Fast Lattice Point Enumeration with Minimal Overhead*

Daniele Micciancio
UCSD
daniele@eng.ucsd.edu

Michael Walter
UCSD
miwalter@eng.ucsd.edu

Abstract

Enumeration algorithms are the best currently known methods to solve lattice problems, both in theory (within the class of polynomial space algorithms), and in practice (where they are routinely used to evaluate the concrete security of lattice cryptography). However, there is an uncomfortable gap between our theoretical understanding and practical performance of lattice point enumeration algorithms. The algorithms typically used in practice have worst-case asymptotic running time $2^{O(n^2)}$, but perform extremely well in practice, at least for all values of the lattice dimension for which experimentation is feasible. At the same time, theoretical algorithms (Kannan, Mathematics of Operation Research 12(3):415-440, 1987) are asymptotically superior (achieving $2^{O(n \log n)}$ running time), but they are never used in practice because they incur a substantial overhead that makes them uncompetitive for all reasonable values of the lattice dimension n. This gap is especially troublesome when algorithms are run in practice to evaluate the concrete security of a cryptosystem, and then experimental results are extrapolated to much larger dimension where solving lattice problems is computationally infeasible.

We introduce a new class of (polynomial space) lattice enumeration algorithms that simultaneously achieve asymptotic efficiency (meeting the theoretical $n^{O(n)} = 2^{O(n \log n)}$ time bound) and practicality, matching or surpassing the performance of practical algorithms already in moderately low dimension. Key technical contributions that allow us to achieve this result are a new analysis technique that allows us to greatly reduce the number of recursive calls performed during preprocessing (from super exponential in n to single exponential, or even polynomial in n), a new enumeration technique that can be directly applied to projected lattice (basis) vectors, without the need to remove linear dependencies, and a modified block basis reduction method with fast (logarithmic) convergence properties. The last technique is used to obtain a new SVP enumeration procedure with $\tilde{O}(n^{n/2e})$ running time, matching (even in the constant in the exponent) the optimal worst-case analysis (Hanrot and Stehlé, CRYPTO 2007) of Kannan's theoretical algorithm, but with far superior performance in practice.

We complement our theoretical analysis with a preliminary set of experiments that not only support our practicality claims, but also allow to estimate the cross-over point between different versions of enumeration algorithms, as well as asymptotically faster (but not quite practical) algorithms running in single exponential $2^{O(n)}$ time and space.

1 Introduction

Enumeration algorithms are a fundamental tool used to solve many computational problems on point lattices that arise in cryptanalysis, as well as in several other important areas of mathematics

*Research supported in part by the DARPA PROCEED program and NSF grant CNS-1117936. Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or NSF.

and computer science, including Diophantine approximation, communication theory and combinatorial optimization. The usefulness of enumeration algorithms to solve lattice problems is well supported both by theoretical and practical considerations. A distinguishing feature of enumeration algorithms is their small memory footprint: the space complexity of lattice point enumeration algorithms is essentially linear in the input size. Moreover, they have good asymptotic runtime, with the best theoretical algorithm (due to Kannan [15], and further analyzed in [14, 13]) achieving $2^{O(n \log n)}$ worst-case time complexity, where n is the dimension of the lattice. We remark that asymptotically faster algorithms to solve lattice problems in single exponential time $2^{O(n)}$ are known (e.g., see [3, 5, 24, 23]), but they all require exponential space $2^{O(n)}$ as well, and Kannan's algorithm currently achieves the best known asymptotic running time upper bound within the important class of polynomial (or even subexponential) space algorithms. On the practical side, due to their simplicity and low memory requirements, implementations of enumeration algorithms perform extremely well in practice, allowing to run experiments in moderately large dimension, despite their worst-case exponential complexity.

However, there is an uncomfortable gap between the theoretical versions of lattice enumeration algorithms, and their practical counterparts. On the one hand, the asymptotic performance of the enumeration algorithms typically used in practice[1] [29, 7] is $2^{O(n^2)}$, much worse than the $2^{O(n \log n)}$ running time of Kannan's algorithm. On the other hand, Kannan's algorithm has been often reported (see e.g., [13]) not to be competitive in practice for all problem sizes where any algorithm can be feasibly run, due to its substantial preprocessing overhead. As a result, the asymptotically faster algorithm of Kannan is considered primarily of theoretical interest, and it is never used in practice. This gap is not much of a concern when algorithms are used in cryptanalysis to show that a proposed cryptosystem is not secure: a broken challenge is a broken challenge, regardless of the algorithm used to break it. But the situation is quite different in the study of lattice based cryptography [2, 26, 22], where cryptographic functions are supported by theoretical proofs of security, and they are widely believed to be computationally hard to break, at least for large enough values of the security parameter. In this setting, the complexity of the most efficient known attacks against the underlying mathematical problem is estimated to choose appropriate values of the security parameters and evaluate the concrete security level offered by the resulting functions. In particular, one is interested in the performance of these algorithms on problem instances that are just too big to be solved in practice. Many of the most successful attacks employ enumeration algorithms, be it for solving the problem directly or as subroutine for an approximation algorithm. Clearly, extrapolating the running times of either algorithms with bad practical performance or bad asymptotic behavior is problematic. In order to get realistic predictions, one needs algorithms that simultaneously achieve good practical performance (for experimental evaluation in low dimension) and good asymptotic behavior (for meaningful extrapolation).

In this paper we set to solve this problem, and introduce a new class of enumeration/preprocessing strategies that allow to achieve $2^{O(n \log n)}$ running time (or even $\tilde{O}(n^{n/2e})$ for the shortest vector problem, SVP), matching the asymptotic worst-case time complexity of Kannan's algorithm [15, 13], without introducing any substantial overhead. Experimental evaluation shows that the resulting algorithms are quite practical indeed: the preprocessing overhead can be made arbitrarily small, making the algorithms competitive with (or even superior to) the traditional enumeration methods already in small dimension.

[1] Stronger preprocessing using block reduction algorithms (as suggested in [7], and discussed later in the paper) can be used to speed up enumeration, but not to the same level as Kannan's algorithm.

The preprocessing overhead of enumeration algorithms. Enumeration algorithms for the solution of lattice problems, first proposed by Fincke and Pohst [8] and Kannan [15], are an instance of the general algorithmic technique called *branch and bound*. In the following we assume some minimal familiarity with enumeration algorithms, or at least lattice problems, and refer the reader to the semi-tutorial paper [1] for more background. Given a lattice basis $\mathbf{B}[\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{R}^{m \times n}$, SVP asks to find the shortest non-zero vector $\mathbf{v} = \sum_i \mathbf{b}_i x_i$ (where the x_i's are integer coefficients) in the corresponding lattice. Note that vectors can only get shorter under orthogonal projections. Enumeration algorithms exploit this fact by recursively enumerating all short vectors in the projected lattice orthogonal to $\mathbf{b}_1$ and, for each of them, collecting all short preimages under the projection using $\mathbf{b}_1$. Finally, in the top level of the recursion the shortest vector is selected from the list and returned.

