# Lattice Enumeration Using Extreme Pruning

Nicolas Gama[1], Phong Q. Nguyen[2], and Oded Regev[3]

[1] GREYC and ENSICAEN, France
[2] INRIA and ENS, France
http://www.di.ens.fr/~pnguyen/
[3] Blavatnik School of Computer Science, Tel Aviv University, Israel
http://www.cs.tau.ac.il/~odedr/

**Abstract.** Lattice enumeration algorithms are the most basic algorithms for solving hard lattice problems such as the shortest vector problem and the closest vector problem, and are often used in public-key cryptanalysis either as standalone algorithms, or as subroutines in lattice reduction algorithms. Here we revisit these fundamental algorithms and show that surprising exponential speedups can be achieved both in theory and in practice by using a new technique, which we call *extreme pruning*. We also provide what is arguably the first sound analysis of pruning, which was introduced in the 1990s by Schnorr *et al.*

## 1 Introduction

A *lattice* is the set of all integer combinations of $n$ linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbb{R}^n$. These vectors are known as a *basis* of the lattice. The most basic computational problem involving lattices is the *shortest vector problem* (SVP), which asks to find a nonzero lattice vector of smallest norm, given a lattice basis as input. The inhomogeneous version of the problem is called the *closest vector problem* (CVP); here we are given an arbitrary vector in addition to the lattice basis and asked to find the lattice point closest to that vector.

Algorithms for these problems can be used to solve a wide range of problems, such as integer programming [16], factoring polynomials with rational coefficients [17], integer relation finding [15], as well as problems in communication theory (see [1,25] and references therein). They are also extremely useful in public-key cryptanalysis, notably they can break special cases of RSA and DSA (see [23] and references therein). And the growing interest in lattice-based cryptography further motivates their study.

There are two main algorithmic techniques for lattice problems. The first technique, known as *lattice reduction*, started with the celebrated LLL algorithm [17] and continued with blockwise algorithms [31,32,12] such as BKZ [32]. It works by applying successive elementary transformations to the input basis in an attempt to make its vectors shorter and more orthogonal. For usual parameters, such algorithms run in polynomial time, but the approximation factor they provide is asymptotically exponential (see [13] for experimental results). A second and more basic approach, which is the focus of our work, is the *enumeration technique*

H. Gilbert (Ed.): EUROCRYPT 2010, LNCS 6110, pp. 257–278, 2010.

which dates back to the early 1980s with work by Pohst [27], Kannan [16], and Fincke-Pohst [11], and is still actively investigated (e.g., [32,1,14,28,30,20,35]). In its simplest form, enumeration is simply an exhaustive search for the best integer combination of the basis vectors. Enumeration algorithms run in exponential time (or worse) but find the shortest vector (as opposed to a loose approximation thereof).

The two approaches are often combined. First, blockwise lattice reduction algorithms rely on a subroutine to find short vectors in a low-dimensional lattice, whose dimension is a parameter known as the "block size". This subroutine is typically implemented through enumeration. Second, the running time of enumeration algorithms crucially depends on the quality of the input basis. Therefore, enumeration algorithms are almost never applied directly to a given basis; instead, one first applies lattice reduction and then runs an enumeration algorithm on the resulting reduced basis.

An alternative algorithmic technique for solving lattice problems was suggested in 2001 by Ajtai, Kumar, and Sivakumar [4] (see also [5,26,21,29] for recent improvements). Although this technique, known as *sieving*, leads to the asymptotically fastest algorithms for solving lattice problems exactly (running in time essentially $2^{O(n)}$), it also requires an exponential amount of space, and as a result, it is so far not useful in practice. We will not discuss this technique in the remainder of the paper.

*Previous results.* As mentioned above, the focus of our work is on *enumeration algorithms* which are important not just as standalone algorithms, but also as routines in lattice reduction algorithms. The basic enumeration algorithm (as in [11,32]) essentially amounts to an exhaustive search for an integer combination of the basis vectors whose norm is small enough, say, at most some given threshold $R$. The search can be seen as a depth-first search on a tree whose leaves correspond to lattice points, and whose internal nodes correspond to partial assignments to the coefficients of the integer combination, or geometrically, to the intersection of the lattice with subspaces (see Sect. 3 for a more detailed description). We include in the tree only those nodes whose norm is at most $R$.

In an attempt to speed up the running time of this algorithm, Schnorr, Euchner, and Hörner [32,33] suggested in the 1990s a modification of the basic enumeration algorithm, called *pruned enumeration*. The rough idea is to prune subtrees of the tree in which the "probability" of finding a desired lattice point is "too small".[1] By doing so, we effectively restrict our exhaustive search to a subset of all possible solutions. The hope is that although this introduces some probability of missing the desired vector, this "probability" would be small compared to the gain in running time.

Experimentally, this led to breaking certain instances of the Chor-Rivest cryptosystem, and as a result, the pruning algorithm of Schnorr and Hörner [33] made it into the popular mathematical library NTL [34], as a subroutine of BKZ [32]. Unfortunately, to the best of our knowledge, no careful analysis of

[1] Making these notions precise requires care, as we shall see later.

pruned enumeration was ever performed. The arguments that appear in the existing literature are, as far as we can tell, mostly intuitive and do not provide a proper analysis of pruned enumeration.[2]

*Our results.* We start out by providing what we believe is the first sound analysis of pruned enumeration. Our analysis applies to a very general family of pruned enumeration algorithms, in which the pruning is determined by an arbitrary *bounding function*, which provides for each level of the tree an upper bound on the distance of nodes that should be considered. As we will see, the running time of the enumeration algorithm is determined by the volume of certain high-dimensional bodies. Our analysis is based on two heuristic assumptions, both of which we believe are very reasonable. We also provide experimental evidence to back these assumptions.

Next, we use our analysis to understand the effect of various bounding functions on the performance of pruned enumeration algorithms. For instance, the analysis can show that well-chosen bounding functions lead asymptotically to an exponential speedup of about $2^{n/4} \approx 1.189^n$ over basic enumeration, while maintaining a success probability $\geq 95\%$.

But our main contribution is the realization that further exponential speedups can be obtained by using bounding functions that *significantly reduce* the search region. With such bounding functions, the probability of finding the desired vector is actually rather low (say, 0.1%), but surprisingly, the running time of the enumeration is reduced by a much more significant factor (say, much more than 1000). A rigorous explanation of why this happens will be given in Section 5. As a result, we can repeat the pruned enumeration algorithm several times (say, 1000) until the desired vector is found, and the total running time becomes significantly smaller than what one would obtain with standard pruned enumeration. We note that we must "reshuffle" the basis vectors before each enumeration as otherwise all the enumerations would behave identically (this will be explained in more detail when we discuss our second heuristic assumption).

We call this method, which we view as our main conceptual contribution, *extreme pruning*. We note that a similar idea is used in other algorithms; for instance, this is one of the underlying ideas in Lenstra's elliptic curve factoring method. We are not aware of any other application of this idea in the context of lattice algorithms. Our analysis shows that a well-chosen extreme pruning leads asymptotically to an exponential speedup of about $(2-\varepsilon)^{n/2} \approx 1.414^n$ over basic enumeration, which is roughly the square of the previous speedup $2^{n/4}$.

*Experimental results.* In practice, our best extreme pruning is able to find the shortest vector of dense knapsack lattices of dimension 110 (resp. 100) in less than 62.12 (resp. 1.73) CPU days of sequential computation on a single 1.86-Ghz core, with a negligible amount of memory and in a trivially parallelizable manner. With plain Schnorr-Euchner enumeration [32] on a BKZ-35 reduced basis, it would have taken $1.38 \cdot 10^9$ (resp. 482000) CPU years, so the speedup is about

[2] In Appendix A, we show some flaws in the analysis of Schnorr and Hörner [33].

$8.1 \cdot 10^9$ (resp. $1.0 \cdot 10^8$). We are currently running new experiments on random lattices (as used in [13]), and we expect similar exponential speedups. To the best of our knowledge, none of these dense lattices can be handled by standard lattice reduction: they are harder than the 350-dimensional lattice (solved in [22]) from the GGH challenges.

*Open Questions.* We expect extreme pruning to improve the performance of lattice reduction algorithms [32,13], but we leave it to future work to assess its precise impact. Our focus in this paper is on high-dimensional enumeration, whereas lattice reduction algorithms typically apply enumeration on blocks whose dimension is rather small; for instance, the experiments of [22] used a block size of 60. Currently, our extreme pruning algorithm improves enumeration by randomization, but still uses negligible space; it would be interesting to see if further improvements can be made by using more space. One way to do so would be to design new algorithms for the closest vector problem with preprocessing (CVPP). Indeed, a good CVPP algorithm can help to prune the enumeration tree in the following way: one would enumerate all the nodes at some depth $k$, and use the CVPP algorithm to discard those which do not lead to any leave, without having to compute all their children. Unfortunately, we have so far been unable to obtain an improvement in practice using state-of-the-art CVPP algorithms [2]. (But see [20] for a theoretically important algorithm that combines CVPP and enumeration.)

