Reducing Lattice Enumeration Search Trees

Mithilesh Kumar, Håvard Raddum, and Srimathi Varadharajan

***Abstract*—We revisit the standard enumeration algorithm for finding the shortest vectors in a lattice, and study how the number of nodes in the associated search tree can be reduced. Two approaches for reducing the number of nodes are suggested. First we show that different permutations of the basis vectors have a big effect on the running time of standard enumeration, and give a class of permutations that give relatively few nodes in the search tree. This leads to an algorithm called hybrid enumeration that has a better running time than standard enumeration when the lattice is large. Next we show that it is possible to estimate the signs of the coefficients yielding a shortest vector, and that a pruning strategy can be based on this fact. Sign-based pruning gives fewer nodes in the search tree, and never missed the shortest vector in the experiments we did.**

***Index Terms*—Lattices, SVP problem, enumeration, pruning**

I. INTRODUCTION

A *lattice* in $\mathbb{R}^n$ is the set of all integer combinations of m linearly independent vectors $b_1, b_2, ..., b_m$ in $\mathbb{R}^n$. In this work we assume $m = n$, but all results can easily be generalized. One of the most basic computational problems concerning lattices is the *shortest vector problem* (SVP): given a lattice basis as an input the task is to find a nonzero lattice vector of smallest norm.

It is known that SVP is NP-hard under randomized reductions [1]. With the current interest in post-quantum cryptography, lattice based cryptographic primitives are among the most promising candidates for achieving secure and efficient quantum safe crypto.

There are two main algorithmic techniques for the lattice problems. The first technique is called *lattice reduction*, and the best known algorithms are the famous LLL algorithm [2] and BKZ algorithm [3]. Both of these algorithms work by applying successive transformations to the input basis in an attempt to make the basis vectors short and as orthogonal as possible. A second and more basic approach, which is the focus of our work, is the *enumeration technique* which is simply an exhaustive search for finding the integer combinations of basis vectors whose norm is small enough.

The search can be seen as a depth-first search tree where internal nodes correspond to the partial assignments of the integer coefficients and the leaves correspond to the lattice points.

Previous results: In the 1980's Fincke, Pohst and Kannan studied how to improve the complexity of the standard algorithm for solving SVP at the time [4], [5], [6]. These algorithms are deterministic and based on exhaustive search of lattice points within a small convex set. In general, the running time of an enumeration algorithm heavily depends on the quality of the input basis. So, suitably pre-processing the input lattice using a basis reduction algorithm is essential before starting a lattice enumeration method.

Recently there have been other approaches using sieving and discrete pruning techniques, see [7], [8], [9], [10]. For a survey paper on lattice reduction algorithms, see [11].

In the 90's Schnorr, Euchner and Hörner introduced the *pruning* technique, by which these algorithms obtained substantial speedups [12], [13]. The rough idea is to prune away sub-trees where the probability of finding the desired lattice vector is small. This restricts the exhaustive search to a subset of all solutions. Although there is a chance of missing the desired vector, the probability of this is small compared to the gain in running time.

The pruning strategy was later studied more rigorously by Gama, Nguyen and Regev in [14] in 2010, introducing what they called *extreme pruning*. Very large parts of the search tree is cut away with extreme pruning. This makes the search very fast, but the probability of finding the shortest vector on a given run is very small. However, the authors show that the search tree is reduced more than the probability of finding the shortest vector, so one obtains a speed-up by just permuting the basis and repeating the process a number of times. The algorithm using extreme pruning is the fastest known, and today's state of the art when it comes to enumeration.

Our contribution: In this paper we propose two new ideas, and show their benefit in speeding up lattice enumeration. First, we propose a new enumeration algorithm called *hybrid enumeration* for computing intervals for the coefficients v_i. Second, we provide an algorithm for estimating the signs (+ or -) of the coefficients $v_1, v_2, ..., v_n$ in the lattice basis $\sum_{i=1}^{n} v_i \boldsymbol{b}_i$. Both these algorithms aims at reducing the size of search tree, thereby providing faster enumeration to find the shortest vector.

One disadvantage with the standard enumeration technique is that the algorithm depends on the computed Gram-Schmidt (GS) orthogonal basis for computing the intervals where the v_i-coefficients can be found. Once the GS orthogonal basis is computed, it fixes the order of the coefficients to be guessed.

In our paper, the hybrid enumeration takes a new approach by computing the intervals in a way that does not depend on GS orthogonalization. This means the basis vectors are not bound by any particular order and we are free to choose which of the untried coefficients v_i to guess on at any given point in the search tree. We show that dynamically changing the order of the guessed v_i's significantly lowers the number of nodes in the search tree compared to the standard enumeration algorithm.

The price to pay for this flexibility is increased work at each node of the search tree. Hence the actual time taken to enumerate a lattice using the new method may be longer than the time taken by the standard GS enumeration. Therefore we

All authors are with Simula UiB, Bergen, Norway

DOI: 10.36244/ICJ.2019.4.2

only propose to use the new enumeration technique at the nodes on the highest levels in the search tree, and then switch to standard GS enumeration for levels lower than that. This still leads to a significant reduction in the number of nodes in comparison with the standard enumeration method, depending on type of lattice and the level where we switch to standard GS enumeration.

The second technique we provide is to estimate the signs of each v_i. The main idea behind the algorithm is to exploit the dot product function which contains information about the length and angle between the basis vectors. Given two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, if the angle between them is less than 90 degrees then their sum $\boldsymbol{a}+\boldsymbol{b}$ is longer than both $\boldsymbol{a}$ and $\boldsymbol{b}$ and $\boldsymbol{a}-\boldsymbol{b}$ will be shorter than at least one of $\boldsymbol{a}$ and $\boldsymbol{b}$. To get a short vector we need to subtract one from another which implies that the sign of these vectors are opposite with respect to each other. Similarly, when the angle between them is more than 90 degrees, then addition gives a short vector, so their relative signs should be the same.

We generalize this observation on n vectors, developing a method for estimating the signs of each v_i together with a confidence measure for each estimate. We then give a pruning strategy where the interval computed for each v_i is cut down using the estimate of the sign and confidence factor. Unlike other pruning methods, this leads to a one-sided pruning where we only cut away a portion of possible v_i values where the sign is believed to be wrong. A useful fact is that our sign-based pruning can be applied on top of any other pruning strategy.

II. PRELIMINARIES

Throughout the paper, we will denote all vectors in boldface type, all matrices as capital letters, and all scalars in lower case italics. Given a linearly independent set of vectors $\{\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_n\}$ in $\mathbb{R}^n$, the lattice $\mathcal{L}$ generated by them is the set

$$\mathcal{L} = \Big\{ \sum_{i=1}^{n} v_i \boldsymbol{b}_i | v_i \in \mathbb{Z} \Big\}$$

of integer linear combination of $\boldsymbol{b}_i$'s. The set of vectors $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n\}$ is called the *lattice basis*.