The set of possible values for each variable x_i is roughly proportional to the inverse length $1/\|\mathbf{b}_i^*\|$ of the component of $\mathbf{b}_i$ orthogonal to $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$. If the basis is preprocessed using the LLL basis reduction algorithm [20] (or one of its block variants, like BKZ, with block size typically fixed in the range 20-30) then each variable can take up to $2^{O(n)}$ possible values, resulting in $2^{O(n^2)}$ overall running time (with a constant in the exponent dependent on the strength of the basis reduction). This is the method most commonly used in practice, as polynomial time LLL or BKZ preprocessing is very cheap, and only takes a negligible fraction of the total running time of enumeration.

In an ingenious paper [15] (see also [17, 16], as well as the recent analysis [13]), Kannan showed that if the basis $\mathbf{B}$ satisfies a stronger notion of reducedness (namely, quasi-Hermite-Korkine-Zolotarev, quasi-HKZ-reduction) the runtime of enumeration drops down to $2^{O(n \log n)}$. Moreover, quasi-HKZ-reduction can also be computed in time $2^{O(n \log n)}$ by means of a polynomial number of recursive calls to HKZ-reduction in dimension $n - 1$. This leads to an SVP algorithm with overall running time $2^{O(n \log n)}$, much better than a straightforward application of enumeration [8]. Unfortunately, the runtime of the final SVP enumeration dominates the recursive preprocessing only asymptotically: in practice, for all reasonable values of the dimension n, the quasi-HKZ preprocessing is not only much slower than the final enumeration (with a quasi-HKZ basis), but even slower than a straightforward application of enumeration with a weakly reduced (e.g., LLL) basis, as commonly run in practice [8, 29, 1, 7]. As a result, Kannan's algorithm is not competitive, and it is never used, despite its superior asymptotic performance.

While the exact source of inefficiency in Kannan's algorithm is not fully understood (beyond the fact that it can be attributed to the preprocessing time), one possible way to make sense of the overhead is to count the *total* number of recursive calls performed by the algorithm. In [14], Helfrich gave an improved analysis of Kannan's preprocessing showing that the top level HKZ reduction performs at most $O(\log n)$ recursive calls to HKZ reduction in dimension $n - 1$. While better than Kannan's original polynomial bound, this still leads to a superexponential number of recursive calls $O(\log n)^n = 2^{O(n \log \log n)}$. So, as a guiding principle in the search for better enumeration algorithms (with good asymptotic running time, but more lightweight preprocessing) one may ask: is it possible to design a preprocessing algorithm that only performs a single exponential $2^{O(n)}$ (or even polynomial $n^{O(1)}$) total number of recursive calls, and still produces bases of quality comparable to Kannan's?

Our results. We provide a positive answer to this question, showing that lattice point enumeration can be performed in time $2^{O(n \log n)}$ while making only a polynomial (in fact, linear) in n

total number of recursive calls during the preprocessing of the basis. More generally, we describe a new family of (polynomial space) enumeration algorithms (parametrized by two integers k, h), and evaluate them both by means of theoretical worst-case asymptotic analysis, and by running experiments in moderate dimension. Special cases of our algorithm include the Fincke-Pohst algorithm (for $k = n$) and a lightweight variant of the algorithm of Kannan (for $k = 1$) described in the next paragraph. Our analytical study shows that values up to $k = O(\log n)$ achieve asymptotic runtime comparable to Kannan's algorithm. One variant of our algorithm even achieves provable $\tilde{O}(n^{n/2e})$ worst-case running time for SVP, matching the optimal analysis of [13] also in the exponent, still maintaining low overhead. Practical experimentation with our algorithm suggests that intermediate values of $k = \Theta(\log n)$ are best, leading to a new enumeration/preprocessing strategy with $2^{O(n \log n)}$ asymptotic worst-case running time and minimal overhead. Beside being asymptotically fast, the resulting algorithms are competitive in practice with traditional enumeration methods already in moderately low dimension $n < 30$, and clearly outperform them as the dimension n gets higher. Finally, we suggest an interesting variant of our algorithm that combines enumeration techniques with the new Voronoi preprocessing method of [23], while keeping the overall space complexity of the algorithm polynomial in the dimension n.

Techniques. Several main technical ideas are at the basis of our improved enumeration algorithm. We recall that Kannan's preprocessing algorithm is based on the following weaker form of HKZ reduction: a basis $\mathbf{B}$ is quasi-HKZ reduced if $\mathbf{B}$ is LLL reduced, and the projection $\pi_1(\mathbf{B})$ of $\mathbf{B}$ orthogonally to its first vector $\mathbf{b}_1$ is HKZ reduced. In [15], quasi-HKZ reduced bases are computed by repeatedly applying the LLL algorithm to the whole basis $\mathbf{B}$, and (recursively) HKZ-reduction to $\pi_1(\mathbf{B})$. (Notice that after HKZ reducing $\pi_1(\mathbf{B})$, the basis $\mathbf{B}$ may not be LLL reduced anymore, and one needs to repeatedly apply both algorithms up to $O(\log n)$ times [14].)

Our first technical contribution is to observe that after LLL reducing a basis $\mathbf{B}$ and recursively HKZ reducing the projected basis $\pi_1(\mathbf{B})$, it is already possible to bound the running time of enumeration by $2^{O(n \log n)}$, even if the basis may not be LLL reduced anymore (and therefore, it may not be quasi-HKZ reduced.) So, repeatedly calling LLL and recursive HKZ reduction is not needed, and an HKZ reduced basis can be obtained by a single enumeration (with $2^{O(n \log n)}$ running time) and two recursive calls to HKZ reduction in dimension $n-1$ (one before, and one after enumeration). This already leads to an algorithm with good asymptotic performance ($2^{O(n \log n)}$, including preprocessing) and total number of recursive calls equal to 2^n, asymptotically smaller than Kannan's $2^{O(n \log \log n)}$. We refer to this first algorithm as the *lightweight Kannan* algorithm. We remark that, as reported in previous experimental studies, Kannan's original algorithm often makes only two (top level) recursive calls in practice, rather than the $O(\log n)$ worst-case bound. So, the lightweight Kannan algorithm cannot be expected to be much faster than Kannan's in practice. However, the $2^{O(n \log n)}$ provable time bound of the lightweight Kannan algorithm provides a good basis for our second improvement.

Our second technical idea is that instead of making recursive calls in dimension $n - 1$, one can make recursive calls in dimension $n - k$. As long as k is reasonably small (e.g., for constant $k = O(1)$), we can still prove that enumeration runs in time $2^{O(n \log n)}$, while the total number of recursive calls is reduced to c^n, where the base $c < 2$ can be made arbitrarily close to 1. Theoretically, this generalized enumeration/preprocessing algorithm runs in time $n^{O(n-k)} \cdot 2^{kn}$. Since enumeration algorithms have exponential complexity, making recursive calls in dimension $n-k$ rather than $n - 1$ can have a substantial impact on the practical performance of the algorithm,

and the total overhead introduced by preprocessing the basis. In fact, our experimental results suggest that a moderately small value of $k = \Theta(\log n)$ leads to the best performance in practice (while maintaining good asymptotic runtime) already in relatively small dimension $n \approx 30$. The total number of recursive calls, for $k = \Theta(\log n)$, is $2^{O(n/\log n)} = 2^{o(n)}$, slightly subexponential in n.