*Roadmap.* We start in Section 2 with some background and notation on lattices, and continue with a description of the basic enumeration algorithm in Section 3. In Section 4 we describe the pruned enumeration algorithm and give our rigorous analysis. Using that analysis, we introduce and analyze the extreme pruning algorithm in Section 5. Finally, we present our experimental results in Sect. 6. Further information is given in the Appendix: App. A discusses the Schnorr-Hörner pruning [33]; and App. B describes the code used in our experiments, which includes an apparently new implementation trick that speeds up enumeration.

## 2 Preliminaries

Lattices are discrete subgroups of $\mathbb{R}^m$. Any lattice $L$ can be defined by a basis, which is a set of linearly independent vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ in $\mathbb{R}^m$ such that $L$ is equal to the set $L(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\sum_{i=1}^n x_i \mathbf{b}_i, x_i \in \mathbb{Z}\}$ of all integer linear combinations of the $\mathbf{b}_i$'s. All the bases of $L$ have the same number $n$ of elements, called the dimension of $L$, and they all have the same volume, called the volume $\mathrm{vol}(L)$ or determinant of $L$. Throughout the paper, we use row representations of matrices. The Euclidean norm of a vector $\mathbf{v} \in \mathbb{R}^m$ is denoted $\|\mathbf{v}\|$. We denote by $\mathrm{Ball}_n(R)$ the $n$-dimensional Euclidean ball of radius $R$, and by $V_n(R) = R^n \cdot \frac{\pi^{n/2}}{\Gamma(n/2+1)}$ its volume. The $n$-dimensional unit sphere is denoted by $S^{n-1}$.

*Shortest vector.* A lattice $L$ contains non-zero vectors of minimal Euclidean norm: this norm is called the *first minimum* $\lambda_1(L)$ of $L$. A vector of norm $\lambda_1(L)$ is called a *shortest vector* of $L$, and is in general unique up to the sign. Hermite's constant $\gamma_n$ is the supremum of the ratio $(\lambda_1(L)/\mathrm{vol}(L)^{1/n})^2$ over all $n$-dimensional lattices. Minkowski's theorem shows that $\sqrt{\gamma_n}$ is smaller than the diameter of the $n$-dimensional ball of volume 1: $\sqrt{\gamma_n} \leq 2 \cdot V_n(1)^{-1/n} \leq \sqrt{n}$.

*Orthogonalization.* A basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$ can be written uniquely as a product $B = \mu \cdot D \cdot Q$ where $\mu = (\mu_{i,j})$ is an $n \times n$ lower-triangular matrix with unit diagonal, $D$ an $n$-dimensional positive diagonal matrix and $Q$ an $n \times m$ matrix with orthonormal row vectors. Then $\mu D$ is a lower triangular representation of $B$ (with respect to $Q$), $B^* = DQ = (\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$ is the Gram-Schmidt orthogonalization of the basis, and $D$ is the diagonal matrix formed by the $\|\mathbf{b}_i^*\|$'s. For all $i \in \{1, \ldots, n+1\}$, we denote by $\pi_i$ the orthogonal projection on $(\mathbf{b}_1, \ldots, \mathbf{b}_{i-1})^{\perp}$. For all $i \in \{1, \ldots, n+1\}$, $\pi_i(L)$ is an $n+1-i$ dimensional lattice generated by the basis $(\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_n))$, with $\mathrm{vol}(\pi_i(L)) = \prod_{j=i}^{n} \|\mathbf{b}_j^*\|$.

*Reduced bases.* Lattice reduction algorithms aim to transform an input basis into a "high quality" basis. There are many ways to quantify the quality of bases produced by lattice reduction algorithms. One popular way, which is particularly useful for our purposes, is to consider the Gram-Schmidt norms $\|\mathbf{b}_1^*\|, \ldots, \|\mathbf{b}_n^*\|$. Intuitively speaking, a good basis is one in which this sequence never decays too fast. In practice, it turns out that the Gram-Schmidt coefficients of bases produced by the main reduction algorithms (such as LLL or BKZ) have a certain "typical shape", assuming the input basis is sufficiently random. This property was thoroughly investigated in [13,24]: accordingly, our speedup analysis assumes to simplify that $\|\mathbf{b}_i^*\|/\|\mathbf{b}_{i+1}^*\| \approx q$ where $q$ depends on the reduction algorithm.

*Gaussian Heuristic.* The Gaussian Heuristic provides an estimate on the number of lattice points inside a "nice enough" set.

**Heuristic 1.** *Given a lattice $L$ and a set $S$, the number of points in $S \cap L$ is approximately* $\mathrm{vol}(S)/\mathrm{vol}(L)$.

In some cases, this heuristic can be proved. For instance, Ajtai showed [3] that for any finite Borel set $S$ of measure $V$ which does not contain 0, the expectation of $S \cap L$ taken over a certain natural distribution on lattices $L$ of volume $D$ is $V/D$. In particular, the expectation of $\lambda_1(L)$ on random lattices of volume $D$ is the radius of the $n$-dimensional ball of volume $D$, that is $D^{1/n} \cdot V_n(1)^{-1/n}$, which is often used as a "prediction" of $\lambda_1(L)$ for a "typical" lattice. There are also counterexamples to this heuristic (see, e.g., [19] for counterexamples in $\mathbb{Z}^n$).

## 3 Enumeration

We recall Schnorr-Euchner's enumeration algorithm [32] which is the enumeration algorithm used in practice, and analyze its cost.

### 3.1 Setting

To simplify the exposition, we assume in the rest of the paper the following setting. Let $L$ be a lattice whose shortest vector $\mathbf{v}$ is unique (up to sign). Our goal is to find $\mathbf{v}$. (Our entire analysis can be extended in a straightforward manner to the closest vector problem.) We assume we are given a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of $L$ and a very good upper bound $R$ on $\lambda_1(L)$ so that finding $\pm\mathbf{v}$ amounts to finding any nonzero lattice vector $\mathbf{w} \in L$ such that $\|\mathbf{w}\| \leq R$, and therefore, we can easily check whether or not the solution is correct. In many practical situations, $\lambda_1(L)$ is known exactly: this is typically true in cryptographic situations, such as in CJLOSS lattices [7]. In the full version, we will explain how to adapt our analysis to the general case of SVP.

### 3.2 Description

To find $\pm\mathbf{v}$, enumeration goes through the *enumeration tree* formed by all vectors in the projected lattices $\pi_n(L)$, $\pi_{n-1}(L)$, $\ldots, \pi_1(L)$ of norm at most $R$. More precisely, the enumeration tree is a tree of depth $n$, and for each $k \in \{0, \ldots, n\}$, the nodes at depth $k$ are all the vectors of the rank-$k$ projected lattice $\pi_{n+1-k}(L)$ with norm at most $R$. In particular, the root of the tree is the zero vector (because $\pi_{n+1}(L) = \{0\}$), while the leaves are all the vectors of $L$ of norm $\leq R$. The parent of a node $\mathbf{u} \in \pi_{n+1-k}(L)$ at depth $k$ is by definition $\pi_{n+2-k}(\mathbf{u})$ at depth $k-1$. And we order the child nodes by increasing Euclidean norm: note that all ancestors of a given node are at most as long as the node, because they are projections of the node.

We note that the tree is symmetric, because if $\mathbf{x} \in L$ then $-\mathbf{x} \in L$. Thus, we halve the tree by restricting to "positive" nodes: we only consider nodes $\pi_{n+1-k}(\mathbf{u})$ where the last nonzero coordinate of $\mathbf{u} \in L$ with respect to $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is positive. From now on, by enumeration tree, we mean this halved enumeration tree, which has a single leaf, either $\mathbf{v}$ or $-\mathbf{v}$. The Schnorr-Euchner algorithm [32] performs a Depth First Search of the tree to find the single leaf. The more reduced the basis is, the less nodes in the tree, and the cheaper the enumeration.