The *inner product* of two vectors $\boldsymbol{a} = (a_1, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, \ldots, b_n)$ is defined as

$$\boldsymbol{a} \cdot \boldsymbol{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n.$$

The *Euclidean norm* of a vector $\boldsymbol{a}$ is defined as $\sqrt{\boldsymbol{a} \cdot \boldsymbol{a}}$ and is denoted $\|\boldsymbol{a}\|$. The vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are said to be *orthogonal* if $\boldsymbol{a} \cdot \boldsymbol{b} = 0$. Given a basis $B = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n\}$ of a lattice $\mathcal{L}$, B is said to be orthogonal if for every pair of distinct vectors $\boldsymbol{b}_i$ and $\boldsymbol{b}_j$ in B are orthogonal.

A lattice $\mathcal{L}$ contains non-zero vectors of shortest length with respect to the Euclidean norm. This parameter is denoted by $\lambda_1(\mathcal{L})$. A vector of norm $\lambda_1(\mathcal{L})$ is called a *shortest vector* of $\mathcal{L}$.

A. Gram-Schmidt orthogonalization

In general, a basis B for a lattice is not orthogonal. The Gram-Schmidt process is a method for orthogonalizing a set of vectors in an n-dimensional Euclidean space $\mathbb{R}^n$. The *projection* of a vector $\boldsymbol{a}$ onto a vector $\boldsymbol{b}$ is defined as

$$P_{\boldsymbol{b}}(\boldsymbol{a}) = \left(\frac{\boldsymbol{b} \cdot \boldsymbol{a}}{\boldsymbol{b} \cdot \boldsymbol{b}} \right) \boldsymbol{b}. \tag{1}$$

The Gram-Schmidt process can then be described via the following equations:

$$\begin{aligned}
\boldsymbol{b}_1^* &= \boldsymbol{b}_1 \\
\boldsymbol{b}_2^* &= \boldsymbol{b}_2 - P_{\boldsymbol{b}_1^*}(\boldsymbol{b}_2) \\
\boldsymbol{b}_3^* &= \boldsymbol{b}_3 - P_{\boldsymbol{b}_1^*}(\boldsymbol{b}_3) - P_{\boldsymbol{b}_2^*}(\boldsymbol{b}_3) \\
&\vdots \\
\boldsymbol{b}_n^* &= \boldsymbol{b}_n - \sum_{j=1}^{n-1} P_{\boldsymbol{b}_j^*}(\boldsymbol{b}_n)
\end{aligned}$$

The set $\{\boldsymbol{b}_1^*, \boldsymbol{b}_2^*, \ldots, \boldsymbol{b}_n^*\}$ is an orthogonal basis for the same space as that spanned by $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n\}$. More generally, for any $1 \leq i \leq n$ the subspace spanned by $\{\boldsymbol{b}_1^*, \boldsymbol{b}_2^*, \ldots, \boldsymbol{b}_i^*\}$ is the same as that spanned by $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_i\}$.

B. Projections

We can generalize the projection given in (1) to apply to a larger space. Let the space V be given by the basis $V = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$. The projection of a vector $\boldsymbol{a}$ onto the space V is then given by

$$P_V(\boldsymbol{a}) = P_{\boldsymbol{b}_1^*}(\boldsymbol{a}) + \cdots + P_{\boldsymbol{b}_k^*}(\boldsymbol{a}),$$

where the $\boldsymbol{b}_i^*$ form the orthogonal basis of V, giving a vector that lies inside the space V.

Lemma 1. *Let V be a subspace of $\mathbb{R}^n$. For any $\boldsymbol{a} \in \mathbb{R}^n$, the vector $\boldsymbol{a} - P_V(\boldsymbol{a})$ is perpendicular to every vector in V.*

Proof. We start with a basis $B_V = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$ for V and expand it to a basis for the entire space by adding some vectors $\boldsymbol{b}_{k+1}, \ldots, \boldsymbol{b}_n$. We then apply the Gram-Schmidt process to get an orthogonal basis $K = \{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*\}$ for $\mathbb{R}^n$. Then K is the concatenation of the two bases $\{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_k^*\}$ and $\{\boldsymbol{b}_{k+1}^*, \ldots, \boldsymbol{b}_n^*\}$, and by the GS property, $\{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_k^*\}$ is a basis for V. Any vector $\boldsymbol{a} \in \mathbb{R}^n$ can be written as $\boldsymbol{a} = r_1 \boldsymbol{b}_1^* + \cdots + r_n \boldsymbol{b}_n^*$ where $r_i = \frac{\boldsymbol{a} \cdot \boldsymbol{b}_i^*}{\boldsymbol{b}_i^* \cdot \boldsymbol{b}_i^*}$.

By definition we have $P_V(\boldsymbol{a}) = r_1 \boldsymbol{b}_1^* + \cdots + r_k \boldsymbol{b}_k^*$. Hence, $\boldsymbol{a} - P_V(\boldsymbol{a}) = r_{k+1} \boldsymbol{b}_{k+1}^* + \cdots + r_n \boldsymbol{b}_n^*$. Since any vector $\boldsymbol{u}$ in V is a linear combination of the vectors in $\{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_k^*\}$, we have $\boldsymbol{u} \cdot (\boldsymbol{a} - P_V(\boldsymbol{a})) = 0$. □

The following lemma provides us with a way to compute the projection of a vector onto a space V, without needing to orthogonalize the basis for V.

Lemma 2. *Let $\boldsymbol{a}$ be a vector in $\mathbb{R}^n$ and let V be a subspace of $\mathbb{R}^n$ with basis $B_V = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\}$. Let A be the matrix with $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k$ as columns. Then*

$$P_V(\boldsymbol{a}) = c_1 \boldsymbol{b}_1 + \cdots + c_k \boldsymbol{b}_k$$

where the c_i are the entries of the vector $(A^TA)^{-1}A^T\boldsymbol{a}$. In particular, the projection can be computed as $P_V(\boldsymbol{a}) = A(A^TA)^{-1}A^T\boldsymbol{a}$.