Our third algorithmic contribution is a new enumeration algorithm that allows to find the shortest vector in a projected lattice $\pi(\mathbf{B})$, without first removing the linear dependencies among the projected basis vectors. We recall that the standard enumeration method, on input $\mathbf{b}_1, \ldots, \mathbf{b}_n$, has running time inversely proportional to the length of the Gram-Schmidt projected vectors $\|\mathbf{b}_i^*\|$. If the input vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ are linearly dependent, then some orthogonalized vector $\mathbf{b}_i^* = \mathbf{0}$ necessarily vanishes, and the standard enumeration algorithm does not even terminate! This precludes the application of enumeration algorithms to solve lattice problems on projected bases, as those that arise in HKZ computation, where after finding a shortest lattice vector $\mathbf{v}$, one needs to recursively HKZ reduce the projection of the lattice basis orthogonally to $\mathbf{v}$. Linear dependencies can be easily removed by running the LLL algorithm [6], but this forces a second recursive call to HKZ, leading to a total number of recursive calls $2^{O(n/k)}$, which is still close to exponential in n. We present a new enumeration algorithm for projected lattices that works even in the presence of linear dependencies, and allows to compute an HKZ basis purely by enumeration, after the initial preprocessing performed before finding $\mathbf{v}$. This new algorithm makes only a single (top level) recursive call, leading to a total number of recursive calls linear in the dimension.

In [13] it is shown that Kannan's algorithm for SVP has running time $\tilde{O}(n^{n/2e})$. So, it is natural to ask if our SVP algorithms obtained using the above techniques also achieve the same constant $1/2e \approx 0.18$ in the exponent. Not surprisingly, building on the results of [13], we prove that for typical preprocessing results (e.g., bases with non increasing orthogonalized lengths $\|\mathbf{b}_i^*\|$) our SVP algorithm also has $\tilde{O}(n^{n/2e})$ worst-case running time. However, it is not clear how to adapt the proof of [13] to atypical bases where the orthogonalized lengths $\|\mathbf{b}_i^*\|$ are arbitrary. Our last technical contribution is a variant of our algorithms based on a modified block basis reduction method that achieves the worst-case running time bound $\tilde{O}(n^{n/2e})$, while maintaining similarly low overhead, and comparable performance in practice. This algorithmic variant with optimal worst-case exponent retains the idea of making recursive calls in dimension $n-k$ substantially lower than n, but makes multiple calls, resulting in a superpolynomial (but still subexponential $2^{o(n)}$) total number of recursive calls.

Experiments While the main contribution of this paper is theoretical, we also implemented our algorithms to assess their practical performance on random lattices, as typically encountered in applications like block basis reduction. Our experiments were aimed primarily at producing useful, qualitative information about the general behavior of different algorithms (like determining cross-over points, and experimental validation of asymptotic analysis), rather than targeting single cryptanalytic challenges or solving problems in record dimensions, which requires a substantial low level optimization effort. For flexibility, most of the experiments were performed using a fast prototype implementation, with no low level coding optimizations, and performance was measured by counting the number of nodes explored during the (preprocessing and final) enumeration process, which also has the advantage of making the experimental results largely independent from technological details. Interestingly, even with such fast prototype implementation, we are able to make several interesting experimental observations that both complement and support our theoretical analysis. For example, the running time of our algorithms in practice is approximately $2^{cn \log n}$,

but with a smaller constant than the theoretical $1/2e \approx 0.18$ worst case, and the exact value of c dependent on the strength of the basis reduction applied before running the algorithm. Also, traditional enumeration algorithms have running time 2^{cn^2} both in theory and in practice, but for an even smaller constant. Still, our asymptotically faster algorithm quickly catches up, and becomes attractive already in dimension below 50 thanks to its reduced (and tunable) overhead. Finally, we implemented our algorithm by modifying `fpLLL`[2], which yields a much faster implementation and allows us to run experiments almost up to dimension 70. Initial experiments seem to confirm that our algorithm outperforms algorithms currently in use significantly. For more details about our experimental results, see Section 4.

Related work. Block reduction algorithms [29, 28, 10] are an effective preprocessing method to improve the running time of lattice enumeration in practice [13, 7]. Such preprocessing has running time exponential in the block size k, so they are used for fixed values of k, typically in the range $20 - 30$. We observe that for any fixed value of the block size k, the running time of enumeration after block reduction is still $2^{O(n^2)}$, though with a smaller constant in then exponent than LLL. In the asymptotic setting, one may consider using block reduction with increasing values of k, but this would inevitably require the recursive HKZ preprocessing of the blocks, and potentially very large (superpolynomial) number of recursive calls. A rigorous analysis of this variant is left to a separate work. We evaluated the effect of block reduction on different algorithms (including ours) experimentally and the results demonstrate that our algorithm still compares favorably already in moderately low dimensions.

Enumeration algorithms commonly used in practice often make use of various pruning heuristics [27, 30, 9] that can considerably speed up execution. While a rigorous analysis of pruning methods (both as a general technique, and as specifically applied to our algorithms) is outside of the scope of this paper, we consider the effect of pruning based on the claims made in [9], asserting that these techniques can speed up enumeration by a factor ranging from $2^{n/4}$ to $2^{n/2}$, depending on the aggressiveness of pruning. Note that single exponential factors do not affect the asymptotical behaviour of enumeration algorithms due to its superexponential runtime, but can have a big impact in practice. Extrapolating our experimental results and comparing to the results from [9], we expect our algorithms to outperform any enumeration algorithm currently employed in practice in high (but tractable) dimensions.

Outline. We will cover some preliminaries in Section 2, before introducing our techniques and the new algorithms in Section 3. Finally, we present our experimental results in Section 4.

2 Preliminaries

Notation We use standard asymptotic notation O, o, ω, Ω, and $\tilde{O}(f(n)) = f^{1+o(1)}(n)$. Numbers and reals are denoted by lower case letters and sets by upper case letters. For $n \in \mathbb{Z}_+$ we denote the set $\{1, \ldots, n\}$ by $[n]$. For vectors we use bold lower case letters and the i-th entry of a vector $\mathbf{v}$ is denoted by v_i. Let $\langle \mathbf{v}, \mathbf{w} \rangle = \sum_i v_i \cdot w_i$ be the scalar product of two vectors, and $\|\mathbf{v}\| = \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$ the standard Euclidean norm. We define the projection of a vector $\mathbf{b}$ orthogonally to a vector $\mathbf{v}$ as $\pi_{\mathbf{v}}(\mathbf{b}) = \mathbf{b} - \frac{\langle \mathbf{b}, \mathbf{v} \rangle}{\|\mathbf{v}\|^2}\mathbf{v}$. Matrices are denoted by bold upper case letters. The j-th row of a matrix $\mathbf{B}$

[2] `http://perso.ens-lyon.fr/damien.stehle/fplll/index.html`

is denoted by $\mathbf{B}_j$, its i-th column by $\mathbf{B}_{,i}$ or as $\mathbf{b}_i$, when no confusion can arise. Furthermore, we denote the submatrix comprising the columns from the i-th to the j-th column (inclusive) as $\mathbf{B}_{[i,j]}$. We extend the projection operator to matrices, where $\pi_{\mathbf{V}}(\mathbf{B})$ is the matrix obtained by applying $\pi_{\mathbf{V}}$ to every column $\mathbf{b}_i$ of $\mathbf{B}$ and $\pi_{\mathbf{V}}(\mathbf{b}_i) = \pi_{\mathbf{v}_k}(\cdots(\pi_{\mathbf{v}_1}(\mathbf{b}_i))\cdots)$. In this work, the log function always has base 2. When we assign the result of the log function to an integer variable, it is always rounded to the smallest integer larger than or equal to the result.

