given factor.

Sieving algorithms to solve SVP. Asymptotically, the most efficient method to solve the exact SVP in practice is based on sieving. Lattice sieving algorithms usually start with exponentially many large lattice vectors, and iteratively combine these vectors together, for instance replacing vectors by their difference with other close vectors in the list, when they are shorter. The algorithm goes on, while the norm of the vectors in the list decreases, until the list contains

[1] Under randomized reductions in the case of SVP.

Algorithm 1 Nguyen-Vidick-like sieve

Input: A set L of vectors, of $\mathcal{L}$, a parameter $\varepsilon > 0$
Output: A shortest vector of $\mathcal{L}$

$R \leftarrow \text{maxNorm}(L)$
$L' \leftarrow \emptyset$
repeat
  $L' \leftarrow \{\boldsymbol{v} \in L \text{ s.t. } \|v\| \leq (1-\varepsilon)R\}$
  $\mathcal{N} \leftarrow \text{FindNeighbors}(L, \gamma = \frac{1}{3} - \varepsilon)$
  for each $(\boldsymbol{u}, \boldsymbol{v}) \in \mathcal{N}$ **do** $L' \leftarrow L' \cup \{\boldsymbol{u} - \boldsymbol{v}\}$
  $L \leftarrow L'$, $R \leftarrow (1-\varepsilon).R$
until $|L'| \approx 1$
return shortestVector in L

the shortest lattice vector, or at least, a very good approximation. Sieving algorithms differ by the strategy to find pairs of vectors close to each other in a list of random lattice vectors. The simplest algorithms essentially test all pairs of vectors using a quadratic-time approach to detect close vectors. For instance, the NV-sieving algorithm [24] performs many straightforward iterations on the list, each iteration computing at most one difference at a time, whilst the Gauss-sieve algorithm [21] performs as many reductions as possible for each vector of the list, and needs backtracking.

The sieving criterion to determine if the difference of two vectors (of approximately the same norm) is smaller is called a *neighboring* criterion, and is roughly equivalent to testing whether the angle between the two vectors is $\leq \pi/3 - \varepsilon$. As explained in [24], a cone of angle $\pi/3$ covers a fraction $\sin(\pi/3)^m$ of the unit sphere, so a list of more than $\tilde{\mathcal{O}}(\sin(\pi/3)^{-m}) = \tilde{\mathcal{O}}(2^{0.208m})$ elements must contain neighbor vectors. For this reason, all known sieving algorithms take as input a list of $2^{0.208m+o(m)}$ vectors, and quadratic-time sieving algorithms have complexity $2^{0.415m+o(m)}$.

Various research has been done and is ongoing to reduce the complexity of sieving algorithms [2, 6, 12, 22, 24, 27, 28]. Additionally, a new technique has been developed recently that is based on locality-sensitive hash functions by Charikar [8] and Andoni et al. [4, 5] which reduces the time and memory complexity in the asymptotic case down to $2^{0.298m+o(m)}$ [17, 18]. All these techniques are time versus memory trade-offs, and therefore, increase the memory complexity by exponential factors. In practice, keeping the memory as small as possible seems to be the most important factor to get a good running time, which is confirmed by challenges [25], where quadratic algorithms with minimal memory $2^{0.21m+o(m)}$ achieve better than all time vs. memory trade-offs. For this reason, we will focus on decreasing the running-time of sieving without affecting the memory.

A simplified blueprint of Nguyen-Vidick-like lattice sieving is summarized in Alg. 1, where the quadratic neighboring search phase has been extracted as Alg. 2. Its asymptotic complexity is $2^{0.41m+o(m)}$ in time and $2^{0.208m+o(m)}$ in memory for a random lattice of dimension m. In the next sections, we propose faster alternatives for Alg. 2, which preserve the memory complexity.