Proof. Since the columns of A are the basis vectors for V, we can write $P_V(\boldsymbol{a}) = c_1\boldsymbol{b}_1 + \cdots + c_k\boldsymbol{b}_k = A\boldsymbol{c}$ for some values c_i. By Lemma 1, the vector $\boldsymbol{a} - P_V(\boldsymbol{a})$ is orthogonal to every vector in V. Hence $A^T(\boldsymbol{a} - P_V(\boldsymbol{a})) = \boldsymbol{0}$ since the matrix/vector multiplication is just taking the inner product of basis vectors with $(\boldsymbol{a} - P_V(\boldsymbol{a}))$. Substituting for $P_V(\boldsymbol{a})$, we get $A^T\boldsymbol{a} - A^TA\boldsymbol{c} = \boldsymbol{0}$ which implies that $\boldsymbol{c} = (A^TA)^{-1}A^T\boldsymbol{a}$. Hence, $P_V(\boldsymbol{a}) = A(A^TA)^{-1}A^T\boldsymbol{a}$. □

C. The standard enumeration algorithm

Let $\mathcal{L}$ be a lattice whose shortest vector v is unique up to the sign. Assume we are given the basis $\{\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_n\}$ of $\mathcal{L}$ and an upper bound R on $\lambda_1(\mathcal{L})$ such that we need to find all vectors $\boldsymbol{w}$ in the lattice $\mathcal{L}$ that satisfy $\|\boldsymbol{w}\| \leq R$.

The shortest vector $\boldsymbol{s} \in \mathcal{L}$ can be written as $\boldsymbol{s} = v_1\boldsymbol{b}_1 + v_2\boldsymbol{b}_2 + ... + v_n\boldsymbol{b}_n$ where the $v_i's$ are unknown integers and $\boldsymbol{b}_i = \sum_{j=1}^{i-1}\mu_{i,j}\boldsymbol{b}_j^*$, where $\mu_{i,j} = (\boldsymbol{b}_i \cdot \boldsymbol{b}_j^*)/(\boldsymbol{b}_j^* \cdot \boldsymbol{b}_j^*)$ are the Gram-Schmidt coefficients. Our goal is to find $\boldsymbol{s}$.

To find $\pm\boldsymbol{s}$, the enumeration goes through an enumeration tree formed by the subspace spanned by the vectors whose norm is at most R. The enumeration tree is a depth first search tree of depth n. Each internal node in the tree is associated with a particular v_i and each outgoing edge represents an assignment of an integer value (obtained from a range) to v_i. In particular the root of the tree is the zero vector, while the leaves are all the vectors of $\mathcal{L}$ whose norm is at most R.

At any node, the enumeration algorithm selects an index i not yet branched for, obtains a set of integers (interval range) I_i for the possible values v_i can take and for each integer $t \in I_i$ the algorithm calls itself recursively to compute the interval for the next level. The length bound here remains constant throughout the algorithm. For $1 \leq k \leq n$, the following inequality (see [14]) needs to be satisfied, essentially defining the interval I_k:

$$\left(v_k + \sum_{i=k+1}^{n}\mu_{i,k}v_i\right)^2\|b_k^*\|^2 + \sum_{j=k+1}^{n}\left(v_j + \sum_{i=j+1}^{n}\mu_{i,j}v_i\right)^2\|b_j^*\|^2 \leq R^2 \quad (2)$$

By the inequality above, for each $1 \leq k \leq n$ the interval range I_k for v_k can be obtained if v_j is known for each $j \in \{k+1, k+2, ..., n\}$. This implies that in the enumeration algorithm, the indices i can only be chosen in the order starting from $n, n-1, ..$ down to 1. In the rest of the paper we refer to the root node of the search tree being at level n, the second highest level being level $n-1$, etc. That is, if a node is at level l in the search tree, then only the coefficient v_l can be selected for branching at that node.

III. HYBRID ENUMERATION

In this section we study how permutations of the basis vectors of a lattice affects the running time of enumeration.

nodes in search tree	BKZ-10	BKZ-20
minimum	60.934.596	4.059.025
average	424.300.658	52.886.123
maximum	1.180.735.200	194.214.522
std. deviation	361.710.571	40.202.374

TABLE I: Number of nodes to fully enumerate the BKZ-reduced SVP40 challenge lattice for 20 random permutations of the basis. The number of nodes in a search tree is highly dependent on the particular permutation.

Based on this we present a good strategy for selecting an order of the basis vectors that results in relatively small search trees when doing enumeration. This can help speed up extreme pruning, by only selecting permutations that give small search trees when iterating the extremely pruned enumeration runs.

A. Variations in Enumeration Complexity from Basis Permutations

As far as we know, there have been no studies of how the complexity of standard enumeration varies when the vectors in the input basis are permuted. To motivate the work that follows, we first present the results of some experiments showing that the number of nodes in the search tree when doing full enumeration is highly sensitive with respect to the order of the basis vectors.

The lattice we use for the demonstration is Darmstadt's SVP40 challenge [15], generated from seed 0. The experiment was done as follows: First, we ran two BKZ-reductions on the SVP40 lattice, one with block size 10 and one with block size 20. Then we did full enumeration of each of the two BKZ-reduced lattices, counting the number of nodes in the search tree. Next we randomized the two BKZ-reduced bases 20 times each, and ran full enumeration on all of them. The average number of nodes in the search trees for the randomized bases are shown in Table I, together with the maximum and minimum numbers observed and the standard deviation.

From Table I we see that the order of the basis vectors has a big impact on the size of the enumeration search tree. The standard deviation is of similar size as the average, showing that the sizes of the search trees vary greatly with the permutation.

Another interesting thing we observed is that the order of the reduced basis as given straight out of BKZ is particularly good for enumeration. Enumerating the SVP40 challenge with the basis order given by BKZ-10 gives a tree with 5.968.085 nodes, and the order given by BKZ-20 gives a tree with 1.232.737 nodes, significantly smaller than the numbers observed for any of the random permutations.

B. Intervals for coefficients

Given a length bound R, basic enumeration will search exhaustively for all lattice vectors of length less than or equal to R. Assume that $\boldsymbol{s} = v_1\boldsymbol{b}_1 + v_2\boldsymbol{b}_2 + \ldots + v_n\boldsymbol{b}_n$ is a vector such that $\|\boldsymbol{s}\| \leq R$. Before the enumeration can start, the μ-matrix $[\mu_{i,j}]$ of Gram-Schmidt coefficients and the orthogonal basis vectors $\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*$ must be computed. The μ-matrix is dependent on the particular order of the basis vectors, and

once it is computed this order remains fixed throughout the standard enumeration routine.

The actual enumeration starts by computing an interval I_n such that $\|\boldsymbol{s}\| \leq R$ implies $v_n \in I_n$. The algorithm then fixes an integer value in I_n for v_n, and based on the choice computes an interval I_{n-1} such that $\|\boldsymbol{s}\| \leq R$ implies $v_{n-1} \in I_{n-1}$. Then an integer is selected from I_{n-1} and assigned to v_{n-1}, and the interval where v_{n-2} must be found is computed. This continues until a selection for v_1 can be made, in which case we find a lattice vector with length less than R, or until an interval I_j that contains no integers is computed.