Lattices A *lattice* Λ is a discrete subgroup of $\mathbb{R}^m$ and is generated by a matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$, i.e. $\Lambda = \mathcal{L}(\mathbf{B}) = \{\mathbf{Bx}\colon \mathbf{x} \in \mathbb{Z}^n\}$. If $\mathbf{B}$ has full column rank, $\mathbf{B}$ is called a *basis* of Λ and $\dim(\Lambda) = n$ is the dimension (or rank) of Λ. A lattice has infinitely many bases, which are related to each other by right-multiplication with unimodular matrices. With each matrix $\mathbf{B}$ we associate its *Gram-Schmidt-Orthogonalization* (GSO) $\mathbf{B}^*$, where the i-th column $\mathbf{b}_i^*$ of $\mathbf{B}^*$ is defined as $\mathbf{b}_i^* = \pi_{\mathbf{B}_{[1,i-1]}}(\mathbf{b}_i) = \pi_{\mathbf{B}^*_{[1,i-1]}}(\mathbf{b}_i)$ (and $\mathbf{b}_1^* = \mathbf{b}_1$). For a fixed matrix $\mathbf{B}$ we extend the projection operation to indices: $\pi_i(\cdot) = \pi_{\mathbf{B}^*_{[1,i]}}(\cdot)$.

For every lattice Λ there are a few invariants associated to it. One of them is its determinant $\det(\mathcal{L}(\mathbf{B})) = \prod_i \|\mathbf{b}_i^*\|$ for any basis $\mathbf{B}$. Even though the basis of a lattice is not uniquely defined, the determinant is and it is efficiently computable given a basis. Furthermore, for every lattice Λ we denote the length of its shortest non-zero vector (also known as the *first minimum*) by $\lambda_1(\Lambda)$, which is always well defined. We use the short-hand notations $\det(\mathbf{B}) = \det(\mathcal{L}(\mathbf{B}))$ and $\lambda_1(\mathbf{B}) = \lambda_1(\mathcal{L}(\mathbf{B}))$. Note that both quantities are well-defined for any generating system $\mathbf{B}$. Minkowski's theorem is a classic result that relates the first minimum to the determinant of a lattice. It states that $\lambda_1(\Lambda) \leq \sqrt{\gamma_n}\det(\Lambda)^{1/n}$, for any Λ with $\dim(\Lambda) = n$, where $\Omega(n) \leq \gamma_n \leq n$ is Hermite's constant. Finding a (even approximate) shortest nonzero vector in a lattice, commonly known as the *Shortest Vector Problem* (SVP), is NP-hard under randomized reductions [18, 21]. The *Closest Vector Problem* (CVP) is the inhomogeneous counterpart of SVP, and it asks to find the lattice point closest to a given target. The SVP easily reduces to CVP [12], so any algorithm for CVP can be used to solve SVP as well. In the case of enumeration algorithms, SVP and CVP are almost equivalent. We state all our results in terms of SVP, but they naturally extend to CVP.

Enumeration The high level idea of the standard enumeration procedure has already been explained in the introduction. For the sake of brevity we will not go further into detail here. For our purposes it is sufficient to note that the overall number of nodes explored by the enumeration procedure to solve SVP/CVP given a basis $\mathbf{B}$ is bounded by $\left(\prod_i \left\lfloor \frac{2r}{\|\mathbf{b}_i^*\|} \right\rfloor + 1\right)$, where r is a valid upper bound for the solution. This bound will be the starting point of our analysis. Note that it can heavily depend on the basis $\mathbf{B}$.

Basis Reduction Basis reduction algorithms deal with the problem of obtaining a "good" basis from an arbitrary basis for some notion of a "good" basis. The LLL algorithm [20] is a polynomial time basis reduction algorithm parameterized by a value $1/4 < \delta < 1$, which allows some trade-off between output quality and running time. In this work, for simplicity, we only consider $\delta = 3/4$, but all results can easily be adapted to any δ. An LLL reduced basis $\mathbf{B} \in \mathbb{Z}^{m\times n}$ (with $\delta = 3/4$) satisfies $\|\mathbf{b}_i^*\| \leq \sqrt{2}\|\mathbf{b}_{i+1}^*\|$ for all $i \in [n-1]$. A much stronger notion is *HKZ reduction*. A basis $\mathbf{B}$ is HKZ reduced if $\|\mathbf{b}_1\| = \lambda_1(\mathbf{B})$ and $\pi_1(\mathbf{B})$ is HKZ reduced. In particular, for any HKZ reduced basis we have $\|\mathbf{b}_i^*\| = \lambda_1(\pi_{i-1}(\mathbf{B}))$. Obviously, this notion inherently solves the SVP problem and, in fact, Kannan's algorithm solves SVP by producing an HKZ reduced basis.

3 A New Asymptotically Fast Enumeration Algorithm

In this section we describe and analyze our new algorithm for lattice point enumeration.

3.1 Parameterizing the Preprocessing

In order to analyze Kannan's algorithm and our variants, we first introduce a new notion of basis reduction that will facilitate the analysis later.

Definition 1 *Let $\mathbf{B} \in \mathbb{Z}^{m \times n}$, $n' = \dim(\mathcal{L}(\mathbf{B}))$, and $\zeta : [n] \to \mathbb{R}_+$. We call $\mathbf{B}$* ζ-reduced*, if for all $i \in [n]$*

$$\|\mathbf{b}_i^*\| > \zeta(i) \det(\mathbf{B})^{1/n'} \quad \Rightarrow \quad \lambda_1(\pi_{i-1}(\mathbf{B})) > \lambda_1(\mathbf{B})$$

and $\mathbf{B}_{[1,k]}$ is ζ-reduced for all $k \in [n-1]$.

Note that the definition covers arbitrary generating systems for lattices, not only bases. The quantification over the sublattices in Definition 3.1 might seem like a stringent condition at first sight but the reduction algorithms that we consider, namely LLL, BKZ and HKZ reduction, naturally meet this condition since all subbases of the form $\mathbf{B}_{[1,k]}$ of a reduced bases $\mathbf{B}$ are also reduced.

It is easy to see that an HKZ reduced basis is ζ-reduced for any constant function $\zeta(i) \geq \sqrt{n}$ by applying Minkowski's bound. The following lemma shows that the LLL algorithm computes ζ-reduced bases for an appropriate value of ζ.