Algorithm 2 FindNeighbors (QuadraticNNS, as in NV-sieve)

Input: List $L = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N)$ of vectors, a parameter γ defining neighbors.
Output: A list of neighbor pairs $(\boldsymbol{u}_i, \boldsymbol{u}_j)$ in L^2

```
1: N ← ∅
2: for i = 1 to N do
3:     for j = 1 to i − 1 do
4:         if angle(u_i, u_j) ≤ πγ then N ← N ∪ (u_i, u_j); break;
5:     end for
6: end for
7: return N
```

2.2 Nearest neighbor search for decoding by May and Ozerov, 2015

May and Ozerov recently presented an efficient method to find pairs of close words in a lists of random binary words. They apply this algorithm to decode binary linear codes [3]. Namely, given two lists L and R of $|L| = |R| = 2^{\lambda m}$ uniformly distributed and pairwise independent vectors from $\mathbb{F}_2^m$, find a pair in $L \times R$ of small Hamming distance γm for $\gamma < 1/2$. The proposed algorithm runs in time $|L|^{\frac{1}{1-\gamma}}$. The smaller γ, the better the running time. This stands in contrast to previously known and widely used methods which are a subroutine that perform a test on all tuples and hence run in time $|L|^2$. A similar task appears also in lattice sieves for which we adapt the algorithm in Sect. 3.

The main idea of the algorithm from [3] is to successively apply randomized filters on the input list, which may be computed individually on each word, and which are designed to eliminate a large number of vectors. In their algorithm, a filter corresponds to $2n$ random positions. A word is accepted by the filter if and only if it contains at exactly n *ones* in these $2n$ positions, and exactly $\frac{1-\delta}{2}n$ *ones* in the first n positions. $\delta \in]0, 1[$ and $n \in \mathbb{N}$ are optimization parameters that they must tune to optimize the success probability.

The important point is that neighbor words have a much larger chance to be accepted by the same filter than independent random words. Therefore, the main idea is to apply different random filters to the input list of words, and to recursively check whether neighbors are in the filtered sublists. If this process is repeated with sufficiently many random filters, then each pair of neighbor words is eventually be retrieved.

Discussion of limitations. In their algorithm, May and Ozerov [3] have to deal with technical issues, which we remove in the case of lattices in Sect. 3 and Sect. 4. The first limitation is that the filtering condition is in practice very restrictive. At each recursive level, the filter eliminates at least all words which are unbalanced at the $2n$ positions. This yields large polynomial losses between each iteration. These polynomial losses, raised to the power of the number of levels, make their algorithm only usable for very large lists.

The second issue was that their binary words had a small length m. For this reason, they had to choose m as their main complexity parameter, and at each recursion level, the choices of n and δ had to be adapted to m. In practice, n was proportional to m, and the proportion coefficient depends on γ. This

dependency between n, δ and m increases their number of recursion levels, which, combined with the losses in the filtering conditions, make the overall algorithm less practical.

Overall, the most important fact to remember is that asymptotically, when the input words are very large, and the input list contains $2^{\mathcal{O}(m)}$ elements, then in the most significant recursion levels, the parameter δ becomes very close to 0 and n is larger than $1/\delta^2$. In this case, the overall complexity of the May-Ozerov algorithm is equivalent to $\tilde{\mathcal{O}}(|L|^{\frac{1}{1-\gamma}+\varepsilon})$, which is sub-quadratic, and gets smaller when γ decreases.

3 Nearest neighbor search adapted for sieving

We here describe a variant of [3] as described in Sect. 2.2 which we will use as a subroutine in a sieving algorithm to find vectors suitable for reduction as we present in Sect. 4. Our algorithm overcomes the limitations of the previous nearest neighbor search algorithm as discussed in the above section. In contrast to the May-Ozerov proposition, we ensure that the filtering conditions are easier to fulfill, and we obtain non-negligible filtering probabilities. We show that in our nearest neighbor search setting, the parameters (equivalent to n and δ) can be arbitrarily fixed, which simplifies a lot the analysis. The overall complexity is again of order $\tilde{\mathcal{O}}(|L|^{\frac{1}{1-\gamma}+\varepsilon})$ searching neighbors at an angle $\gamma\pi$ in an input list L. It hence improves over a classic search of pairs of vectors performed in quadratic time.

Setting. We are given a list of N uniformly distributed real vectors of $\mathbb{R}^m$ of norm one, and we wish to find all pairs of vectors of an angle inferior to $\gamma\pi$ (radians), that we call neighbors. If the input list has N elements, we expect to find about $N^2 \sin(\gamma\pi)^m$ pairs of neighbors. The naive approach is the quadratic nearest neighbor search (QuadraticNNS), which tests all pairs, and which is essentially optimal when N is arbitrarily large. Instead, we will focus on the particular case where N is smaller than $1/\sin(\gamma\pi)^m$. In this case, we expect to find $\mathcal{O}(N)$ neighbors, and we will adapt and simplify the algorithm of [3] to find these pairs in sub-quadratic time $\mathcal{O}(N^{\frac{1}{1-\gamma}+\varepsilon})$.