Intervals are computed recursively in the order $I_n, I_{n-1}, \ldots, I_2, I_1$, and all values from all intervals must be tried to do a complete search that guarantees that a shortest vector will be found. In the following, we denote the length of an interval I_i by $|I_i|$.

Basic enumeration assumes the μ-matrix is computed once and for all before actual enumeration starts, but this is not strictly necessary. We can set every basis vector $\boldsymbol{b}_i$ in the basis as the last one, recompute the μ-matrix, and find the interval of possible coefficients for I_i. Doing this allows us to make a choice of which vector to first fix the coefficient for. For instance, we may select the basis vector giving the shortest interval as the first one to branch for.

This strategy can be generalized and done at any point during enumeration: Assume v_j for $j \in J \subseteq \{1, \ldots, n\}$ have been fixed, where $|J| = k$. All remaining basis vectors $\boldsymbol{b}_i$ for $i \in (\{1, \ldots, n\} \setminus J)$ can be tried by placing them successively in position $n-k$ in the basis. The μ-matrix and the coefficient intervals are re-computed for every choice, and the vector giving the shortest interval is selected as the next one to branch for. In this way we may dynamically change the order of which basis vector to branch for, while the enumeration algorithm is running.

Remark: To compute the smallest interval at a given node, we do not need to re-compute the full μ-matrix. We only need to re-compute the entries in μ from the point where we have changed the order of the basis vectors. For example, if we are computing the interval for v_j, only the rows of μ with indices higher than j needs to be updated when setting $\boldsymbol{b}_j$ last.

C. Strategy for selecting order for basis vectors

The strategy we use for choosing the order of basis vectors to branch for follows a greedy approach: We always choose the next v_i to try as the one with the shortest interval I_i. The rationale for this strategy can be explained via the following lemma, basically saying that the interval for one v_i shortens, when more of the other coefficients are fixed.

Lemma 3. *Let $J_1 \subseteq J_2 \subseteq (\{1, \ldots, n\} \setminus \{i\})$. Let $I_i(J_1)$ be the interval for v_i after values of $v_j, j \in J_1$ have been fixed, and let $I_i(J_2)$ be the interval for v_i after some additional v_j's, $j \in J_2 \setminus J_1$ have been fixed. Then $|I_i(J_1)| \geq |I_i(J_2)|$.*

Proof. From Equation (2) we see that the length of $I_i(J_1)$ is determined by the sum

$$\sum_{j=k+1}^{n} \left(v_j + \sum_{i=j+1}^{n} \mu_{i,j} v_i \right)^2 \left\| b_j^* \right\|^2, \tag{3}$$

while the center of the interval is determined by

$$\sum_{i=k+1}^{n} \mu_{i,k} v_i.$$

When we branch in an unspecified order, (3) can be written as

$$\sum_{j \in J_1} t_j^2,$$

where the t_j's are terms decided by the specific order in which the indices in J_1 were chosen. The larger this sum becomes, the smaller $|I_i(J_1)|$ will be. The terms in the sum are all positive, so expanding with the extra terms to create the sum $\sum_{j \in J_2} t_j^2$ before branching for v_i can only decrease the length of I_i. Hence $|I_i(J_1)| \geq |I_i(J_2)|$. □

Lemma 3 shows that the longer we wait to select a particular v_i to branch for, the shorter its interval I_i will become. The idea for the branching strategy is that intervals that are long when few v_j's have been selected will become short by the time the algorithm is forced to branch on them. This will lead to relatively small search trees.

One way to more easily see this is in the case when one I_i becomes empty after fixing the v_j's for $j \in J$, for some J. Say the branching order has been fixed from the start, the values of v_j, $j \in J$ have been fixed, and that I_i is empty, but v_i is only to be branched for after another 10 v_k's have been fixed. Even though it is clear (if we compute I_i) that all choices of values for the v_k's will lead to a dead end, the traditional enumeration algorithm will try all of them before backtracking away from this sub-tree. By always selecting the next v_i to branch for as the one with the shortest interval, v_i will be selected as soon as $|I_i| = 0$ (the shortest length possible), and backtracking into the v_j's where $j \in J$ will start immediately.

D. Cost vs effect for minimizing intervals

The drawback of checking which of the remaining indices to branch for is the extra work done in each node. If we compute an interval I_i using the μ-matrix of Gram-Schmidt coefficients, we in general have to recompute the μ-matrix as part of the process. The complexity for computing this matrix for one index is $\mathcal{O}(n^3)$ multiplications, and doing this for every remaining index not yet branched for gives overall complexity of $\mathcal{O}(n^4)$ in each node.

Computing an interval I_i using the projection method involves inverting a matrix, which also has complexity $\mathcal{O}(n^3)$. Repeating for all unbranched indices again gives an overall complexity of $\mathcal{O}(n^4)$ for the work done in each node. These complexities are quite high considering they have to be done for each node. However, they are still polynomial and the number of nodes in a search tree is super-exponential in n, so if the reduction in the number of nodes is big enough it is still worthwhile.

As we saw in Table I, the number of nodes in a search tree without using any minimizing strategy depends heavily on the order of the basis vectors. The order of the basis vectors does

not matter when using the minimizing strategy as the vectors will be sorted as part of the enumeration routine. Hence it is hard to say anything in general about how large the effect of minimizing intervals will have, since it depends on how "lucky" the initial order of the vectors is.

We have tested the minimizing strategy on random lattices of relatively small dimensions ($10 \leq n \leq 20$), and compared the number of nodes in these search trees with the number of nodes in the search trees using standard enumeration. The minimizing strategy indeed leads to search trees with much fewer nodes, on the average the reduction is approximately by a factor n for the small dimensions we looked at. As the increase in workload in each node is by a factor $\mathcal{O}(n^4)$, applying the minimizing strategy in every node is not worth the extra effort.

E. Hybrid enumeration

When values for many v_j's have been assigned (for $j \in J$), the effect of minimizing intervals for the relatively few remaining indices in $\{1, \ldots, n\} \setminus J$ is small. On the other hand, applying the minimizing strategy on the very first v_j's to be fixed has a much greater effect. The number of large sub-trees rooted high up in the full tree when no ordering strategy is applied, become significantly smaller when minimizing intervals. In the extreme case of some interval becoming empty, the whole sub-tree gets pruned away.

Thus we propose to only apply the minimizing strategy on the relatively few nodes at the highest levels of the search tree. This has the benefit of a relatively low cost for a high effect. We call enumeration with the strategy of minimizing intervals for the first few levels of the tree for *hybrid enumeration*.