Lemma 1 *Let $\mathbf{B} \in \mathbb{Z}^{m \times n}$ an LLL reduced basis. Then, $\mathbf{B}$ is ζ-reduced for $\zeta(i) = 2^{\frac{n-1}{4}}$.*

Proof Let $\mathbf{B}$ an LLL reduced basis of a lattice Λ. For any i, we prove that if $\lambda_1(\Lambda) \geq \lambda_1(\pi_{i-1}(\Lambda))$, then $\|\mathbf{b}_i^*\| \leq 2^{(n-1)/4} \det(\Lambda)^{1/n}$. Since $\mathbf{B}$ is LLL reduced, we have $\|\mathbf{b}_k^*\| \leq \sqrt{2}\|\mathbf{b}_{k+1}^*\|$ for all k. In particular, $\|\mathbf{b}_i^*\| \leq 2^{(k-i)/2}\|\mathbf{b}_k^*\|$ for all $k \geq i$, and $\|\mathbf{b}_1\| \geq \lambda_1(\Lambda) \geq \lambda_1(\pi_{i-1}(\Lambda)) \geq \min_{k \geq i} \|\mathbf{b}_k^*\| \geq \|\mathbf{b}_i^*\| 2^{-(n-i)/2}$. So, we also have $\|\mathbf{b}_i^*\| \leq 2^{(n-i+k)/2}\|\mathbf{b}_k^*\|$ for $k < i$. It follows that $\|\mathbf{b}_i^*\|^n \leq \prod_k 2^{((k-i) \bmod n)/2}\|\mathbf{b}_k^*\| = 2^{n(n-1)/4} \det(\Lambda)$ and $\|\mathbf{b}_i^*\| \leq 2^{(n-1)/4} \det(\Lambda)^{1/n}$. $\square$

Next, we analyze the runtime of the standard enumeration procedure on a ζ-reduced basis.

Theorem 1 *Let $\mathbf{B} \in \mathbb{Z}^{m \times n}$ be a ζ-reduced basis with $\zeta(i) \geq \sqrt{n}$ for all $i \in [n]$. Then there is an efficiently computable set $M \subset \mathbb{Z}^n$ with $|M| \leq 3^n \prod_{i=1}^n \zeta(i)$ such that there is a vector $\mathbf{x} \in M$ with $\|\mathbf{Bx}\| = \lambda_1(\mathbf{B})$.*

Proof Let $\Delta = \det(\mathbf{B})$ and $r = \sqrt{n}\Delta^{1/n}$ be the Minkowski bound of $\mathcal{L}(\mathbf{B})$. We start out by noting that we can assume w.l.o.g. that $\|\mathbf{b}_i^*\| \leq \zeta(i)\Delta^{1/n}$ for all $i \in [n]$, because if there is an i with $\|\mathbf{b}_i^*\| > \zeta(i)\Delta^{1/n}$, we can ignore the entire sublattice $\mathcal{L}(\mathbf{B}_{[i,n]})$ due to ζ-reducedness and apply the result recursively to the reduced basis $\mathbf{B}_{[1,i-1]}$. Now we simply bound the number of steps in the enumeration by

$$|M| \leq \prod_{i=1}^{n} \left\lfloor \frac{2r}{\|\mathbf{b}_i^*\|} + 1 \right\rfloor. \tag{1}$$

Observe that for any real $\alpha \geq 0$ we have $\lfloor 2\alpha + 1 \rfloor \leq \max\{2, 3\alpha\}$. This can easily seen to be true: If $\alpha < 1$, then $2\alpha + 1 < 3$ and $\lfloor 2\alpha + 1 \rfloor \leq 2$. Otherwise, $\alpha \geq 1$ and $2\alpha + 1 \leq 2\alpha + \alpha = 3\alpha$. Setting

$\alpha = r/\|\mathbf{b}_i^*\|$, we can bound each term in Equation 3.1 by

$$\left\lfloor \frac{2r}{\|\mathbf{b}_i^*\|} + 1 \right\rfloor \leq \max\{2, 3r/\|\mathbf{b}_i^*\|\} = \frac{r}{\|\mathbf{b}_i^*\|} \max\{2\|\mathbf{b}_i^*\|/r, 3\} \leq \frac{3\zeta(i)\Delta^{1/n}}{\|\mathbf{b}_i^*\|}$$

So, the size of M is at most $|M| \leq \prod_i \frac{3\zeta(i)\Delta^{1/n}}{\|\mathbf{b}_i^*\|} = 3^n \prod_i \zeta(i)$. □

Intuitively, Theorem 3.1 states that when calling the enumeration procedure on a ζ-reduced basis, the running time is bounded by the product of the ζ values for all basis vectors and a single exponential factor. Recall that Fincke-Pohst enumerates on LLL reduced basis, so from Lemma 3.1 and Theorem 3.1 we immediately derive the well known $2^{O(n^2)}$ bound on its running time. In contrast, Kannan's algorithm produces a ζ-reduced basis before the enumeration with $\zeta(1) = 2\sqrt{n}$ and $\zeta(i) = \sqrt{n}$ for all $i > 1$. This corresponds to the notion of quasi-HKZ-reducedness [13]. It follows that the complexity of the enumeration procedure on such bases (and by induction also the whole algorithm) is in $\tilde{O}(n^{n/2})$.

However, in order to achieve the $\tilde{O}(n^{n/2})$ complexity, requiring $\zeta(1)$ to be this small seems to be an overkill. In fact, after running LLL we obtain $\zeta(1) = 2^{\frac{n-1}{4}}$. So, after a recursive call on the basis $\pi_1(\mathbf{B})$, which does not change $\zeta(1)$, we obtain a basis that is ζ-reduced for the function $\zeta(1) = 2^{\frac{n-1}{4}}$, $\zeta(i > 1) = \sqrt{n}$. Clearly, using this ζ function in Theorem 3.1 also exhibits a worst-case complexity of $\tilde{O}(n^{n/2})$. This *lightweight Kannan* algorithm reduces the number of recursive calls before enumeration from $\log(n)$ to 1 and so the overall number of recursive calls to 2^n, while preserving the asymptotic runtime.

Furthermore, our analysis suggests a natural generalization. We can introduce a degree of freedom by allowing a variable number $\kappa(n)$ of basis vectors to have an exponential ζ-bound as opposed to the sublinear bound obtained by the recursive call. This can be achieved by first LLL reducing the basis $\mathbf{B}$ and recursing on the basis $\pi_{\kappa(n)}(\mathbf{B})$ before enumerating. This will result in bounds of $\zeta(i \leq \kappa(n)) = 2^{\frac{n-1}{4}}$, $\zeta(i > \kappa(n)) = \sqrt{n}$, which, when plugged into Theorem 3.1, yield an upper bound of $\tilde{O}(2^{n\kappa(n)/4} n^{(n-\kappa(n))/2})$. Note that for $\kappa(n) = 1$ this variant corresponds to the lightweight Kannan algorithm, while for $\kappa(n) = n$ it degenerates to the Fincke-Pohst algorithm. Using intermediate values for $\kappa(n)$ allows us to interpolate between these two algorithms and thus this variant can be seen as a generalization of them. The new parameter can be used to balance the preprocessing with the enumeration: for larger values of $\kappa(n)$, the recursive call is cheaper, but enumeration is harder, and vice versa. In order to maintain an asymptotic upper bound of $n^{O(n)}$ we need to ensure that $\kappa(n) = O(\log(n))$, which results in a runtime of $\tilde{O}(n^{0.75n})$, only a little worse than the one for the lightweight Kannan of $\tilde{O}(n^{n/2})$, but with significantly lighter preprocessing.