Basic principles. The main idea of the algorithm remains to successively apply randomized filters, which are efficient functions from $\mathbb{R}^m$ to $\{0, 1\}$. Random vectors should be accepted by the filter with some low probability P_f. But most importantly, a pair of neighboring vectors must have a much larger probability P_p to be simultaneously accepted by the filter than P_f^2 for a pair of randomly chosen vectors. As we will see later, this difference of probability between P_p and P_f^2 is precisely what causes the sub-quadratic complexity.

In our setting, the set of random filters is denoted $\mathcal{F}(n, \delta)$, and is parametrized by a number $n \in \mathbb{N}$ of tests, and a weight constraint $\delta \in]0, 1[$. To instantiate a filter $f \in \mathcal{F}$, we choose $n \in \mathbb{N}_{>0}$ directions $\boldsymbol{u}_i$ uniformly over the 1-sphere of $\mathbb{R}^m$. A non-zero vector $\boldsymbol{v} \in \mathbb{R}^n$ passes the filter f if and only if there are at most $\frac{1-\delta}{2}n$ directions $\boldsymbol{u}_i$ such that $\langle \boldsymbol{v}, \boldsymbol{u}_i \rangle \geq 0$. In this case, we set $f(\boldsymbol{v}) = 1$, else $f(\boldsymbol{v}) = 0$ means that the vector $\boldsymbol{v}$ is rejected.

Procedure. Let $\gamma < 1/2$ be fixed ($\gamma\pi$ is the angle between neighbors) and let $\delta > 0$, $n \in \mathbb{N}$ and K be three optimization parameters. Let L be a list of N real vectors in $\mathbb{R}^m$, whose directions are uniformly distributed on the sphere of radius one. The goal is to establish the list of all *neighboring pairs*.

- If N is too small (say $N \leq N_{\min}$), we use the naive quadratic algorithm, which checks all pairs.
- If N is larger, we repeat K times the following: We choose a random filter $f \leftarrow \mathcal{F}(n, \delta)$, we extract only the sublist $L' = \{\boldsymbol{v} \in L \text{ s.t. } f(\boldsymbol{v}) = 1\}$ of vectors which pass the filter and we continue recursively on L'.

Algorithm 3 presents a pseudo code of the algorithm.

Algorithm 3 Nearest neighbor search (NNS)

Require: A list L of N unit vectors of $\mathbb{R}^m$
Ensure: The list of all neighboring vectors
Global: Repetitions K, an angle $\pi\gamma$ defining neighbors, filtering parameters δ, n, a lower-bound $N_{min} \in \mathbb{N}$ for the recursion

if $N < N_{min}$ **then**
  Return $(\boldsymbol{v}, \boldsymbol{w}) \in L^2$ s.t. $\langle \boldsymbol{v}, \boldsymbol{w} \rangle \geq \cos(\pi\gamma)$ via QuadraticNNS(L)
end if
$\mathcal{N} \leftarrow \emptyset$
for $i = 1$ to K **do**
  Pick a random filter $f \in \mathcal{F}$
  Filter $L' = \{\boldsymbol{v} \in \mathcal{L} \text{ s.t. } f(\boldsymbol{v}) = 1\}$
  $\mathcal{N} \leftarrow \mathcal{N} \cup \mathrm{NNS}(L')$
end for
Return $\mathcal{N}$

Analysis. The success of Alg. 3 depends on two probabilities. We call $P_f(\delta, n)$ the probability that a vector $\boldsymbol{v}$ is accepted by a random filter, and $P_p(\gamma, \delta, n)$ the probability that two vectors $\boldsymbol{v}, \boldsymbol{w}$, having an angle $\gamma\pi$, are simultaneously accepted by the same random filter. Due to the spherical symmetry of the problem, we may arbitrarily fix $\boldsymbol{v} = (1, 0, \ldots, 0)$ and $\boldsymbol{w} = (\cos(\gamma\pi), \sin(\gamma\pi), 0, \ldots, 0)$ without affecting the probabilities when calculating them.