One parameter for hybrid enumeration is the level in the tree where we switch from finding an optimal order based on minimizing intervals to classic enumeration where the basis is in some given and fixed order. We call this parameter the *switch level*.

More precisely, when we reach a node at the switch level we do the following: We compute the interval lengths for remaining indices one last time, and permute the remaining basis vectors according to these lengths. Indices with the shortest intervals will be branched for first. Then we do normal enumeration for the sub-tree rooted at the current node, using this fixed order for the whole sub-tree. Pseudo code for hybrid enumeration is given in Algorithm 1.

For $B = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$, we regard the root node of the tree (at the top) to be at level n, and the short vectors of $\mathcal{L}(B)$ will be found at level 0. Note that we can run basic enumeration of the lattice by calling HybridEnumerate$(B, R, n+1, n)$. Calling HybridEnumerate(B, R, n, n) will also run basic enumeration, but the basis is first permuted according to the strategy of minimizing intervals. This makes it easy to compare the benefit of using hybrid enumeration over basic enumeration.

Algorithm 1 HybridEnumerate(B, R, sl, l)

Input: The basis vectors $B = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}$ of a lattice $\mathcal{L}$, a length bound R, the current level l, and the switch level sl.
Output: All vectors $\boldsymbol{s} \in \mathcal{L}$ with $\|s\| \leq R$

if $l > sl$ **then**
  $I_i \leftarrow$ shortest interval for $\boldsymbol{b}_i \in B$
  for $v_i \in I_i$ **do**
    $r \leftarrow$ min. length added to $\|\boldsymbol{s}\|$ due to choice of v_i
    HybridEnumerate$(B \setminus \{\boldsymbol{b}_i\}, R - r, sl, l - 1)$
  end for
end if
if $l = sl$ **then**
  Compute intervals $I_j, \forall \boldsymbol{b}_j \in B$
  Sort B according to $|I_j|$, basis vectors on bottom of B has shortest intervals
  HybridEnumerate$(B, R, sl, l - 1)$
end if
if $l < sl$ **then**
  Run standard enumeration on B with length bound R
end if

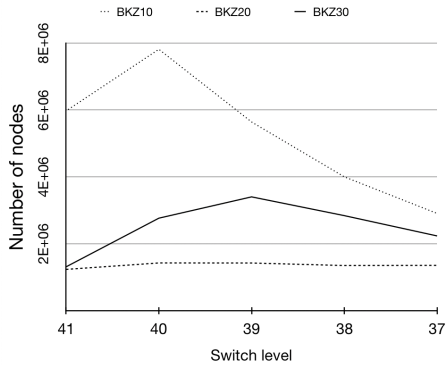
F. Experiments

We have tested hybrid enumeration on several of the SVP challenges of [15] and counted the number of nodes hybrid enumeration gives for different switch levels. The lattice bases were first reduced by running BKZ-β on them, for $\beta \in \{10, 20, 30\}$. For each reduced lattice, we ran hybrid enumeration with switch levels ranging from $n+1$, equivalent to standard enumeration, to $n-4$, counting the nodes in each search tree. The results are shown as plots in Figure 1.

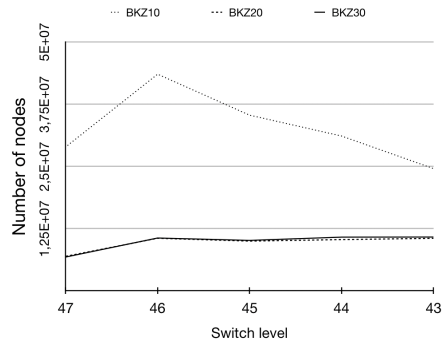
We see a few trends from these plots. First, there is not much difference between BKZ-20 and BKZ-30 regarding the quality of the bases. Both of them give search trees with approximately the same number of nodes, and applying the strategy of minimizing intervals does not change this by much. Also, the order of the basis vectors given by hybrid enumeration yields search trees approximately as small as the order given by BKZ. This is in contrast to the random orders used for computing the numbers in Table I, that shows a large increase in the number of nodes. Hence the strategy of sorting the basis vectors according to interval lengths clearly is a good approach.

For the BKZ-10 reduced bases, we see a much bigger effect. First, we see that BKZ-10 gives a significantly weaker reduction than BKZ-20 or BKZ-30, leading to larger enumeration search trees. The order as given by BKZ-10 is still good for enumeration, and doing one initial sorting of the basis according to interval lengths (switch level n) increases the search tree. However, lowering the switch level has a clear impact and significantly reduces the number of nodes in the search tree, beyond the low number of nodes given by the initial BKZ-order.

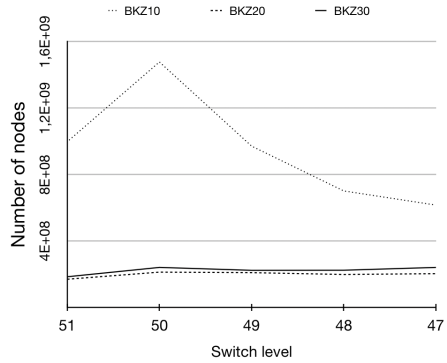
Of course, what matters in the end for a lattice enumeration algorithm is its complexity, measured in the actual time taken. We recorded the times taken in all the experiments, to see if the extra work done in the nodes at and above the switch level is worth the effort. For the enumeration of BKZ-20 and BKZ-30 reduced bases it is clearly not worth the effort as the number of nodes stay almost the same for the various switch



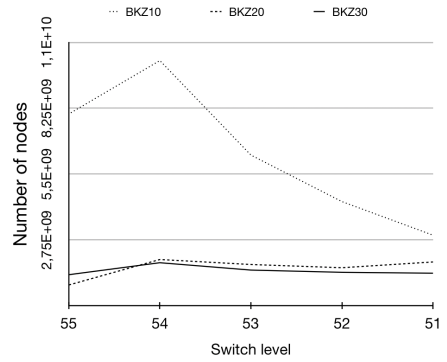
(a) SVP40



(b) SVP46



(c) SVP50



(d) SVP54

Fig. 1: Number of nodes using hybrid enumeration on lattice bases pre-processed with BKZ-β for $\beta \in \{10, 20, 30\}$.