Concretely, the shortest vector $\mathbf{v} \in L$ may be written as $\mathbf{v} = v_1\mathbf{b}_1 + \cdots + v_n\mathbf{b}_n$ where the $v_i$'s are unknown integers and $\mathbf{b}_i = \mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*$. Then $\mathbf{v} = \sum_{j=1}^{n} \left(v_j + \sum_{i=j+1}^{n} \mu_{i,j}v_i\right) \mathbf{b}_j^*$, which gives the norms of its projections as:

$$\|\pi_{n+1-k}(\mathbf{v})\|^2 = \sum_{j=n+1-k}^{n} \left(v_j + \sum_{i=j+1}^{n} \mu_{i,j}v_i\right)^2 \|\mathbf{b}_j^*\|^2, \ 1 \leq k \leq n \qquad (1)$$

Now, if $\mathbf{v}$ is a leaf of the tree, then the $n$ inequalities $\|\pi_{n+1-k}(\mathbf{v})\| \leq R$ together with (1) enable us to perform an exhaustive search for the coordinates $v_n, v_{n-1}, \ldots, v_1$ of $\mathbf{x}$:

$$\sum_{j=n+1-k}^{n} \left(v_j + \sum_{i=j+1}^{n} \mu_{i,j}v_i\right)^2 \|\mathbf{b}_j^*\|^2 \leq R^2, \quad 1 \leq k \leq n,$$

which can be rewritten for $1 \leq k \leq n$ as

$$\left| v_{n+1-k} + \sum_{i=n+2-k}^{n} \mu_{i,j} v_i \right| \leq \frac{\sqrt{R^2 - \sum_{j=n+2-k}^{n} \left( v_j + \sum_{i=j+1}^{n} \mu_{i,j} v_i \right)^2 \|\mathbf{b}_j^*\|^2}}{\|\mathbf{b}_{n+1-k}^*\|} \tag{2}$$

We start with $k = 1$ in (2), that is: $0 \leq v_n \leq R/\|\mathbf{b}_n^*\|$ because we restricted to "positive" nodes. This allows to perform an exhaustive search for the integer $v_n$, and we do so by increasing values of $v_n$. Now, assume that the projection $\pi_{n+2-k}(\mathbf{v})$ has been guessed for some $k$: the integers $v_{n+2-k}, \ldots, v_n$ are known. Then (2) enables to compute an interval $I_{n+1-k}$ such that $v_{n+1-k} \in I_{n+1-k}$, and therefore to perform an exhaustive search for $v_{n+1-k}$. A Depth First Search of the tree corresponds to enumerating $I_{n+1-k}$ from its middle, by increasing values of $\|\pi_{n+1-k}(\mathbf{v})\|$, namely $v_{n+1-k} = \lfloor -\sum_{i=n+2-k}^{n} \mu_{i,j} v_i \rceil, \lfloor -\sum_{i=n+2-k}^{n} \mu_{i,j} v_i \rceil \pm 1$, and so on.

### 3.3 Complexity

The running time of the enumeration algorithm is $N$ polynomial-time operations where $N$ is the total number of tree nodes. Hence, in order to analyze this running time, we need to obtain good estimates of $N$. As already suggested by Hanrot and Stehlé [14], a good estimate of $N$ can be derived from the Gaussian heuristic. More precisely, the number of nodes at level $k$ is exactly half the number of vectors of $\pi_{n+1-k}(L)$ of norm $\leq R$ (where the half comes because we halved the tree). Since $\mathrm{vol}(\pi_{n+1-k}(L)) = \prod_{i=n+1-k}^{n} \|\mathbf{b}_i^*\|$, the Gaussian heuristic predicts the number of nodes at level $k$ scanned by the Schnorr-Euchner algorithm to be close to

$$H_k = \frac{1}{2} \cdot \frac{V_k(R)}{\prod_{i=n+1-k}^{n} \|\mathbf{b}_i^*\|}. \tag{3}$$

If this holds, then $N \approx \sum_{k=1}^{n} H_k$. In Sect. 6.1, we present experiments that strongly support this heuristic estimate.

For a typical reduced basis, we have $\|\mathbf{b}_i^*\|/\|\mathbf{b}_{i+1}^*\| \approx q$ where $q$ depends on the reduction algorithm (see [13]). The bound $R = \sqrt{\gamma_n}\mathrm{vol}(L)^{1/n}$ is optimal in the worst case. Since $\sqrt{\gamma_n} = \Theta(\sqrt{n})$, an elementary computation shows that (3) becomes:

$$H_k \approx \frac{\|\mathbf{b}_1\|^{n-k} 2^{O(n)}}{q^{(n-1-k)(n-k)/2}\mathrm{vol}(L)^{(n-k)/n}} = \frac{q^{(n-k)(n-1)/2} 2^{O(n)}}{q^{(n-1-k)(n-k)/2}} = q^{(n-k)k/2} 2^{O(n)},$$

where the right-hand term is always less than $q^{n^2/8} 2^{O(n)}$ because $(n-k)(k/2)$ is maximized for $k = n/2$. Hence:

$$H_k \lessapprox q^{n^2/8} 2^{O(n)}.$$

Thus, $\max_k H_k$ is super-exponential in $n$ and is reached for $k \approx n/2$, which is consistent with experiments (see Fig. 1 of Sect. 5.2). For small values of $n$, the term $2^{O(n)}$ is not negligible, and may shift a bit the maximum index $k \approx n/2$.

We note that if we make the (reasonable) assumption that the location of the leaf is uniform, the number of nodes scanned by the enumeration algorithm will only be $N/2$ in expectation, and not $N$. For simplicity, we ignore this factor 2 in the sequel.

Finally, we mention that *rigorous* bounds on $N$ exist. For instance, if the basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is LLL-reduced, and $R = \|\mathbf{b}_1\|$, then it is well-known that $N$ is at most $2^{O(n^2)}$. Also, Hanrot and Stehlé [14] showed that if the basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is so-called quasi-HKZ-reduced, and $R = \|\mathbf{b}_1\|$, then $N \leq n^{n/(2e)+o(n)}$. See [14] for details.

## 4 Pruned Enumeration

Since enumeration is expensive, it is tempting not to enumerate all the tree nodes, by discarding certain branches. The idea of pruned enumeration goes back to Schnorr and Euchner [32], and was further studied by Schnorr and Hörner [33]. For instance, one might intuitively hope that $\|\pi_{n/2}(\mathbf{v})\|^2 \lessapprox \|\mathbf{v}\|^2/2$, which is more restrictive than the inequality $\|\pi_{n/2}(\mathbf{v})\|^2 \leq \|\mathbf{v}\|^2$ used by enumeration. Formally, pruning replaces each of the $n$ inequalities $\|\pi_{n+1-k}(\mathbf{v})\| \leq R$ by $\|\pi_{n+1-k}(\mathbf{v})\| \leq R_k$ where $R_1 \leq \cdots \leq R_n = R$ are $n$ real numbers defined by the pruning strategy. This means that one replaces $R$ by $R_k$ in each of the $n$ inequalities (2).

At the end of their paper [32] , Schnorr and Euchner briefly proposed $R_k = R\min(1, \sqrt{(1.05)k/n})$, but did not provide any analysis, only limited experiments. Schnorr and Hörner [33] later proposed another choice of $R_k$'s, which we discuss in App. A: we show that this pruning has flaws, and that the analysis of [33] is mostly incorrect; in particular, if the heuristic analysis of [33] was correct, it would imply polynomial-time algorithms for the shortest vector problem, while the problem is NP-hard under randomized reductions.

We now provide what we believe is the first rigorous analysis of pruned enumeration: The next two subsections deal with the running time and the success probability. In principle, this analysis can be used to optimize the choice of the bounding function in the pruning algorithm. However, we did not attempt to do that since this is not the main focus of our paper. Instead, our analysis will be used in the next section to show how exponential speedups (both in theory and in practice) can be achieved through the use of extreme pruning.

### 4.1 Running Time Analysis

The running time of the pruned enumeration algorithm is given by

$$T_{\text{node}} \cdot N$$

where $T_{\text{node}}$ is the average amount of time spent processing one node in the enumeration tree, and $N$ is the number of nodes in the pruned tree. In order to estimate $N$, we estimate the number of nodes at each level of the search tree

using the Gaussian heuristic (Heuristic 1). As we shall see later, our estimates agree very nicely with the experiments, giving some further justification to the use of the Gaussian heuristic.