The first probability is easy to estimate: $\boldsymbol{v}$ is accepted if and only if at most $j_0 = \frac{1-\delta}{2}n$ directions are in the same hemisphere as $\boldsymbol{v}$ which happens with probability

$$P_f(\delta, n) = \frac{1}{2^n} \sum_{j=0}^{\frac{1-\delta}{2}n} \binom{n}{j} \ . \tag{1}$$

Since the vectors in the input list are uniformly distributed on the sphere, we thus expect to keep $N \cdot P_f$ elements in L' on average after the filtering step on N elements. The number of accepted elements decreases quickly when δ increases, as the weight requirement on the vectors becomes more restrictive.

Heuristic 1 (Number of elements in lists). We assume that in each step of Alg. 3 we filter NP_f elements given N elements.

The second probability P_p is a bit longer to evaluate. We analyze the setting as illustrated in Fig. 3 where we see two vectors at an angle $\gamma\pi$ and the intersection of the two hemispheres $H_{\boldsymbol{v}}, H_{\boldsymbol{w}}$ defined by $\boldsymbol{v}, \boldsymbol{w}$. We draw n directions $\boldsymbol{u}_i$ and observe in which part of the intersecting hemispheres they fall. We now describe the dependency between: (i) the angle and (ii) the number of directions that fall in the four sections defined by the hemispheres. Let us assume the following scenario: At most j_0 directions are in the hemisphere $H_{\boldsymbol{v}}$ centered in $\boldsymbol{v}$, and at the same time, at most j_0 directions are in the hemisphere $H_{\boldsymbol{w}}$ centered in $\boldsymbol{w}$. If we note i the number of vectors in $H_{\boldsymbol{v}} \cap H_{\boldsymbol{w}}$, j the number of vectors in $H_{\boldsymbol{v}} \cap \overline{H}_{\boldsymbol{w}}$, k the number of vectors in $\overline{H}_{\boldsymbol{v}} \cap H_{\boldsymbol{w}}$, the exact value of P_p can be computed as:

$$P_p(\gamma, \delta, n) = \frac{1}{2^n} \sum_{\substack{i \in [0, j_0], \\ j,k \in [0, j_0 - i]}} \frac{n!}{i!j!k!(n\text{-}i\text{-}j\text{-}k)!} \gamma^{j+k} (1-\gamma)^{n-j-k} \quad \text{with } j_0 = \frac{1-\delta}{2} n. \tag{2}$$

By definition of the probability P_p, it suffices to repeat the main loop about $K = \ln(x) \cdot P_p^{-1}$ times, so that with probability $\geq 1 - \frac{1}{x}$, each pair of neighbors appears at least once in the same sublist. As long there are at most x recursive levels in total, an elementary induction on the recursive calls in Alg. 3 proves that each pair of neighbors is recovered with constant probability.

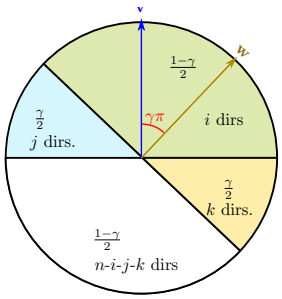

Fig. 2. Intersection of hemispheres.

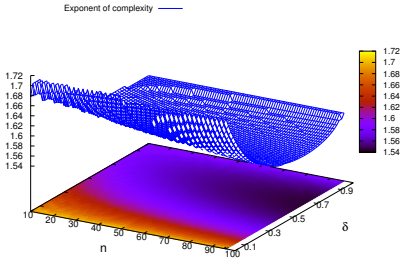

Fig. 3. Development of exponent for different choices of n, δ, $\gamma = 1/3$.

3.1 Complexity analysis for fixed parameters

In this section, we analyze the complexity of our nearest neighbor search (cf. Alg. 3) when the three parameters γ, δ and n are fixed. This means that the probabilities P_p and P_f are constant and we can compute the complexity as presented in the following. We also provide example parameters to illustrate their size and impact on the complexity. We now state our main contribution of the section. Given as input a list of N words, we note $C(N)$ the time complexity, expressed as a number of dot products on $\mathbb{R}^m$.