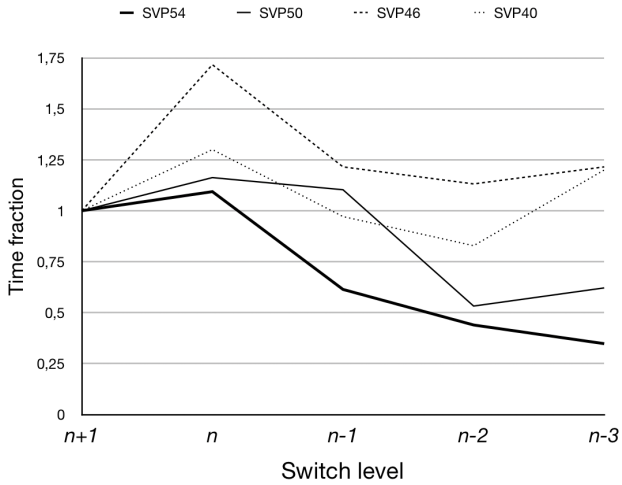


Fig. 2: Fraction of time taken for doing full hybrid enumeration on BKZ-10 reduced lattice bases, compared to time taken for standard enumeration.

levels. For enumerating the lattices only reduced by BKZ-10, there is a significant decrease in the number of nodes as the switch level decreases. Is this enough to weigh up for the $\mathcal{O}(n^4)$ work done in each node at and above the switch level?

In Figure 2 we have plotted the fraction of time taken for enumerating the four lattices we have used, compared to standard enumeration (switch level $n + 1$). The typical time taken for running these instances ranged from about one minute for the SVP46 basis, to about 24 hours for the SVP54, both pre-processed with BKZ-10. The experiments were run on a DELL computer running Linux with two 2.8 GHz AMD EPYC 7451 24-Core processors and 188 GB of RAM.

We observe a few things from Figure 2. First, except for SVP46, the time it takes to do hybrid enumeration is less than the time for doing standard enumeration, for some switch level. Using switch level n gives an increase in time because of an increase in the number of nodes. For deeper switch levels the reduction in the number of nodes is actually worth the extra work done in the few nodes at the top. Second, for the bigger lattices, the time saving is largest, with full hybrid enumeration for SVP54 using switch level 51 only taking 34.8% of the time it takes to do full standard enumeration. Third, we also see there is an optimal switch level. For SVP40 and SVP50, hybrid enumeration takes longer for switch level $n - 3$ than for $n - 2$, even though the number of nodes is less for switch level $n - 3$. The reduction in the number of nodes is then not worth the extra work for all nodes on level $n - 3$.

Figure 2 is only for BKZ-10 reduced bases, and for better BKZ reductions we do not demonstrate an improvement in running time. However, the lattices we are able to do full enumeration for in practice have dimensions in the range 40 - 60, and a block size of 20 and 30 when running BKZ is then a large portion of that. We see in the plots that there is not much difference between BKZ-20 and BKZ-30 reduced bases, and there is hardly any improvement to be done for these cases. They appear to be quite optimal from the start.

We conjecture that for higher dimensions, like $n = 150$, BKZ-30 would not give an optimally reduced basis, and that hybrid enumeration then would show the same improvements as we see with the BKZ-10 reduced bases in our experiments. All in all, we claim that if one wants to do full enumeration on large lattices that are not optimally reduced, then hybrid enumeration will be faster than standard enumeration.

IV. Sign-based pruning

Going back to the expansion of a shortest vector in terms of the basis vectors $\boldsymbol{s} = \sum_{i=1}^{n} v_i \boldsymbol{b_i}$, an enumeration algorithm computes possible values for each coefficient v_i. The equation for computing the coefficients v_i indicates that the range I_i for v_i is likely to contain both positive and negative values. As both $\boldsymbol{s}$ and $-\boldsymbol{s}$ are shortest vectors, we are content in finding either of those. If we could a priori *know* the sign of these integers (that is whether $v_i \leq 0$ or $v_i \geq 0$), we could discard appropriate values from I_i, making the enumeration tree smaller. Effectively, this would provide us with another strategy for pruning. In this section, we describe an algorithm for making educated guesses for the signs of these coefficients and how to use them for pruning. In the following we assume that the lattice basis has been reduced, and that the lengths of the basis vectors are of low variance.

A. Sign-estimation

First we show how to compute the signs of the coefficients of the shortest vector when the dimension on the given lattice is only 2. Let us consider a lattice in 2 dimensions with basis vectors $\{\boldsymbol{b}_1, \boldsymbol{b}_2\}$. If $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are obtuse to each other (i.e. the angle between them is more than 90°), then a shortest vector $\boldsymbol{s} = v_1\boldsymbol{b}_1 + v_2\boldsymbol{b}_2$ can only be obtained if the signs of v_1 and v_2 are the same. Similarly, if they are acute (angle less than 90°) to each other, a shortest vector can only be obtained if the signs of v_1 and v_2 are opposite to each other. It is easy to see this, as a (positive) sum of two vectors pointing in approximately the same direction can only increase in length.

To extend this observation to higher dimensions we define the dot-product matrix M, where $M_{ij} = \boldsymbol{b}_i \cdot \boldsymbol{b}_j$. Two vectors have a positive dot product when the angle between them is less than 90° and a negative dot product when the angle between them is larger than 90°. Moreover, the magnitude of $\boldsymbol{b}_i \cdot \boldsymbol{b}_j$ relative to the product of the lengths of $\boldsymbol{b}_i$ and $\boldsymbol{b}_j$ is a measure of how parallel or anti-parallel $\boldsymbol{b}_i$ and $\boldsymbol{b}_j$ are.

The algorithm for computing the sign of coefficients is shown in Algorithm 2. The algorithm computes a vector $\boldsymbol{\sigma}$ of signs with entries $+1$ or -1. The sign for the coefficients v_i are computed one at a time, and the estimated sign of v_i depends on the signs of coefficients that have already been computed. Intuitively, the algorithm compares each basis vector with some reference vector to estimate the sign of the corresponding coefficient.

The sign of the first basis vector $\boldsymbol{b}_1$ is set to be positive by default, so $\sigma_1 = +1$. This is without loss of generality since both $\boldsymbol{s}$ and $-\boldsymbol{s}$ are shortest vectors and at least one of them must have non-negative v_1. The vector $\boldsymbol{b}_1$ is set as the reference vector $\boldsymbol{a}$ for the next basis vector. The first row of M contains the inner product of $\boldsymbol{b}_1 (= \boldsymbol{a})$ with all the other basis vectors. The basis vector with the largest inner product in absolute value is both a relatively long vector, and makes an angle close to 0° or 180° with $\boldsymbol{b}_1$. Let $\boldsymbol{b}_i$ be this basis vector. Then the sign of v_i is set to -1 if $M_{1i} > 0$, otherwise σ_i is set to $+1$. The reference vector is updated to $\boldsymbol{a} = \boldsymbol{a} + \sigma_i \boldsymbol{b}_i$.