Our estimate is very similar to the one in Equation 3, except that instead of using balls of radius $\sqrt{R}$, we use *cylinder-intersections* of radii $(R_1, \ldots, R_k)$ for $1 \leq k \leq n$. More specifically, define the ($k$-dimensional) cylinder-intersection of radii $R_1 \leq \cdots \leq R_k$ as the set

$$C_{R_1,\ldots,R_k} = \left\{ (x_1, \ldots, x_k) \in \mathbb{R}^k,\ \forall j \leq k,\ \sum_{l=1}^{j} x_l^2 \leq R_j^2 \right\}.$$

Notice that the set of vertices in level $k$ of the pruned tree correspond exactly to the points of the projected lattice $\pi_{n+1-k}(L)$ that are inside $C_{R_1,\ldots,R_k}$. Therefore, using the Gaussian heuristic, we can estimate the number of nodes in the enumeration tree using

$$N = N_{R_1,\ldots,R_n}(\|\mathbf{b}_1^*\|, \ldots, \|\mathbf{b}_n^*\|) = \frac{1}{2} \sum_{k=1}^{n} \frac{V_{R_1,\ldots,R_k}}{\prod_{i=n+1-k}^{n} \|\mathbf{b}_i^*\|} \tag{4}$$

where $V_{R_1,\ldots,R_k}$ denotes the volume of $C_{R_1,\ldots,R_k}$, and the factor half is as a result of the symmetry in the SVP problem.

There are several ways to compute or approximate the volume of cylinder intersections $V_{R_1,\ldots,R_k}$. The simplest and most naïve method, which is the one we used at first in our numerical optimizations, is based on a Monte Carlo method. Namely, by observing that the cylinder intersection $C_{R_1,\ldots,R_k}$ is contained in a ball of radius $R_k$, we can write

$$V_{R_1,\ldots,R_k} = V_k(R_k) \cdot \Pr_{\mathbf{u} \sim \mathrm{Ball}_k} \left( \forall j \in [1,k],\ \sum_{i=1}^{j} u_i^2 \leq \frac{R_j^2}{R_k^2} \right). \tag{5}$$

The number of samples required to estimate the above probability by Monte Carlo sampling is proportional to its inverse. One can speed things up significantly by replacing the ball with a smaller containing body (such as another cylinder intersection) whose volume is known and from which we can sample uniformly.

For certain interesting choices of radii $(R_1, \ldots, R_k)$, rigorous estimates can be obtained, as we shall see in Section 5. Moreover, one particular case in which we can compute the volume *exactly* is when $R_1 = R_2$, $R_3 = R_4$, etc. and $k$ is even. This is because the distribution of the vector $(u_1^2 + u_2^2, u_3^2 + u_4^2, \ldots, u_{k-1}^2 + u_k^2)$ when $\mathbf{u}$ is chosen from $\mathrm{Ball}_k$ is given by a Dirichlet distribution with parameters $(1, \ldots, 1)$ ($k/2 + 1$ ones), which is simply a uniform distribution over the set of all vectors whose coordinates are non-negative and sum to at most 1 (see Page 593 of [9]). This leads to an easy way to compute the probability in Eq. 5 exactly (as it amounts to computing the volume of a certain polytope). This calculation can also be combined with the Monte Carlo simulation above, leading to much faster running times.

Finally, let us also mention that there is a large body of work showing *provably polynomial time* algorithms that provide a good approximation to the volume of *any* convex body (see, e.g., [10,18]). However, in practice these algorithms are rather slow and are therefore probably not too useful for our purposes.

### 4.2 Success Probability Analysis

We let $p_{\text{succ}}$ denote the "probability" that the target vector is still in the tree after the pruning. Prior to our work, the implicit assumption was that one should choose a bounding function so as to minimize the running time while keeping $p_{\text{succ}}$ reasonably high, say 95%. As we shall see in the next section, surprising speedups can be obtained through *extreme pruning*, i.e., when $p_{\text{succ}}$ is very small.

Before proceeding with the analysis, we must explain what we mean by "probability", since the pruning algorithm is entirely deterministic. (We note that in previous work this was often glossed over; see App. A). In order to meaningfully talk about the success probability $p_{\text{succ}}$, we must assume some kind of distribution on the inputs (since the pruning algorithm is entirely deterministic). For that purpose, we make the following heuristic assumption on the input basis:

**Heuristic 2.** *The distribution of the coordinates of the target vector* $\mathbf{v}$*, when written in the normalized Gram-Schmidt basis* $(\mathbf{b}_1^*/\|\mathbf{b}_1^*\|, \ldots, \mathbf{b}_n^*/\|\mathbf{b}_n^*\|)$ *of the input basis, look like those of a uniformly distributed vector of norm* $\|\mathbf{v}\|$*.*

We use here the imprecise term 'looks like' on purpose. It should be interpreted simply as saying that the estimate on $p_{\text{succ}}$ obtained by performing the analysis under the above assumption on the coordinates of the shortest vector corresponds to what one observes in practice on any reasonable distribution of inputs (see Sect. 6 for experiments). We note that Heuristic 2 would follow from a (stronger) natural heuristic on reduced bases:

**Heuristic 3.** *The distribution of the normalized Gram-Schmidt orthogonalization* $(\mathbf{b}_1^*/\|\mathbf{b}_1^*\|, \ldots, \mathbf{b}_n^*/\|\mathbf{b}_n^*\|)$ *of a random reduced basis* $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ *looks like that of a uniformly distributed orthogonal matrix.*

Here, we assume that the reduction is not too strong, that is, the reduced basis is made of vectors that are significantly longer than the shortest vector of the lattice. In typical randomly constructed lattices, the number of such vectors is exponential,[3] and hence, we may hope that the reduced basis is not oriented in any particular direction.

We now estimate the success probability $p_{\text{succ}}$. Let $\mathbf{v}$ be the target vector, and let $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ be its coordinates in the orthonormal basis $(\mathbf{b}_n^*/\|\mathbf{b}_n^*\|, \ldots, \mathbf{b}_1^*/\|\mathbf{b}_1^*\|)$ (notice that the coordinates are reversed, with $x_1$ corresponding to $\mathbf{b}_n$ etc.). By definition, $\mathbf{v}$ belongs to the pruned tree if and only

[3] Moreover, for any $c \geq 1$ and any $n$-dimensional lattice $L$, the number of lattice points of norm at most $2cV_n(1)^{-1/n}\text{vol}(L)^{1/n}$ is at least $\lceil c^n \rceil$. But this bound only applies for radii above Minkowski's upper bound, which is twice the Gaussian heuristic.

if for all $k = 1, \ldots, n$, $\sum_{j=1}^{k} x_j^2 \leq R_k^2$. By Heuristic 2, $\mathbf{x}$ is distributed like a uniform vector subject to the constraints that $\|\mathbf{x}\| = \|\mathbf{v}\|$. Hence, we can estimate $p_{\text{succ}}$ as

$$p_{\text{succ}} = p_{\text{succ}}(R_1, \ldots, R_n) = \Pr_{u \sim S^{n-1}\|\mathbf{v}\|/R} \left( \forall j \in [1, n], \sum_{l=1}^{j} u_l^2 \leq \frac{R_j^2}{R_n^2} \right) \quad (6)$$

where $S^{n-1}$ denotes the unit sphere in $n$ dimensions. As before, one can estimate this probability through Monte Carlo simulation, or compute it exactly in certain cases (e.g., using the fact that if $u$ is chosen from $S^{n-1}$ for even $n$, then $(u_1^2 + u_2^2, \ldots, u_{n-3}^2 + u_{n-2}^2)$ is distributed uniformly over all vectors whose coordinates are non-negative and sum to at most 1).

## 5 Extreme Pruning

In this section we present our main contribution, the extreme pruning algorithm, whose main idea is to apply pruning using bounding functions whose $p_{\text{succ}}$ is very small. Our algorithm takes as input a lattice basis, as well as $n$ real numbers $R_1^2 \leq \cdots \leq R_n^2 = R^2$ (the bounding function) where $R_k$ corresponds to the pruning in depth $k$. The goal of the algorithm is to find a vector of length at most $R$, and is described in Algorithm 1.

**Algorithm 1.** Extreme Pruning

Repeat until a vector of length at most $R$ is found:

1. Randomize the input basis, and apply basis reduction to it.
2. Run the enumeration on the tree pruned with radii $R_1, \ldots, R_n$, as explained in Sect. 4.

We are being deliberately imprecise in Step 1 of the algorithm. First, we do not know what the best method of randomization is, and it is quite likely that this does not matter much. In our experiments, we simply multiplied the input basis by some small unimodular matrix chosen at random, but one can also use other methods. Second, as we shall see in the analysis below, the choice of basis reduction has a great effect on the overall running time, and has to be set properly. The choice of basis reduction algorithm, as well as of the bounds $R_1, \ldots, R_n$ will be the topic of Sections 5.2 and 5.3. But first we analyze the expected running time of the algorithm.

### 5.1 Running Time Analysis

We now analyze the expected running time of the extreme pruning algorithm based on the analysis in Section 4.