We summarize the runtimes of the algorithms discussed in this section in Table 1.

3.2 Enumeration in Projected Lattices

Even though the methods from the previous section can be used to solve SVP with only 1 top level recursive call to HKZ (in lower dimension), the total number of recursive calls to HKZ is still exponential because a second top level recursive call is performed during postprocessing to extend the shortest vector into a full HKZ basis. Here we will show that we can eliminate the recursive call in the postprocessing (which will bring down the overall number of recursive calls to $O(n)$).

	Kannan	General	FinckePohst	Lightweight Kannan	Our
$\kappa(n)$	1	variable	n	1	$\log n$
$\zeta(i \leq \kappa(n))$	$\sqrt{2n}$	$2^{(n-1)/4}$	$2^{(n-1)/4}$	$2^{(n-1)/4}$	$2^{(n-1)/4}$
$\zeta(i > \kappa(n))$	$\sqrt{n}$	$\sqrt{n}$	-	$\sqrt{n}$	$\sqrt{n}$
Bound	$\sqrt{2}n^{n/2}$	$2^{n\kappa(n)/4}n^{(n-\kappa(n))/2}$	$2^{n^2/4}$	$2^{n/4}n^{n/2}$	$n^{0.75n}$
# Recursive Calls	$2^{n \log\log n}$	$2^{n/\kappa(n)}$	n	2^n	$2^{n/\log n}$

Table 1: Comparison of asymptotic runtime of different algorithms according to Theorem 3.1 where we ignore the factor 3^n, since it is the same across all algorithms. *General* refers to the generalized algorithm and *Our* to the one where we set $\kappa(n) = \log n$.

Dealing with Linear Dependencies Our goal in this paragraph is to HKZ reduce a basis $\mathbf{B}$, in which the shortest vector $\mathbf{v}$ has already been found. In order to do so, we note that after projecting the basis orthogonally to $\mathbf{v}$, the columns of $\mathbf{B}$ are not linearly independent anymore, so $\pi_{\mathbf{v}}(\mathbf{B})$ is not a basis. For each basis vector $\mathbf{b}_i$ the enumeration procedure decides how many nodes to explore depending on the value $r/\|\mathbf{b}_i^*\|$ (where the bound r is part of the input). If $\|\mathbf{b}_i^*\| = 0$, as is the case for vectors $\mathbf{b}_i$ of the generating system that are linearly dependent on previous vectors $\mathbf{B}_{[1,i-1]}$, this does not make sense anymore. Now let $\mathbf{B}^*$ be the GSO of the projected basis $\pi_{\mathbf{v}}(\mathbf{B})$. Let $\mathbf{v} = \mathbf{B}\mathbf{w}$ and note that $\|\mathbf{b}_i^*\| = 0$ iff w_i is the last non-zero entry of $\mathbf{w}$. We can add arbitrary multiples of $\mathbf{w}$ to any coefficient vector $\mathbf{x}$ without changing the length of $\pi_{\mathbf{v}}(\mathbf{B}\mathbf{x})$. This means that for any vector, and in particular any shortest non-zero vector, in $\mathcal{L}(\pi_{\mathbf{v}}(\mathbf{B}))$ there is a vector $\pi_{\mathbf{v}}(\mathbf{B}\mathbf{x})$ of the same length and with $0 \leq x_i < w_i$. The following lemma formalizes this idea and generalizes it to projecting the basis to arbitrarily many vectors.

Lemma 2 *Let $\mathbf{B} \in \mathbb{Z}^{m\times n}$ be the basis of an n-dimensional lattice Λ and $\mathbf{V} = \mathbf{B}\mathbf{W}$ a basis of a j-dimensional sublattice $\mathcal{L}(\mathbf{V}) \subset \Lambda$, i.e. $\mathbf{W} \in \mathbb{Z}^{n\times j}$. We assume the indices of the last non-zero entries of each column in $\mathbf{W}$ are pairwise different. Let $t_i = \max\{t\colon \mathbf{W}_{t,i} \neq 0\}$ be the indices of the last non-zero entry in each column of $\mathbf{W}$ and $I = \{t_1, \ldots, t_j\}$. Then, for any vector $\mathbf{y} \in \mathbb{Z}^n$ there exists a vector $\mathbf{x} \in \mathbb{Z}^n$ with $0 \leq x_{t_i} < |\mathbf{W}_{t_i,i}|$ for all $t_i \in I$ such that $\|\pi_{\mathbf{V}}(\mathbf{B}\mathbf{x})\| = \|\pi_{\mathbf{V}}(\mathbf{B}\mathbf{y})\|$.*

Proof Observe, that given a vector $\mathbf{y} \in \mathbb{Z}^n$ we can add arbitrary multiples of any column $\mathbf{W}_{\cdot,i}$ to $\mathbf{y}$ without changing the projected length $\|\pi_{\mathbf{V}}(\mathbf{B}\mathbf{y})\|$, because $\mathbf{V} = \mathbf{B}\mathbf{W}$. It follows that we can iterate over the t_i's in I in decreasing order (w.r.t. t_i) and reduce y_{t_i} modulo $\mathbf{W}_{t_i,i}$ by adding the correct multiple of $\mathbf{W}_{\cdot,i}$ to $\mathbf{y}$. □

Note that Lemma 3.2 immediately yields an enumeration procedure to find the shortest vector in a projected lattice using the coefficient matrix $\mathbf{W}$ of the vectors we are projecting orthogonally to. Specifically, the procedure needs to enumerate a number of cosets of the lattice for each vector in $\mathbf{V}$. We present pseudocode for the new procedure in Algorithm 3.1. The assumption that the indices of the last nonzero entries in the columns of $\mathbf{W}$ are pairwise different is not a limitation, because we can transform $\mathbf{W}$ into Hermite Normal Form (HNF) in polynomial time and linear space [25]. This transformation will not change the projected basis $\pi_{\mathbf{V}}(\mathbf{B})$ since it does not change the lattice $\mathcal{L}(\mathbf{V})$.

Lemma 3.2 implies that we can introduce another degree of freedom to our algorithm, which we will call η, to specify how many vectors of an HKZ reduced bases should be enumerated directly before making a recursive call.