Now we want to find a basis vector which is *most* parallel or anti-parallel to $\boldsymbol{a}$. For this we look at the largest entry in the vector $D = M_1 + \sigma_i M_i$, where M_1 is the top row of M and M_i is the i'th row. The largest entry in absolute value in D (except for index 1 and i) indicates the third vector, say $\boldsymbol{b}_j$, to estimate the sign for. If $D_j > 0$ then $\sigma_j = -1$, and if $D_j \geq 0$, $\sigma_j = +1$. The vector $\sigma_j \boldsymbol{b}_j$ is added to $\boldsymbol{a}$ and D is updated to $D = D + \sigma_j M_j$. The algorithm continues like this until all basis vectors have had their signs estimated.

Algorithm 2 ComputeSign(B)

Input: The basis vectors B of the lattice $\mathcal{L}$.
Output: A vector $\boldsymbol{\sigma}$ that contains the estimated sign of each coefficient v_i in $\boldsymbol{s} = \sum_i v_i \boldsymbol{b}_i$ where $\boldsymbol{s}$ is a shortest vector, and a vector $\boldsymbol{\gamma}$ of real values indicating confidence for each estimate.

Compute dot-product matrix M such that $M_{ij} = \boldsymbol{b}_i \cdot \boldsymbol{b}_j$.
Initialize $D := M_1$ where M_1 is the top row of M.
Set $\sigma_1 = +1$ and $\gamma_1 = 1$.
Set reference lattice vector $\boldsymbol{a} := \boldsymbol{b}_1$
Set the counter $n_s := 1$.
while $n_s \leq n$ **do**
  Let i be the index of $\max\{|D_j| | j$ is not among already fixed signs$\}$.
  if $D_i > 0$ **then**
    Set $\sigma_i = -1$
  else
    Set $\sigma_i = +1$
  end if
  Set $\boldsymbol{a} = \boldsymbol{a} + \sigma_i \boldsymbol{b}_i$
  Set $D = D + \sigma_i M_i$
  Compute $\gamma_i = |\frac{\boldsymbol{a} \cdot \boldsymbol{b}_i}{\|\boldsymbol{a}\| \|\boldsymbol{b}_i\|}|$
  Set $n_s = n_s + 1$
end while

The signs computed in Algorithm 2 are not necessarily correct for a shortest vector. For each variable v_i, we compute a number $0 \leq \gamma_i \leq 1$ to denote how confident we are that the computed σ_i is correct. When $\gamma_i = 1$ we are certain that the corresponding σ_i is correct and $\gamma_i = 0$ means we have no knowledge whether the sign for v_i should be positive or negative. We compute the confidence values of the estimated signs as follows: Let $J \subset \{1, \ldots, n\}$ be the set of indices for which values have been fixed and let the reference vector be $\boldsymbol{a} = \sum_{j \in J} \sigma_j \boldsymbol{b}_j$. Then the confidence value for the σ_i estimate is given as $\gamma_i = |\frac{\boldsymbol{a} \cdot \boldsymbol{b}_i}{\|\boldsymbol{a}\| \|\boldsymbol{b}_i\|}|$.

The intuition behind this measure for confidence is that if two vectors are very close to being parallel, then having the same sign on the coefficients of these vectors will always lead to a longer vector as their sum, pointing approximately in the same direction as the other two. In order to be part of a short

Lattice	Pre-processing	node fraction	shortest vector found
SVP40	BKZ10	0.670	yes
SVP40	BKZ20	0.745	yes
SVP40	BKZ30	0.665	yes
SVP46	BKZ10	0.682	yes
SVP46	BKZ20	0.750	yes
SVP46	BKZ30	0.800	yes

TABLE II: Measure of effect of sign-based pruning. The node fraction is the number of nodes in pruned search tree compared to the number of nodes in the full enumeration search tree.

vector $\boldsymbol{s}$, the other basis vectors must be able to offset this long vector. If the signs of the coefficients are opposite, a sum of the two approximately parallel basis vectors would be much shorter. It is easier to sufficiently offset a short vector than a long one, in order to find the shortest vector overall.

When two vectors are close to being parallel then $\frac{\boldsymbol{a}\cdot\boldsymbol{b}_i}{\|\boldsymbol{a}\|\|\boldsymbol{b}_i\|}$ is close to being 1, and when two vectors are close to being anti-parallel $\frac{\boldsymbol{a}\cdot\boldsymbol{b}_i}{\|\boldsymbol{a}\|\|\boldsymbol{b}_i\|}$ is close to being -1. In both cases $\gamma_i \approx 1$.

On the other hand, when $\boldsymbol{a}$ and $\boldsymbol{b}_i$ are close to orthogonal (i.e. $\boldsymbol{a}\cdot\boldsymbol{b}_i \approx 0$), then $\boldsymbol{a}+\boldsymbol{b}_i$ and $\boldsymbol{a}-\boldsymbol{b}_i$ will be of almost equal lengths, and we can only to a little extent distinguish which of the two cases that will be most easily offset by the other basis vectors. The confidence value will therefore be close to 0 in this case.

We now turn to how we use the confidence values to prune intervals in the search tree.

B. Pruning intervals based on sign estimation

We can use the sign estimations and their confidence values to shorten the intervals computed for enumeration, while still maintaining a high probability we do not prune away all shortest vectors.

For a node in the search tree where possible values for v_i are tried, let I_i be the interval computed for v_i. Let $I_i^+ := I_i \cap [0,\infty)$ and $I_i^- := I_i \cap (-\infty, 0]$. For an interval $I := [l, m]$ and a positive number $\alpha \in \mathbb{R}$, let us define the interval αI to be $[\alpha l, \alpha m]$. If $\sigma_i = -1$, then I_i is pruned to $I_i = (1-\gamma_i)I_i^+ \cup I_i^-$. If $\sigma_i = +1$, then I_i is pruned to $I_i = (1-\gamma_i)I_i^- \cup I_i^+$. In other words, we cut away a portion of the interval where we believe a correct value for v_i will not be found. The portion cut away is proportional to the confidence we have in our estimate.

An advantage of this pruning strategy is that it can be put on top of any other pruning strategy. The sign-based pruning does not depend on how the intervals are computed. This pruning strategy reduces the search tree as long as the given intervals are non-empty and cuts away integer values that are opposite in sign to the predicted sign.

C. Experiments

We have used a few of the SVP challenge lattices to test the sign-based pruning strategy. We measured both the reduction in the number of nodes in the search tree, and whether the pruning failed to find the shortest vector. The results are summarized in Table II.

What we see in Table II is that in the experiments we never failed to find the shortest vector, and that the reduction in the number of nodes is by a modest but still significant fraction. One explanation for this is that we cut away the ends of the intervals, which only takes away small subtrees from the whole enumeration tree. The v_i-values found at the ends of the intervals are those that consume much of the length limit R when selected, probably quickly leading to dead ends anyway. Cutting away these values may not prune away very large parts of the search tree. Still, it is worthwhile to apply the sign-based pruning as it costs practically nothing in terms of extra complexity. The actual run times are cut down by almost the same fraction as the reduction in the number of nodes.