First, we estimate the probability of success in each iteration of the algorithm by $p_{\mathrm{succ}}(R_1, \ldots, R_n)$, as in Eq. 6, which is based on Heuristic 2. Here, we explicitly perform a randomization before reduction, and we stress that Heuristic 2 produces estimates on $p_{\mathrm{succ}}$ that agree very nicely with our experiments (see Sect. 6 for more details).

Next, we estimate the running time of each iteration of the algorithm. Let us denote the (average) running time of Step 1 by $T_{\mathrm{reduc}}$. Once a choice of randomization and reduction algorithm is fixed, this time can be easily estimated experimentally. The running time of Step 2 can be estimated by $N_{R_1,\ldots,R_n}(\|\mathbf{b}_1^*\|, \ldots, \|\mathbf{b}_n^*\|)$, as in Eq. 4. Notice that this running time depends on the Gram-Schmidt coefficients of the basis produced in Step 1, and might vary from one iteration to another. In order to simplify the analysis, we assume that these Gram-Schmidt coefficients are the same throughout all iterations, and denote them by $\bar{\mathbf{b}}_1^*, \ldots, \bar{\mathbf{b}}_n^*$. This is partly justified by the observation that bases produced by known reduction algorithms have a clear shape that depends only on the reduction algorithm and not so much on the input basis. Alternatively, it should be straightforward to refine our analysis so that it takes into account the *distribution* of Gram-Schmidt coefficients produced in Step 1, as opposed just to their average. Yet another possibility is to modify the algorithm so that 'bad' bases (i.e., those that differ significantly from the average behavior) are discarded.

To summarize, we can estimate the expected time required to find the desired vector by

$$T_{\mathrm{extreme}}(R_1, \ldots, R_n, \bar{\mathbf{b}}_1^*, \ldots, \bar{\mathbf{b}}_n^*) := \frac{T_{\mathrm{reduc}} + T_{\mathrm{node}} \cdot N_{R_1,\ldots,R_n}(\bar{\mathbf{b}}_1^*, \ldots, \bar{\mathbf{b}}_n^*)}{p_{\mathrm{succ}}(R_1, \ldots, R_n)}. \quad (7)$$

### 5.2 Choosing Parameters for the Experiments

In order to optimize the running time of the extreme pruning algorithm, we need to choose the basis reduction algorithm used in Step 1, as well as the bounding parameters $R_1 \leq \cdots \leq R_n$ used in Step 2. These choices are crucial in determining the running time of the algorithm.

Since finding the exact optimum of the expression in Eq. (7) seems difficult, we decided to try numerical optimization. We wrote a program that starts from the linear bounding function $R_k^2 = (k/n) \cdot R^2$ and successively tries to apply small random modifications to it. After each such modification it checks if the expression in Eq. (7) decreased or not, with an exact computation based on the Dirichlet distribution (as described in Sect. 4.1). If it did, the modification is accepted; otherwise it is rejected.

This yielded bounding functions whose predicted running time is only 62.1 CPU days (including reduction time) in dimension 110 for hard knapsack lattices, which is significantly better than the linear bounding function (see Figure 1).

### 5.3 Asymptotic Analysis

In this section, we provide a rigorous justification to the effectiveness of extreme pruning. We do this through an analysis of three bounding functions, whose

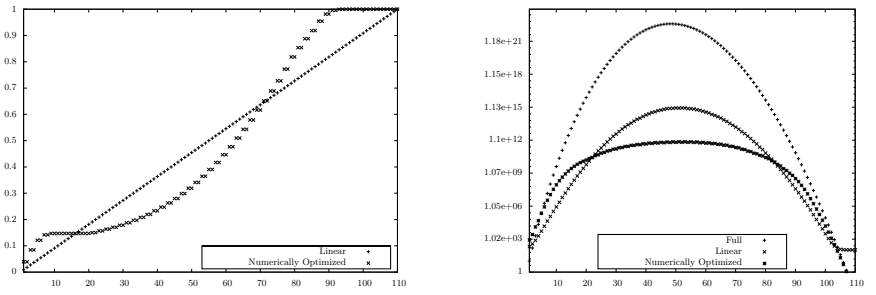


**Fig. 1.** On the left are the linear function and our best bounding function found by numerical optimization. On the right we compare the estimated expected number of nodes (with respect to the depth) visited in a run of full enumeration, extreme pruning with the linear function, and extreme pruning using our best bounding function. Note that basis reduction times are ignored here.

asymptotic behavior we can analyze (under our reasonable heuristic assumptions). The first is the linear bounding function. The speedup it offers over full enumeration is provably exponential, but it is not significantly better than what we can achieve using non-extreme pruning. This is perhaps not surprising since its success probability $p_{\text{succ}}$ is $1/n$, which is relatively high (so this bounding function is not too 'extreme'). Our second example is a step bounding function. We show that this function obtains an exponential improvement over our best non-extreme bounding function in terms of the number of nodes scanned in the middle level of the tree. This function nicely highlights the reason extreme pruning is superior to non-extreme pruning. Our third bounding function is a piecewise linear bounding function. Its analysis combines the previous two analyses and leads to the best speedups we can rigorously demonstrate.

Throughout this section, we make the simplifying assumption that $\|\mathbf{v}\| = R$ (which, as discussed above, is the case in many scenarios, and is also the worst case for pruning algorithms). We also ignore the reduction time, which in practice might make the claimed speedups a bit smaller (but does not affect the asymptotics). Finally, instead of giving an absolute bound on the number of nodes in (each level of) the enumeration tree, we compare this number to that in the full enumeration tree. This allows us to focus on analyzing the volume of cylinder intersections, and ignore the properties of the given basis, which we leave for future work.

*Linear pruning.* We define the linear bounding function as $R_k^2 = (k/n) \cdot R^2$, for $k = 1, \ldots, n$. The motivation for this setting comes from the fact that if $\mathbf{v}$ is a uniformly random vector of length $R$ (as we assume in our heuristic), then the expectation of the squared norm of its projection on the first $k$ coordinates is exactly $(k/n) \cdot R^2$. Although this bounding function leads to running times that are far from optimal, it is interesting as it can be analyzed rigorously and shown to provide an exponential speedup over full enumeration. In the full version,

we prove the following claim, which states that the success probability of linear pruning is exactly $1/n$:

*Claim.* Let $u$ be a vector uniformly distributed in the unit sphere $S^{n-1}$. Then,

$$\Pr_u\left(\forall j \in \{1,\ldots,n\},\ \sum_{i=1}^{j} u_i^2 \leq \frac{j}{n}\right) = \frac{1}{n}.$$

This also shows that linear pruning keeps an exponentially small proportion (between $1/k\,(k/n)^{k/2}$ and $(k/n)^{k/2}$ at each depth $k$) of nodes:

**Corollary 1.** *For any integer $n \geq 1$,*

$$V_n(1)/n \leq \mathrm{vol}\left\{u \in \mathrm{Ball}_n(1) : \forall j \in \{1,\ldots,n\},\ \sum_{i=1}^{j} u_i^2 \leq \frac{j}{n}\right\} \leq V_n(1). \qquad (8)$$

Hence, compared to full enumeration, linear pruning reduces the number of nodes at depth $k$ by the multiplicative factor $(n/k)^{k/2}$, up to some polynomial factor. As we saw in Section 3, for typical reduced bases, most of the nodes in the enumeration tree are concentrated around $n/2$, in which case the speedup is approximately $2^{n/4} \approx 1.189^n$.

*Step bounding function.* We now present an example in which extreme pruning is significantly superior to non-extreme pruning, at least in the asymptotic sense.

Consider the step bounding function given by $R_k^2 = \alpha R^2$ for $1 \leq k \leq n/2$, and $R_k = R$ otherwise, where $\alpha > 0$ is a constant to be determined later. (We fix the location of the step to $n/2$ for simplicity; the analysis can be easily extended to any step function.) With this bounding function, the number of nodes in the middle level of the pruned enumeration tree compared to that in the middle level of the full enumeration tree is smaller by a factor of $\alpha^{\frac{n}{4}}$. We omit the analysis for other levels of the tree because our next bounding function will have a much better performance in this respect, and its analysis is not more complicated.

We now compute the success probability $p_{\mathrm{succ}}$. Eq. 6 tells us that $p_{\mathrm{succ}}$ is equal to the probability that for a random point $\mathbf{u} = (u_1,\ldots,u_n)$ on the sphere $S^{n-1}$, we have $\sum_{i=1}^{n/2} u_i^2 \leq \alpha$. It follows from classical concentration inequalities (see, e.g., Lemma 2.2 in [8]) that this probability is $1 - 2^{-\Omega(n)}$ if $\alpha > 1/2$, and $2^{-\Omega(n)}$ if $\alpha < 1/2$. Hence, for $\alpha > 1/2$, we are in the non-extreme regime, and by choosing $\alpha$ close to $1/2$ we obtain that the number of nodes in the middle level of the pruned enumeration tree is smaller by a factor of about $2^{n/4} \approx 1.189^n$ than that in the full enumeration tree. Since most of the nodes are located at depths around $n/2$, this is a good approximation of the speedup offered by non-extreme pruning using a step function.