Algorithm 1 Enumeration procedure for linearly dependent generating systems

`procedure` Enum ($\mathbf{B}$, $\mathbf{W}$, r)

Input: A lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, a coefficient matrix $\mathbf{W} \in \mathbb{Z}^{n \times j}$ in HNF, and an upper bound $r \in \mathbb{R}$

Output: All lattice vectors in $\pi_{\mathbf{BW}}(\mathcal{L}(\mathbf{B}))$ with Euclidean norm less than r

```
1   if n = 0 return {0}
2   S' = Enum(π_1(B), W_{-1}, r)        \\W_{-1} denotes W without its first row
3   S = ∅
4   for all v ∈ S'
5     if ‖π_BW(b_1)‖ > 0 then
6       S = S ∪ {v + x b_1 | x ∈ Z and ‖π_BW(v + x b_1)‖ ≤ r}
7     else
8       \\denote by w the unique entry in W_1 s.t. w is the only non-zero entry in its column
9       S = S ∪ {v + x b_1 | x ∈ Z and 0 ≤ x < w}
10  return S
```

Technical comments about Algorithm 3.1 The vectors in V' are lattice vectors in the projected lattice. Technically, they need to be lifted to the current dimension. The reader can just think of the algorithm working on linear combinations of $\mathbf{B}$ and the addition in line `6` and `9` as prepending a coordinate – then the lifting is implicit. The case in line `6` refers to the traditional enumeration, the set of coordinates to enumerate is efficiently computable and inverse proportional to $\|\mathbf{b}_i^*\|$.

Runtime Analysis of the Enumeration With the change of the enumeration procedure we have to revisit the proof of Theorem 3.1 to take projected lattices into account. The following theorem relates the worst-case running time of the modified enumeration procedure to the ζ-reducedness of the projected basis, which in turn will be analyzed in the next paragraph.

Theorem 2 *Let $\mathbf{B}$, $\mathbf{V}$, $\mathbf{W}$, and I as in Lemma 3.2. Additionally, let $\pi_{\mathbf{V}}(\mathbf{B})$ be ζ-reduced for some function ζ with $\zeta(i) \geq \sqrt{n}$ for all $i \in [n]$. Then there is an efficiently computable set $M \subset \mathbb{Z}^n$ with $|M| \leq 3^{n-j} \prod_{i \notin I} \zeta(i)$ such that there is a vector $\mathbf{x} \in M$ with $\|\pi_{\mathbf{V}}(\mathbf{Bx})\| = \lambda_1(\pi_{\mathbf{V}}(\mathbf{B}))$.*

Proof Let $\tilde{\mathbf{B}} = \pi_{\mathbf{V}}(\mathbf{B})$ and $\tilde{r} = \sqrt{n-j}\det(\tilde{\mathbf{B}})^{1/(n-j)}$ the Minkowski bound for $\mathcal{L}(\tilde{\mathbf{B}})$. From the discussion in the previous paragraph (cf. Algorithm 3.1) it is clear that the size of the search space is now bounded by

$$|M| \leq \left(\prod_{i=1}^{j} \mathbf{W}_{t_i,i}\right)\left(\prod_{i \notin I} \left\lfloor \frac{2\tilde{r}}{\|\tilde{\mathbf{b}}_i^*\|} + 1 \right\rfloor\right). \tag{2}$$

We start by bounding the first term. For that let $\mathbf{B}' = \mathbf{B} \backslash \{\mathbf{b}_t : t \in I\}$, i.e. the basis with the same basis vectors as $\mathbf{B}$ but without the vectors whose GSOs vanish after projecting orthogonally to $\mathbf{V}$. Furthermore, let $\mathbf{B}''$ be the basis with the same vectors as $\mathbf{B}$ but each vector in $\{\mathbf{b}_{t_i} : t_i \in I\}$ replaced by $\mathbf{v}_i$ respectively. Then we have $\det(\mathbf{V}) \prod_{i \notin I} \|\tilde{\mathbf{b}}_i^*\| = \det(\mathbf{V}) \det(\pi_{\mathbf{V}}(\mathbf{B}') = \det(\mathbf{V}|\mathbf{B}') = \det(\mathbf{B}'') = \det(\mathbf{B}) \prod_{t_i \in I} \mathbf{W}_{t,i}$ and so $\prod_{t_i \in I} \mathbf{W}_{t_i,i} = \frac{\det(\mathbf{V}) \prod_{i \notin I} \|\tilde{\mathbf{b}}_i^*\|}{\det(\mathbf{B})} = \prod_{i \notin I} \|\tilde{\mathbf{b}}_i^*\| / \det(\tilde{\mathbf{B}})$. Using the

ζ-reducedness of the projected lattice for the other terms as in Theorem 3.1, we obtain

$$|M| \leq \frac{\prod_{i \notin I} \|\tilde{\mathbf{b}}_i^*\|}{\det(\tilde{\mathbf{B}})} \prod_{i \notin I} \frac{3\zeta(i) \cdot \det(\tilde{\mathbf{B}})^{1/(n-j)}}{\|\tilde{\mathbf{b}}_i^*\|} = 3^{n-j} \prod_{i \notin I} \zeta(i). \qquad \square$$

Note that the bound given in Theorem 3.2 is independent of $\mathbf{W}$. Intuitively, this is explained by the following observation: while a large value $\mathbf{W}_{t_i,i}$ requires a large number of cosets to be enumerated for the respective dependent vector in the generating system, these cosets are much sparser than the original lattice.

ζ-Reducedness of Projected Lattices To analyze the worst-case runtime of the enumeration procedure in projected lattices, it remains to discuss the ζ-reducedness of projected bases. The following theorem states the conditions for the original basis under which the projected basis is reduced.

Theorem 3 *Let $\mathbf{B}$, $\mathbf{V}$, $\mathbf{W}$, and I as in Lemma 3.2. Additionally, let $\mathbf{V}$ generate the same lattice as the first j vectors of some HKZ reduced basis for $\mathcal{L}(\mathbf{B})$. For an arbitrary $k \in [n]$, if $\mathbf{B}$ is such that $\pi_k(\mathbf{B})$ is HKZ reduced and $\|\mathbf{b}_i^*\| \leq \tilde{\zeta}(i) \det(\mathbf{B})^{1/n}$ for some $\tilde{\zeta} : [k] \to \mathbb{R}_+$ and all $i \leq k$, then $\pi_{\mathbf{V}}(\mathbf{B})$ is $(\gamma_{n-j+1}^{j/2(n-j)} \cdot \zeta)$-reduced for $\zeta(i \leq k) = \tilde{\zeta}(i)$ and $\zeta(i > k) = \sqrt{n}$.*

For the proof of the theorem we need a lemma to bound the GSO vectors of the projected basis. Specifically, Lemma 3.3 states that the GSO vectors can only get shorter when projecting the basis.

Lemma 3 *Let $\mathbf{B} \in \mathbb{Z}^{m \times n}$ be an arbitrary matrix and let $\mathbf{v}$ be an arbitrary vector. Furthermore, let $\tilde{\mathbf{B}} = \pi_{\mathbf{v}}(\mathbf{B})$. Then $\|\mathbf{b}_i^*\| \geq \|\tilde{\mathbf{b}}_i^*\|$ for all i.*

Proof It is tempting to assume that $\tilde{\mathbf{b}}_i^* = \pi_{\mathbf{v}}(\mathbf{b}_i^*)$, from which the lemma would follow trivially. However, a simple counter example shows that this is not true:

$$\mathbf{B} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 2 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

and $\mathbf{v} = \mathbf{b}_3$. Then it can be easily verified that $(1, 0, 0)^T = \tilde{\mathbf{b}}_2^* \neq \pi_{\mathbf{v}}(\mathbf{b}_2^*) = (1, 1/2, 0)^T$.