V. CONCLUSIONS

Public key encryption schemes based on lattices are one of the most promising approaches for achieving quantum safe crypto, and it is important to understand the hardness of the SVP problem on which they are based. Lattice enumeration plays a central role in the best known methods for solving SVP, so studying how to speed up lattice enumeration is important for assessing the security of lattice-based encryption. In this paper we have explored two different ideas for speeding up lattice enumeration.

First we looked at how permutations of the basis vectors of a lattice affect the running time of the standard enumeration algorithm. We demonstrate that the particular order of the basis vectors have a big impact on the number of nodes in the search tree and the running time. Next we identified particular permutations that give relatively small search trees. Dynamically finding the best permutations has a high cost on its own. However, if the lattice dimension is big enough and the pre-processing does not leave a strongly reduced basis, it is well worth the effort to apply the strategy in the relatively few nodes at the top of the search tree. We call this type of enumeration for hybrid enumeration.

Secondly, we looked at the possibility of estimating the signs of the coefficients giving a shortest vector. We can only estimate the signs with some degree of confidence, but the estimates and the confidence values lead directly to a pruning strategy. Unlike other pruning strategies that cuts away values from both ends of the interval where a coefficient v_i can be found, sign-based pruning only cuts values from one side of the interval (the side where the values have the "wrong" sign). Sign-based pruning can therefore be applied together with any other pruning strategy one may use.

The experiments of sign-based pruning give a reduction in the number of nodes in the search tree compared to standard enumeration, but the reduction is not great. However, we never failed to find the shortest vector using sign-based pruning. This may indicate that the pruning we employed from the confidence measure is not aggressive enough, and that larger parts of the intervals could be cut away without sacrificing too much accuracy in solving the SVP. Further studies of sign-based pruning is topic for future work.

REFERENCES

[1] M. Ajtai, "The Shortest Vector Problem in L2 is NP-hard for Randomized Reductions (Extended Abstract)," in *Proceedings of the Thirtieth 9 Annual ACM Symposium on Theory of Computing*, ser. STOC '98. New York, NY, USA: ACM, 1998, pp. 10–19, DOI: 10.1145/276698.276705.

[2] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, pp. 515–535, 1982, DOI: 10.1007/BF01457454.

[3] C. P. Schnorr, "A hierarchy of polynomial time lattice basis reduction algorithms," *Theoretical Computer Science*, vol. 53, pp. 201–224, 1987, DOI: 10.1016/0304-3975(87)90064-8.

[4] U. Fincke and M. Pohst, "A procedure for determining algebraic integers of given norm," in *Proceedings of the European Computer Algebra Conference on Computer Algebra*, ser. EUROCAL '83. London, UK, UK: Springer-Verlag, 1983, pp. 194–202, DOI: 10.1007/3-540-12868-9_103.

[5] R. Kannan, "Improved algorithms for integer programming and related lattice problems," in *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '83. New York, NY, USA: ACM, 1983, pp. 193–206, DOI: 10.1145/800061.808749.

[6] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," *Mathematics of Computation*, pp. 463 – 471, 1985, DOI: 10.2307/2007966.

[7] T. Laarhoven and A. Mariano, "*Progressive lattice sieving," in Post-Quantum Cryptography*, T. Lange and R. Steinwandt, Eds. Springer International Publishing, 2018, pp. 292–311, DOI: 10.1007/978-3-319-79063-3_14.

[8] M. Schneider, "Sieving for shortest vectors in ideal lattices," in *Progress in Cryptology – AFRICACRYPT 2013*, A. Youssef, A. Nitaj, and A. E. Hassanien, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 375–391, DOI: 10.1007/978-3-642-38553-7_22.

[9] Y. Aono and P. Q. Nguyen, "Random sampling revisited: Lattice enumeration with discrete pruning," in *Advances in Cryptology – EUROCRYPT 2017*, J.-S. Coron and J. B. Nielsen, Eds. Springer International Publishing, 2017, pp. 65–102, DOI: 10.1007/978-3-319-56614-6_3.

[10] Y. Aono, P. Q. Nguyen, and Y. Shen, "Quantum lattice enumeration and tweaking discrete pruning," in *Advances in Cryptology – ASIACRYPT 2018*, T. Peyrin and S. Galbraith, Eds. Springer International Publishing, 2018, pp. 405–434, DOI: 10.1007/978-3-030-03326-2_14.

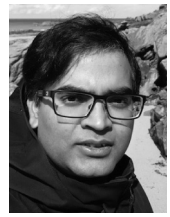
[11] P. Q. Nguyen, "Lattice reduction algorithms: Theory and practice," in *Advances in Cryptology – EUROCRYPT 2011*, K. G. Paterson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 2–6, DOI: 10.1007/978-3-642-20465-4_2.

[12] C. P. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Math. Program.*, vol. 66, no. 2, pp. 181–199, Sep. 1994, DOI: 10.1007/BF01581144.

[13] C. P. Schnorr and H. H. Hörner, "Attacking the chor-rivest cryptosystem by improved lattice reduction," in *Proceedings of the 14th Annual International Conference on Theory and Application of Cryptographic Techniques*, ser. EUROCRYPT'95. Berlin, Heidelberg: Springer-Verlag, 1995, pp. 1–12, DOI: 10.1007/3-540-49264-X_1.

[14] N. Gama, P. Q. Nguyen, and O. Regev, "Lattice enumeration using extreme pruning," in *Advances in Cryptology – EUROCRYPT 2010*, H. Gilbert, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 257–278, DOI: 10.1007/978-3-642-13190-5_13.

[15] N. Schneider and N. Gama, "SVP Challenge," https://www.latticechallenge.org/svp-challenge/index.php.



Mithilesh Kumar was born in India. He has done masters in Physics from IIT Kanpur and masters in computer science from CMI Chennai. In 2014 he started his PhD studies in algorithms at the University of Bergen, and he received his PhD degree from this university in 2017. His primary interests are graph theory, algorithms, lattices and quantum computation.

Håvard Raddum was born in Bergen, Norway and received his master degree from the Department of mathematics at the University of Bergen in 1999. In 2005 he received a PhD in cryptography from the University of Bergen. He has done research on the cryptanalysis of ciphers, and is interested in algebraic aspects of cryptographic primitives. Håvard currently leads the cryptography research group at Simula UiB.

Srimathi Varadharajan is a PhD student at Simula UiB, working on the mathematics of cryptography. She got a master degree in mathematics from Royal Holloway University of London in 2015.