Let us now consider what happens when we take $\alpha < 1/2$ and move into the extreme pruning regime. By definition, $\sum_{i=1}^{n/2} u_i^2$ is distributed according to the beta distribution $\mathrm{Beta}(\frac{n}{4}, \frac{n}{4})$. Therefore, we can express $p_{\mathrm{succ}}$ using the regularized incomplete beta function as:

$$p_{\mathrm{succ}} = I_\alpha(\frac{n}{4}, \frac{n}{4}) \geq \frac{(\frac{n}{2}-1)!}{(\frac{n}{4})!(\frac{n}{4}-1)!}\,\alpha^{\frac{n}{4}}(1-\alpha)^{\frac{n}{4}} - 1 = \Omega\left(\tfrac{1}{\sqrt{n}}(4\alpha(1-\alpha))^{\frac{n}{4}}\right)$$

where the inequality follows from integration by parts of the incomplete beta function. Hence, the total number of middle level nodes that will be scanned in the extreme pruning algorithm (which can be seen as an approximation of the overall running time) is smaller than that in full enumeration by a factor of

$$\Omega\left(\tfrac{1}{\sqrt{n}}(4(1-\alpha))^{\frac{n}{4}}\right).$$

This expression is maximized for very small $\alpha > 0$, in which case the speedup is asymptotically roughly $4^{\frac{n}{4}} \approx 1.414^n$, which is greatly superior to the speedup of $1.189^n$ obtained with non-extreme pruning.

As this example nicely demonstrates, the advantage of extreme pruning comes from the fact that for small $\alpha > 0$, although the success probability is exponentially small, the volume of the search region decreases by a stronger exponential factor.

*Piecewise linear bounding function.* Consider the bounding function defined as $R_k^2 = (2k/n)\alpha \cdot R^2$ for $k = 1, \ldots, n/2$, and $R_k^2 = (2\alpha - 1 + 2k(1-\alpha)/n) \cdot R^2$ otherwise, where we assume that $0 < \alpha < 1/4$ is a constant. In the full version, using similar arguments than for linear and step pruning, we show that: $p_{\text{succ}} \geq \Omega\left(n^{-5/2}(4\alpha(1-\alpha))^{\frac{n}{4}}\right)$. Ignoring polynomial factors, for sufficiently large $n$, the total number of level $k$ nodes that will be scanned in the extreme pruning algorithm is shown to be smaller than that in full enumeration by a factor of

$$\Omega\Big(2^{\frac{n}{2}-\frac{k}{2}}\,\alpha^{\frac{n}{4}-\frac{k}{2}}\,(1-\alpha)^{\frac{n}{4}}\left(\frac{n}{k}\right)^{\frac{k}{2}}\Big)$$

for $k \leq n/2$ and by a factor of

$$\Omega\Big(2^{\frac{n}{2}-\frac{k}{2}}\,(1-\alpha)^{\frac{n}{2}-\frac{k}{2}}\left(\frac{n}{k}\right)^{\frac{k}{2}}\Big).$$

for $k > n/2$. We see that for levels around $n/2$ (say, $0.49n \leq k \leq 0.51n$), in order to maximize the expressions above, one should choose a small $\alpha$, in which case the speedup is roughly of $2^{\frac{n}{2}} \approx 1.414^n$. For other values of $k$, the number of nodes may actually increase, but typically these levels contain a small fraction of the nodes, and the global asymptotical speedup is not affected.

## 6 Experiments

*The setup.* All our experiments are run on 64-bit Xeon processors with frequency 1.86 GHz, and compiled with g++ version 4.2.4 x86_64 (options -O9 -ffast-math -funroll-loops -ftree-vectorize). Running times are provided for a single core.

*Implementation.* For lattice reduction, we used fplll [6]'s implementation of LLL, and NTL [34]'s implementation of BKZ [32]. We implemented our own enumeration algorithms (see App. B) in basic C++, using double and long arithmetic; we plan to release the source codes. The input basis and its Gram-Schmidt orthogonalization were pre-computed with NTL [34] in RR precision, and then rounded

to double precision before entering the enumeration procedure. While floating-point arithmetic is known to cause stability problems during LLL reduction, we did not experience such problems during enumeration, even up to dimension 110; we note that a rigorous analysis of enumeration with floating-point arithmetic has recently been done in [28].

*Lattices.* It is important to test algorithms on lattices that do not have a special structure that can be exploited by standard reduction algorithms. On the other hand, we need to be able to decide if the algorithm was successful, therefore $\lambda_1(L)$ must be known. For concreteness, we performed all our experiments on hard knapsack lattices, namely the so-called CJLOSS lattices [7] of density 0.94 where the knapsack solution was further chosen with exactly as many 0s as 1s. For these lattices, we can easily check whether the vector found is the shortest vector because it corresponds to knapsack solutions. It can also be checked experimentally that they are quite dense, in the sense that their first minimum $\lambda_1(L)$ is close to the Gaussian heuristic, which can be seen as an indication that they are hard instances of SVP (see [13]). Moreover, these lattices, we believe, serve as a good representative of hard lattices that typically occur in practice. In particular, we believe that the results reported here are not limited to CJLOSS lattices, but in fact represent a general phenomenon. We are currently running new experiments on random lattices (as used in [13]), for which we have a tight estimate on the first minimum $\lambda_1(L)$, but do not know shortest vectors in advance.

*Rate of enumeration.* The amount of time an enumeration algorithm spends per node in the tree might depend slightly on the depth of that node in the tree. Luckily, this dependence is typically not too strong, and more importantly, most nodes are concentrated around the same depth of the tree (see, e.g., Fig. 1). In all our experiments in dimensions 100–110, the running time of the enumeration algorithm was directly proportional to the number of nodes in the tree, and the rate of enumeration was very close to $0.94 \cdot 10^7$ nodes per second, or equivalently $0.3 \cdot 10^{15}$ nodes per core-year. This is faster than fplll's [6] implementation (which is itself an improvement over NTL [34]) by about 40%. This is due to a code optimization described in App. B.

**Table 1.** Pruning vs. full enumeration

| dim | 90 | 100 | 110 | 120 |
|---|---|---|---|---|
| Full enumeration | $1.2 \cdot 10^{17}$ | $1.6 \cdot 10^{20}$ | $4.0 \cdot 10^{23}$ | $1.9 \cdot 10^{27}$ |
| Schnorr-Hörner | $2.3 \cdot 10^{12}$ | $7.4 \cdot 10^{14}$ | $4.7 \cdot 10^{17}$ | $3.8 \cdot 10^{20}$ |
| Linear pruning | $1.1 \cdot 10^{11}$ | $2.6 \cdot 10^{13}$ | $1.0 \cdot 10^{16}$ | $8.3 \cdot 10^{18}$ |
| Extreme pruning | n/a | $7.7 \cdot 10^{11}$ | $2.5 \cdot 10^{13}$ | n/a |

*Estimated running times.* Table 1 compares the expected total number of nodes to be scanned under four different bounding functions for CJLOSS lattices in dimensions 90–120; this number of nodes is obtained by multiplying the expected number of nodes in the pruned tree (as estimated by the Gaussian heuristic) by $1/p_{\text{succ}}$. Recall that $0.3 \cdot 10^{15}$ nodes represent one year of sequential computation. The first row corresponds to full enumeration under BKZ-35 reduced bases. The second row corresponds to Schnorr-Hörner's pruning under BKZ-35 reduced bases with an optimal choice of parameter $p$ so that the success probability is still greater than 90% (this corresponds to $p = 48, 57, 67, 77$ for dimensions $90, 100, 110$, and 120 respectively). The third row corresponds to extreme pruning using the linear bounding function under BKZ-35 reduced bases. And the last row corresponds to extreme pruning using our best numerically optimized bounding function (which we computed only for dimensions 100 and 110) and the optimum BKZ-30 (resp. BKZ-32) in dim 100 (resp. 110).

Here we are only considering the number of nodes scanned, and ignoring the basis reduction time. Except our numerically optimized bounding function, it is negligible. For our best bounding function it adds about 50% to the total running time. Another caveat is that the number of nodes in full enumeration (as well as in Schnorr-Hörner pruning and linear pruning) decreases as we increase the block size of the reduction algorithm beyond 35. However, this does not decrease the overall running time by much since the running time of reduction algorithms depends exponentially on the block size.