Theorem 1 (Complexity of NNS). *Let $\gamma < 1/2$, $\delta < 1$ and n be fixed. Let L be a list of $N \leq poly(m) \sin(\pi\gamma)^{-m}$ uniformly random vectors in the sphere of*

dimension m. We call $d = \ln(P_p^{-1})/\ln(P_f^{-1})$ the exponent such that $P_f^d = P_p$. Then the time complexity of Alg. 3 is $\tilde{\mathcal{O}}(N^d)$.

Proof. The overall time complexity to treat a list of length N is given by

$$C(N) = \frac{1}{2}N^2 \text{ when } N \leq N_{\min} \tag{3}$$

$$C(N) = 1/P_p \cdot (C(P_f N) + nN)) \text{ when } N > N_{\min} \tag{4}$$

The first row corresponds to the quadratic algorithm. The second row includes the $1/P_p$ repetitions in which we count one recursive call and one filtering process, composed of n dot products per vector. The induction goes on until the list is very small, which typically means, $N_{min} = \text{poly}(m)$. In order to simplify and solve the above recurrence, we set $X = 1/P_f$ and $u_k = C(N_{min}X^k)$ for $k \in \mathbb{N}$. Then, the sequence $(u_k)_{k\in\mathbb{N}}$ satisfies the following linear recurrence:

$$u_0 = \frac{1}{2}N_{\min}^2,\ u_k = X^d u_{k-1} + N_{\min}X^{d+k}$$

which has the explicit solution:

$$u_k = \left(\frac{1}{2}N_{\min}^{2-d} + \frac{N_{\min}^{1-d}X^d}{X^{d-1}-1}\right)(N_{\min}X^k)^d - \frac{X^d}{X^{d-1}-1}(N_{\min}X^k)$$

Thus, assuming that the complexity of the algorithm increases with N, the final asymptotic complexity satisfies for all $N \geq N_{\min}$

$$C(N) \leq (N_{\min}^{2-d} + 2N_{\min}^{1-d}P_f^{-1})N^d.$$

We recall that this algorithm is particularly interesting for $N = \tilde{\mathcal{O}}(\sin(\pi\gamma)^{-m})$ which is exponential in the dimension of the vector space. Therefore, it suffices to choose $N_{\min}$ polynomial or slightly sub exponential in m so that the whole factor in $C(N)$ vanishes before N^d. Therefore, we have proved that each choice of parameters n, δ, γ corresponds to an exponent d such that asymptotically, finding all pairs of neighbors in a list of size $N \leq \text{poly}(m)/\sin(\gamma\pi)^m$ can be achieved with time complexity $\tilde{\mathcal{O}}(N^d)$, and memory $O(N)$.

To get a feeling of the parameters in practice, we give some example parameters n, δ in Table 1. We fix $\gamma = 1/3$ and compute the resulting exponent d of the complexity. Also Fig. 3 illustrates the complexity development as n and δ change, again for $\gamma = 1/3$.

The next section analyses the asymptotic case and presents the value to which d tends. The analysis shows that asymptotically $1.5 + \varepsilon$ for $\gamma = 1/3$ and small $\varepsilon < 1$ is achieved. The development in Tabel 1 gives us a glance of this behavior in moderate dimensions.

3.2 Complexity analysis in the asymptotic case

In this section, we prove the following theorem, which proves that for each angle $\gamma\pi$, there exists a choice of parameters n, δ for which the complexity exponent of our nearest neighbor search decreases to $\frac{1}{1-\gamma}$.

Table 1. Practical values of n, δ for $\gamma = 1/3$.

n	δ	d	$1/P_f$ ($\approx N_{\min}$)	$1/P_p$ ($\approx K$)
30	0.15	1.6491	5.53	16.79
50	0.25	1.6088	30.81	248.32
70	0.25	1.6026	48.32	499.97
90	0.25	1.5910	134.53	2436.99
30	0.4	1.6019	46.76	473.13
40	0.4	1.5919	120.56	2056.17
50	0.4	1.5842	303.01	8534.24
60	0.4	1.5781	748.94	34366.4
70	0.4	1.5730	1829.42	135444
90	0.4	1.5652	10652.1	2.01164e+06
150	0.4	1.5507	1.88883e+06	5.39998e+09
200	0.35	1.5493	2.40156e+06	7.68068e+09