*Actual running time.* The actual running times match the predictions well. In practice, extreme pruning is able to find the shortest vector of 0.94-density CJLOSS lattices of dimension 110 in less than 63 CPU days (including reduction time) of sequential computation on a single core, with a negligible amount of memory and in an easily parallelizable manner. With plain Schnorr-Euchner enumeration [32] on a BKZ-35 reduced basis, it would have taken $1.38 \cdot 10^9$ (resp. 482000) CPU years, so the speedup is about $8.1 \cdot 10^9$ (resp. $1.0 \cdot 10^8$).

### 6.1 Verifying the Heuristics

In this section we report on some experiments meant to verify our heuristic assumptions.

*The accuracy of the Gaussian heuristic.* Although the Gaussian heuristic was already suggested as a useful heuristic for analyzing the running time of enumeration algorithms (see [14]), we are not aware of any published experimental verification of the heuristic. We therefore ran a significant number of experiments comparing the actual number of nodes in the enumeration tree to the prediction given by the Gaussian heuristic. We tried both CJLOSS lattices and random lattices with several different reduction algorithms and with either full enumeration or pruned enumeration. In all cases, the estimates given by the Gaussian heuristic were very precise, and typically matched the exact count of nodes to within an error of at most 5%.

*Randomness of reduced bases.* The success of our experiments gives some evidence to the validity of Heuristics 2 and 3. In the full version, we report on additional experiments whose goal is to validate these heuristics directly. We note that the heuristics cannot apply to very strong reduction notions, where the first basis vector is with very high probability the shortest lattice vector: in such cases, there is no need for enumeration since the shortest vector is already provided to us. But it seems to apply to weaker yet still strong reduction notions, such as BKZ-30 in dimension 100.

**Acknowledgements.** Part of this work was supported by the Commission of the European Communities through the ICT program under contract ICT-2007-216676 ECRYPT II, by the Binational Science Foundation, by the Israel Science Foundation, by the European Commission under the Integrated Project QAP funded by the IST directorate as Contract Number 015848, by the Wolfson Family Charitable Trust, and by a European Research Council (ERC) Starting Grant.

## References

1. Agrell, E., Eriksson, T., Vardy, A., Zeger, K.: Closest point search in lattices. IEEE Trans. on Info. Theory 48(8), 2201–2214 (2002)
2. Aharonov, D., Regev, O.: Lattice problems in NP intersect coNP. Journal of the ACM 52(5), 749–765 (2005); Preliminary version in FOCS 2004
3. Ajtai, M.: Generating random lattices according to the invariant distribution (Draft) (March 2006)
4. Ajtai, M., Kumar, R., Sivakumar, D.: A sieve algorithm for the shortest lattice vector problem. In: Proc. 33rd ACM Symp. on Theory of Computing (STOC), pp. 601–610 (2001)
5. Ajtai, M., Kumar, R., Sivakumar, D.: Sampling short lattice vectors and the closest lattice vector problem. In: Proc. of 17th IEEE Annual Conference on Computational Complexity (CCC), pp. 53–57 (2002)
6. Cadé, D., Pujol, X., Stehlé, D.: FPLLL library, version 3.0 (September 2008), `http://perso.ens-lyon.fr/damien.stehle`
7. Coster, M., Joux, A., LaMacchina, B., Odlyzko, A., Schnorr, C., Stern, J.: An improved low-density subset sum algorithm. In: Computational Complexity (1992)
8. Dasgupta, S., Gupta, A.: An elementary proof of the Johnson-Lindenstrauss lemma. Technical report, ICSI, Berkeley, TR-99-006 (1999)
9. Devroye, L.: Non-uniform random variate generation (1986), `http://cg.scs.carleton.ca/~luc/rnbookindex.html`
10. Dyer, M., Frieze, A., Kannan, R.: A random polynomial-time algorithm for approximating the volume of convex bodies. J. ACM 38(1), 1–17 (1991)
11. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Mathematics of Computation 44(170), 463–471 (1985)
12. Gama, N., Nguyen, P.Q.: Finding short lattice vectors within Mordell's inequality. In: Proc. 40th ACM Symp. on Theory of Computing, STOC (2008)
13. Gama, N., Nguyen, P.Q.: Predicting Lattice Reduction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 31–51. Springer, Heidelberg (2008)

14. Hanrot, G., Stehlé, D.: Improved analysis of Kannan's shortest lattice vector algorithm (extended abstract). In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 170–186. Springer, Heidelberg (2007)
15. Håstad, J., Just, B., Lagarias, J.C., Schnorr, C.-P.: Polynomial time algorithms for finding integer relations among real numbers. SIAM J. Comput. 18(5) (1989)
16. Kannan, R.: Improved algorithms for integer programming and related lattice problems. In: Proc. 15th ACM Symp. on Theory of Computing, STOC (1983)
17. Lenstra, A.K., Lenstra Jr., H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische Ann. 261, 513–534 (1982)
18. Lovász, L., Vempala, S.: Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. J. Comput. Syst. Sci. 72(2), 392–417 (2006)
19. Mazo, J.E., Odlyzko, A.M.: Lattice points in high dimensional spheres. Monatsheft Mathematik 17, 47–61 (1990)
20. Micciancio, D., Voulgaris, P.: A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In: Proc. 42nd ACM Symp. on Theory of Computing, STOC (2010)
21. Micciancio, D., Voulgaris, P.: Faster exponential time algorithms for the shortest vector problem. In: Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 1468–1480 (2010)
22. Nguyen, P.Q.: Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto 1997. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 288–304. Springer, Heidelberg (1999)
23. Nguyen, P.Q.: Public-key cryptanalysis. In: Luengo, I. (ed.) Recent Trends in Cryptography. Contemporary Mathematics, vol. 477. AMS–RSME (2009)
24. Nguyen, P.Q., Stehlé, D.: LLL on the average. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 238–256. Springer, Heidelberg (2006)
25. Nguyen, P.Q., Vallée, B. (eds.): The LLL Algorithm: Survey and Applications. Information Security and Cryptography. Springer, Heidelberg (2009)
26. Nguyen, P.Q., Vidick, T.: Sieve algorithms for the shortest vector problem are practical. J. of Mathematical Cryptology 2(2), 181–207 (2008)
27. Pohst, M.: On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. SIGSAM Bull. 15(1), 37–44 (1981)
28. Pujol, X., Stehlé, D.: Rigorous and efficient short lattice vectors enumeration. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 390–405. Springer, Heidelberg (2008)
29. Pujol, X., Stehlé, D.: Solving the shortest lattice vector problem in time $2^{2.465n}$, IACR eprint report number 2009/605 (2009)
30. Schnorr, C.-P.: Average time fast SVP and CVP algorithms for low density lattices. TR Goethe Universität Frankfurt (January 4, 2010)
31. Schnorr, C.-P.: A hierarchy of polynomial lattice basis reduction algorithms. Theoretical Computer Science 53(2-3), 201–224 (1987)
32. Schnorr, C.-P., Euchner, M.: Lattice basis reduction: improved practical algorithms and solving subset sum problems. Math. Programming 66, 181–199 (1994)
33. Schnorr, C.-P., Hörner, H.H.: Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 1–12. Springer, Heidelberg (1995)
34. Shoup, V.: Number Theory C++ Library (NTL) version 5.4.1, `http://www.shoup.net/ntl/`
35. Stehlé, D., Watkins, M.: On the extremality of an 80-dimensional lattice (2010) (manuscript)

# A Schnorr-Hörner Pruning

Here we revisit the pruning strategy described by Schnorr and Hörner in [33], and analyze it in our framework. Their pruning strategy is implemented in NTL [34] as a subroutine to BKZ [32]. It turns out that their bounding function suffers from some fundamental flaws and is clearly inferior to our proposed bounding functions. Furthermore, we show that the analysis of [33] is not satisfying.

## A.1 Description

In more detail, given a basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$, the Schnorr-Hörner pruning strategy is defined by the following bounding function, which is parameterized by an integer $p > 0$,

$$\begin{aligned} R_k^2 &= R^2 - \left(2^{-p} \operatorname{vol}(L(\mathbf{b}_1, \ldots, \mathbf{b}_{n-k}))/V_{n-k}\right)^{2/(n-k)} \\ &= R^2 - \frac{1}{\pi}\left(2^{-p} \operatorname{vol}(L(\mathbf{b}_1, \ldots, \mathbf{b}_{n-k}))\, \Gamma((n-k)/2+1)\right)^{2/(n-k)}, \end{aligned} \tag{9}$$

where $V_{n-k} = V_{n-k}(1)$ is the volume of the unit ball in $n-k$ dimensions. Note that [33] used a different description than the one we gave, but both descriptions can be shown to be equivalent.

In the full version, we rigorously analyze this pruning strategy, and show that it is inferior to our extreme pruning (see also Table 1). Here, we briefly mention several disadvantages. First, the fact that the bounding function depends on a parameter $p$ is undesirable; the analysis of [33] does not give any clear indication on the optimal choice of $p$. Second, the bounding function may not be positive when $p$ is too small, in which case failure is certain. Third, even for larger values of $p$, the bounding function may initially decrease, in which case some nodes enumerated in the top levels of the tree are guaranteed to lead to a dead end. In other words, by replacing their bounding function $R_1^2, \ldots, R_n^2$ with the bounding function defined by $R'^2_k = \min(R_k^2, \ldots, R_n^2)$, we obtain exactly the same success probability at a lower running time.

## A.2 The Analysis of Schnorr and Hörner

We now present some of our observations regarding the original analysis given by Schnorr and Hörner [33, Sect. 3] and why we believe it is flawed. It should be stressed that the analysis presented there is quite terse, and there is a possibility that our interpretation of it is not what the authors had in mind; yet we think that there is sufficient evidence to suggest that the use of their pruning function should be avoided and we feel that it is important to bring this to the community's attention.

The core of their analysis is [33, Thm. 2], which states that if $x_{n+2-k}, \ldots, x_n \in \mathbb{Z}$ are fixed, and if $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ is a random basis of $L$ such that its Gram-Schmidt coefficients $\mu_{i,j}$ are independent and uniformly distributed modulo 1, then the

vector $\mathbf{t} = x_{n+2-k}\mathbf{b}_{n+2-k} + \cdots + x_n\mathbf{b}_n$ is such that $\mathbf{t} - \pi_{n+2-k}(\mathbf{t})$ is uniformly distributed modulo the lattice $\bar{L}$ spanned by $\mathbf{b}_1, \ldots, \mathbf{b}_{n+1-k}$, which implies, by [33, Lemma 1], that the expectation $E$ of the number of $(x_1, \ldots, x_{n+1-k}) \in \mathbb{Z}^{n+1-k}$ such that $\|x_1\mathbf{b}_1 + \cdots + x_n\mathbf{b}_n\| \leq R$ is $V_{n+1-k}(\sqrt{R^2 - \|\pi_{n+2-k}(\mathbf{t})\|^2})/\mathrm{vol}(\bar{L})$. And [33] seems to interpret $E$ as the expectation of the number of leaves (in the enumeration tree) which derive from the node $\pi_{n+2-k}(\mathbf{t})$ at depth $k-1$.

Then, [33] claims that Thm. 2 implies that the probability that the pruned enumeration misses the shortest vector is at most $2^{-p}c$ where $c$ is said to be experimentally proportional to $c_{p,n}2^p$ for random LLL-reduced bases, where $c_{p,n}$ decreases to 0 as $p$ increases. The failure probability is claimed to be experimentally $\leq 0.9$ for $n < 30$ and $p = 7$. Finally, [33] claims that under heuristic arguments, they can show that for $p > \log_2 n$: given a random basis $(\mathbf{b}_1, \ldots, \mathbf{b}_n)$ of $L$ and a bound $R \leq \|\mathbf{b}_1\|$, their pruned enumeration performs on the average only $O(n^2 2^p)$ arithmetic steps to output a lattice vector $\mathbf{v} \in L$ such that $\|\mathbf{v}\| \leq R$ if $R \geq \lambda_1(L)$, or nothing if $R < \lambda_1(L)$. However, no proof is provided, and the claim looks suspicious: indeed, by taking $p = \lceil \log_2 n \rceil$, it would imply polynomial-time algorithms for the shortest vector problem (which is NP-hard under randomized reductions), because $n^2 2^p$ is polynomial in $n$.

There are further problems in the analysis of [33]. First of all, the assumption in Thm. 2 that the $\mu_{i,j}$'s of a random reduced basis are uniformly distributed is not supported by experiments: the experiments of [24] show that the distribution of the coefficients $\mu_{i,i-1}$ of a random LLL-reduced basis is far from being uniform. More importantly, the use of Thm. 2 in [33] to analyze the success probability of pruned enumeration is improper: if one selects $(x_{n+2-k}, \ldots, x_n)$ as the last $k-1$ coordinates of a fixed lattice vector, then these coordinates depend on the random basis, and therefore Thm. 2 cannot be applied. Similarly, the expectation $E$ cannot be viewed as the number of leaves in the enumeration tree which derive from the node $\pi_{n+2-k}(\mathbf{t})$ at depth $k-1$, because when the random basis varies, the tree varies too, so for any choice of $x_{n+2-k}, \ldots, x_n \in \mathbb{Z}$, the node $\pi_{n+2-k}(\mathbf{t})$ may appear in one tree, but not in other trees, so this expectation of the number of leaves cannot be properly defined.

## B Pseudo-code of the Pruned Enumeration Code

Here we provide the pseudo-code used to implement the enumeration algorithm in our experiments (see Algorithm 2): a detailed explanation appears in the full version. The code is based on the original Schnorr-Euchner enumeration algorithm [32], with several minor modifications. The first easy modification (see Line 10) is that we support a general bounding function $R_1 \leq \cdots \leq R_n$. The second modification (see Lines 1, 15-17, and 25) is a certain optimization that seems to give in practice a speedup by about 40%. To the best of our knowledge, this improvement has not appeared yet in the literature nor in any software package. Finally, another very minor modification is that we abort the procedure as soon as a vector shorter than $R = R_n$ is found.

**Algorithm 2.** Pruned Enumeration

**Input:** A basis $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$
A bounding function $R_1^2 \leq \cdots \leq R_n^2$
The Gram-Schmidt coefficient matrix $\mu$ (a lower-triangular matrix with ones on the diagonal), together with the norms of the Gram-Schmidt vectors $\|\mathbf{b}_1^*\|^2, \ldots, \|\mathbf{b}_n^*\|^2$.
**Output:** The coefficients of a lattice vector satisfying the bounds (if it exists)

$\sigma \leftarrow (0)_{(n+1)\times n}$; $r_0 = 0; r_1 = 1; \cdots; r_n = n$
$\rho_1 = \rho_2 = \cdots = \rho_{n+1} = 0$; *// partial norm*
$v_1 = 1; v_2 = \cdots = v_n = 0$ *// current combination*
$c_1 = \cdots = c_n = 0$ *// centers*
$w_1 = \cdots = w_n = 0$ *// jumps*
last_nonzero $= 1$; *// largest $i$ for which $v_i \neq 0$; zero if all $v_i = 0$*
$k = 1$;
**while** true **do**
  $\rho_k = \rho_{k+1} + (v_k - c_k)^2 \cdot \|\mathbf{b}_k^*\|^2$ *// compute norm squared of current node*
  **if** $\rho_k \leq R_{n+1-k}^2$ *(we are below the bound)* **then**
    **if** $k = 1$ **then**
      **return** $(v_1, \ldots, v_n)$; *(solution found; program ends)*
    **else**
      $k \leftarrow k - 1$ *// going down the tree*
      $r_{k-1} \leftarrow \max(r_{k-1}, r_k)$ *// to maintain the invariant for $j < k$*
      **for** $i = r_k$ **downto** $k + 1$ **do** $\sigma_{i,k} \leftarrow \sigma_{i+1,k} + v_i \mu_{i,k}$ **endfor**
      $c_k \leftarrow -\sigma_{k+1,k}$ *// $c_k \leftarrow -\sum_{i=k+1}^{n} v_i \mu_{i,k}$*
      $v_k \leftarrow \lfloor c_k \rceil$; $w_k = 1$
    **end if**
  **else**
    $k \leftarrow k + 1$ *// going up the tree*
    **if** $k = n + 1$ **then**
      **return** $\emptyset$ *(there is no solution)*
    **end if**
    $r_{k-1} \leftarrow k$ *// since $v_k$ is about to change, indicate that $(i, j)$ for $j < k$ and $i \leq k$ are not synchronized*
    *// update $v_k$*
    **if** $k \geq$ last_nonzero **then**
      last_nonzero $\leftarrow k$
      $v_k \leftarrow v_k + 1$;
    **else**
      **if** $v_k > c_k$ **then** $v_k \leftarrow v_k - w_k$ **else** $v_k \leftarrow v_k + w_k$
      $w_k \leftarrow w_k + 1$
    **end if**
  **end if**
**end while**