Theorem 2. *Let $\gamma \in]0, \frac{1}{2}[$ be a parameter and $\varepsilon > 0$ be a small quantity, there exists parameters $n \in \mathbb{N}$ and $\delta > 0$ such that the exponent $d = \ln(P_p(\gamma, n, \delta))/\ln(P_f(n, \delta))$ is $\leq \frac{1}{1-\gamma} + \varepsilon$. For these choices of parameters, the time complexity of Alg. 3 on an input list of $N = \sin(\pi\gamma)^{-m}$ vectors is asymptotically $\tilde{\mathcal{O}}(N^{\frac{1}{1-\gamma}+\varepsilon})$.*

To prove this theorem, we compute asymptotic equivalents of $\ln(P_p)$ and $\ln(P_f)$ for large $n \in \mathbb{N}$, and small $\delta > 0$.

Proof. The probability P_f in (1) is a sum of binomial terms, and since the last index $j_0 = \frac{1-\delta}{2}n$ is lower than $n/2$, P_f satisfies the following bounds:

$$\frac{1}{2^n}\binom{n}{\frac{1-\delta}{2}n} \leq P_f(n, \delta) \leq n \cdot \frac{1}{2^n}\binom{n}{\frac{1-\delta}{2}n} \ .$$

Using the Stirling formula, and the entropy function $H(x) = x\log_2(x) + (1-x)\log_2(1-x)$, we deduce that asymptotically,

$$\ln(P_f^{-1}) \underset{n\to\infty}{\sim} n\ln(2)\left(1 - H\left(\frac{1-\delta}{2}\right)\right) \tag{5}$$

For small values of δ, the entropy function has the following Taylor expansion: $H(\frac{1-\delta}{2}) = \ln(2) - \frac{\ln(2)}{2}\delta^2 + \circ(\delta^3)$. Therefore, we also have the simpler equivalent:

$$\ln(P_f^{-1}) \underset{\substack{\delta\to 0 \\ n\delta^2\to\infty}}{\sim} \frac{1}{2}n\delta^2 \tag{6}$$

We now do the same for the probability P_p, and start by finding the maximal term in the expression of P_p from (2). First, we remark that the multinomial coefficient is the product between $\frac{n!}{i!(j+k)!(n-i-j-k)!}$ and the binomial coefficient $\frac{(j+k)!}{j!k!}$. When the sum $j+k$ is fixed, this binomial coefficient reaches its maximum when $j = k$. Then, writing $i = ni'$ and $j = nj'$, the maximal term of the expression of P_p corresponds to the maximum of the function:

$$f(i',j') := \ln\left(\frac{n!}{ni'!(nj'!)^2(n-ni'-2nj')!}\gamma^{2nj}(1-\gamma)^{n-2nj'}\right)$$

over the domain $i' \in [0, \frac{1-\delta}{2}], j' \in [0, \frac{1-\delta}{2} - i']$. We may approximate $\frac{1}{n}f(i',j')$ with its equivalent via Stirling formula:

$$\frac{f(i',j')}{n} \approx -i'\ln(i') - 2j'\ln(j') - (1-i'-2j')\ln(1-i'-2j') + 2j'\ln(\gamma) + (1-2j')\ln(1-\gamma)$$

A quick computation of the partial derivatives shows that this function increases with i' and j' on its entire domain. The maximum is reached at the upper-frontier of the domain where $i' = \frac{1-\delta}{2} - j'$. Then, substituting i' with $\frac{1-\delta}{2} - j'$ and derivating again, we get that the extremum is reached for $j' = j'_0$ satisfying:

$$\ln(\frac{1-\delta}{2} - j'_0) - 2\ln(j'_0) + \ln(\frac{1+\delta}{2} - j'_0) - 2\ln(\frac{1-\gamma}{\gamma}) = 0$$

or equivalently, j'_0 is the positive solution of the second degree equation:

$$\left[\left(\frac{1-\gamma}{\gamma}\right)^2 - 1\right] . {j'_0}^2 + j'_0 - \left(\frac{1-\delta^2}{4}\right) = 0$$

This is the exact expression of j'_0.

$$j'_0 = \frac{\gamma}{2}\left(\frac{-\gamma + \sqrt{(1-\gamma)^2 - (1-2\gamma)\delta^2}}{1-2\gamma}\right) \tag{7}$$

Finally, $\ln(P_p(\gamma, n, \delta))$ admits the following equivalent when n grows:

$$\ln(P_p(\gamma, n, \delta)) \sim n \cdot \left(\ln(2) - f\left(\frac{1-\delta}{2} - j'_0, j'_0\right)\right) \tag{8}$$

Again, computing the Taylor expansions of $j'_0 = \frac{\gamma}{2} - \frac{\gamma}{4(1-\gamma)}\delta^2 + \circ(\delta^2)$ from (7), and then substituting in $f\left(\frac{1-\delta}{2} - j'_0, j'_0\right)$, we obtain a simpler equivalent when δ is small and n is larger than $1/\delta^2$.

$$\ln(P_p(\gamma, n, \delta)) \underset{\substack{\delta\to 0 \\ n\delta^2\to\infty}}{\sim} \frac{n\delta^2}{2(1-\gamma)} \tag{9}$$

Finally, dividing the two equivalents (9) and (6), we obtain the limit exponent, which proves the theorem:

$$d = \frac{\ln(P_p(\gamma, n, \delta))}{\ln(P_f(n, \delta))} \underset{\substack{\delta\to 0 \\ n\delta^2\to\infty}}{\sim} \frac{1}{1-\gamma} \tag{10}$$

4 Application to lattice sieves

We can apply the above algorithm to solve a shortest vector problem (SVP) or approximate SVP for a given lattice $\mathcal{L}$ of dimension m by means of a sieving algorithm. For instance, the NV sieving algorithms first samples an exponentially large set L_0, of $N = 2^{0.2075m+o(m)}$ long lattice vectors of same norm R_0. Then, a sieving iteration step selects pairs of neighbor vectors from $L_0 \times L_0$ whose distance is $\leq (1-\varepsilon)R_0$, stores their differences in a new list L_1, and continues with L_1.

The NV strategy reduces the running-time by a polynomial factor by ensuring that a vector of the list can be reduced by at most one other vector. If this approach is faster than testing all pairs, it also has the drawback to decrease the number of returned neighbor pairs for L_1, which is by definition always smaller than L_0. And overall, the complexity of an iteration remains quadratic in N.

Here, the idea is to replace a sieving iteration with our sub-quadratic nearest neighbor search algorithm. Since two neighbors have an angle smaller than $\pi/3$, which corresponds to $\gamma = \frac{1}{3}$, this can effectively decrease the time complexity from $\tilde{\mathcal{O}}(N^2)$ to $\tilde{\mathcal{O}}(N^{1.5+\varepsilon})$, depending on the choices of n, δ as in the previous section. More precisely, the asymptotic time complexity drops from $2^{0.4150m+\circ(m)}$ to $2^{0.3112m+\circ(m)}$, and the memory remains of the order of the input $2^{0.2075m+\circ(m)}$.

The substitution of the nearest neighbor search in sieving algorithms is straightforward, modulo the usual minor differences in the distribution of vectors that need to be taken into account.

- First, the input vectors have a discrete distribution on the lattice instead of a continuous radial distribution over $\mathbb{R}^m$. In the first iterations of sieving, it is likely that the directions of Gaussian lattice vectors are indistinguishable from random directions in $\mathbb{R}^m$. However, since the radius of the pool of vectors decreases by a constant factor $1+\varepsilon$ at each iteration, it will eventually reach a critical radius where there are not enough lattice vectors to fill the input list. This was already the case in the original NV-sieve, and it is the main reason why sieving ends up with the shortest lattice vector.
- In the end of each iteration, the difference between each pair of neighbors is put into the input list for the next iteration. At this point, we need to use the heuristic that the difference of all neighbor pairs have uniformly distributed directions in $\mathbb{R}^m$. This is actually the main heuristic in every sieving algorithms.

5 Conclusion

We have presented a new nearest neighbor search algorithm which enumerates all pairs of neighbor vectors in the unit sphere of $\mathbb{R}^m$. This has the immediate application to reduce the complexity of lattice sieving without affecting the memory. It remains open to know where is the practical cross-over point between this strategy and other sieving techniques.

References

1. M. Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions (extended abstract). In *Proceedings of the 30th annual ACM symposium on Theory of computing*, STOC '98, pages 10–19, New York, NY, USA, 1998. ACM Press.
2. M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the 33rd annual ACM symposium on Theory of computing*, STOC 2001, pages 601–610, New York, NY, USA, 2001. ACM Press.
3. I. O. Alexander May. On computing nearest neighbors with applications to decoding of binary linear codes. In *Advances in Cryptology – Eurocrypt 2015*, Lecture Notes in Computer Science. Springer, 2015.
4. A. Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn. Beyond locality-sensitive hashing. In *SODA*, pages 1018–1028, 2014.
5. A. Andoni and I. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *SODA*, 2015.
6. A. Becker, N. Gama, and A. Joux. A sieve algorithm based on overlattices. *LMS Journal of Computation and Mathematics*, 17:49–70, 2014.
7. J. W. Bos, M. Naehrig, and J. van de Pol. Sieving for shortest vectors in ideal lattices: a practical perspective. Cryptology ePrint Archive, Report 2014/880, 2014.
8. M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.
9. D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In U. Maurer, editor, *Advances in Cryptology – EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer Berlin Heidelberg, 1996.
10. D. Coppersmith. Finding a small root of a univariate modular equation. In M. Darnell, editor, *Advances in Cryptology – EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165. Springer Berlin Heidelberg, 1996.
11. I. Dinur, G. Kindler, and S. Safra. Approximating-cvp to within almost-polynomial factors is np-hard. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS '98)*, pages 99–109, Washington, DC, USA, 1998. IEEE Computer Society.
12. G. Hanrot, X. Pujol, and D. Stehlé. Algorithms for the shortest and closest lattice vector problems. In *Coding and Cryptology - Third International Workshop, IWCC 2011, Qingdao, China, May 30-June 3, 2011. Proceedings*, pages 159–190, 2011.
13. T. Ishiguro, S. Kiyomoto, Y. Miyake, and T. Takagi. Parallel Gauss sieve algorithm : Solving the SVP in the ideal lattice of 128-dimensions. Cryptology ePrint Archive, Report 2013/388, 2013.
14. A. Joux and J. Stern. Lattice reduction: A toolbox for the cryptanalyst. *Journal of Cryptology*, 11(3):161–185, 1998.
15. R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the 15th annual ACM symposium on Theory of computing*, STOC '83, pages 193–206, New York, NY, USA, 1983. ACM Press.
16. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.
17. T. Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. *IACR Cryptology ePrint Archive*, 2014:744, 2014.
18. T. Laarhoven and B. de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. *IACR Cryptology ePrint Archive*, 2015:211, 2015.
19. A. Mariano, S. Timnat, and C. Bischof. Lock-free GaussSieve for linear speedups in parallel high performance SVP calculation. *SBAC-PAD'14 - 26th International Symposium on Computer Architecture and High Performance Computing*, 2014.
20. D. Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, pages 92–, Washington, DC, USA, 1998. IEEE Computer Society.
21. D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proceedings of the 42nd ACM*

symposium on Theory of computing, STOC '10, pages 351–358, New York, NY, USA, 2010. ACM.

22. D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms (SODA '10)*, pages 1468–1480. ACM/SIAM, Society for Industrial and Applied Mathematics, 2010.
23. B. Milde and M. Schneider. A parallel implementation of gausssieve for the shortest vector problem in lattices. In *Parallel Computing Technologies*, volume 6873 of *Lecture Notes in Computer Science*, pages 452–458. Springer Berlin Heidelberg, 2011.
24. P. Q. Nguyen and T. Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008.
25. M. Schneider, N. Gama, P. Baumann, and P. Nobach. http://www.latticechallenge.org/svp-challenge/halloffame.php.
26. K.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66:181–199, 1994.
27. X. Wang, M. Liu, C. Tian, and J. Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 1–9, New York, NY, USA, 2011. ACM.
28. F. Zhang, Y. Pan, and G. Hu. A Three-Level Sieve Algorithm for the Shortest Vector Problem. In T. Lange, K. Lauter, and P. Lisonek, editors, *SAC 2013 - 20th International Conference on Selected Areas in Cryptography*, Lecture Notes in Computer Science, pages 29–47, Burnaby, Canada, 2013. Springer Berlin Heidelberg.