The lemma can still easily be seen to be true by considering the length of $\mathbf{b}_i^*$ as the shortest distance between the endpoint of $\mathbf{b}_i$ and the linear subspace spanned by $[\mathbf{b}_1, \cdots, \mathbf{b}_{i-1}]$. Now consider the (unique) line segment of length $\|\mathbf{b}_i^*\|$ between the endpoint of $\mathbf{b}_i$ and $\text{span}(\mathbf{b}_1, \cdots, \mathbf{b}_{i-1})$. Under orthogonal projection this line segment has length exactly $\|\pi_{\mathbf{v}}(\mathbf{b}_i^*)\|$ and is connecting the endpoint of $\tilde{\mathbf{b}}_i$ and $\text{span}(\tilde{\mathbf{b}}_1, \cdots, \tilde{\mathbf{b}}_{i-1})$. Furthermore, we know that $\|\tilde{\mathbf{b}}_i^*\|$ is the shortest distance between the endpoint of $\tilde{\mathbf{b}}_i$ and $\text{span}(\tilde{\mathbf{b}}_1, \cdots, \tilde{\mathbf{b}}_{i-1})$ and so obviously $\|\tilde{\mathbf{b}}_i^*\| \leq \|\pi_{\mathbf{v}}(\mathbf{b}_i^*)\| \leq \|\mathbf{b}_i^*\|$. $\square$

Now we are ready to prove Theorem 3.3.

Proof [of Theorem 3.3] First, let $\mathbf{B}$ be a generating system with $\|\mathbf{b}_i^*\| \leq \tilde{\zeta}(i) \det(\mathbf{B})^{1/n}$ for $i \leq k$ and some $\tilde{\zeta}$, and $\pi_k(\mathbf{B})$ is HKZ reduced. Let $\mathbf{v} = \sum_{i=1}^{t} x_i \mathbf{b}_i$ with $x_t \neq 0$ and $\|\mathbf{v}\| = \lambda_1(\mathbf{B})$ for some

$t \leq n$. We now discuss the projected generating system $\tilde{\mathbf{B}} = \pi_{\mathbf{v}}(\mathbf{B})$. We first obtain a lower bound for $\det(\tilde{\mathbf{B}})$:

$$\det(\mathbf{B})^{1/n} / \det(\tilde{\mathbf{B}})^{1/(n-1)} = \det \mathbf{B}^{1/n} \cdot \left(\frac{\|\mathbf{v}\|}{\det(\mathbf{B})}\right)^{1/(n-1)} = \left(\frac{\|\mathbf{v}\|}{\det(\mathbf{B})^{(1/n)}}\right)^{1/(n-1)} \leq \sqrt{\gamma_n^{1/(n-1)}}$$

For $i \leq k$ we know from Lemma 3.3 that

$$\|\tilde{\mathbf{b}}_i^*\| \leq \|\mathbf{b}_i^*\| \leq \tilde{\zeta}(i) \det(\mathbf{B})^{1/n} \leq \sqrt{\gamma_n^{1/(n-1)}} \tilde{\zeta}(i) \det(\tilde{\mathbf{B}})^{1/(n-1)}$$

Furthermore, for $k < i \leq t$ we know from HKZ reducedness of $\pi_k(\mathbf{B})$ and $i \leq t$ that $\|\mathbf{b}_i^*\| \leq \lambda_1(\mathbf{B}) \leq \sqrt{n} \det(\mathbf{B})^{1/n}$, and so

$$\|\tilde{\mathbf{b}}_i^*\| \leq \|\mathbf{b}_i^*\| \leq \sqrt{n} \det(\mathbf{B})^{1/n} \leq \sqrt{\gamma_n^{1/(n-1)}} \sqrt{n} \det(\tilde{\mathbf{B}})^{1/(n-1)}$$

Finally, it is easy to see that $\pi_t(\tilde{\mathbf{B}}) = \pi_t(\mathbf{B})$ is HKZ reduced. This proves the theorem for $j = 1$. Let $\zeta(i \leq k) = \tilde{\zeta}(i)$ and $\zeta(i > k) = \sqrt{n}$ as in the theorem. Applying the above step j times results in a projected generating system $\tilde{\mathbf{B}}$, such that

$$\|\tilde{\mathbf{b}}_i^*\| \leq \prod_{l=1}^{j} \left(\gamma_{n-l+1}^{1/2(n-l)}\right) \zeta(i) \det(\tilde{\mathbf{B}})^{1/(n-j)} \leq \gamma_{n-j+1}^{j/2(n-j)} \zeta(i) \det(\tilde{\mathbf{B}})^{1/(n-j)}$$

for $i \leq \max(I)$, where the last inequality follows from Mordell's inequality. Since $\pi_{\max(I)}(\tilde{\mathbf{B}}) = \pi_{\max(I)}(\mathbf{B})$ is still HKZ reduced and thus $(\gamma_{n-j+1}^{j/2(n-j)} \cdot \zeta)$-reduced (because $\gamma_{n-j+1}^{j/2(n-j)} > 1$), this proves the theorem. □

We remark that Theorem 3.3 uses a stronger assumption on the initial basis $\mathbf{B}$ than mere ζ-reduction for a certain function. It requires for the first k vectors to have an absolute bound.

3.3 The New Enumeration Algorithm

In this section we present our new algorithm. The design is mainly driven by the results in the previous sections. We introduce two parameters: $\kappa(n)$ allows to adjust the preprocessing, and $\eta(n)$ allows to adjust the number of vectors to enumerate before making a recursive call to achieve full HKZ reduction. The new algorithm is described in Algorithm 3.2. As most other lattice algorithms, it makes extensive use of orthogonal projection. As a notational convention, for any orthogonal projection π and algorithm A, we write $\pi^{-1}(A(\pi(\mathbf{B})))$ to describe the application of algorithm A to the projected lattice $\pi(\mathbf{B})$, while performing the same (linear) operations on the original lattice basis $\mathbf{B}$. More formally, if A is an algorithm that on input a lattice basis $\mathbf{B}$ outputs one or more lattice vectors, the modified algorithm $\pi^{-1}(A(\pi(\mathbf{B})))$ applies A on the projected lattice $\pi(\mathbf{B})$, expresses the result of running A as a linear combination $\pi(\mathbf{B})\mathbf{x}$ of the basis vectors, and outputs the same linear combination of the original basis $\mathbf{Bx}$. In practice, this is most commonly implemented by manipulating linear combinations of the original basis vectors $\mathbf{B}$ during the execution of A on $\pi(\mathbf{B})$.

We remark that the algorithm requires the basis $\mathbf{B}$ to be integral, but Line `5` and `11` contain recursive calls on projected bases, which might not be integral. However, it is folklore that integrality can always be achieved for projections of integral bases by scaling. The enumeration procedure called by the algorithm in line `4` and `10` is the one described in Section 3.2 (cf. Algorithm 3.1). As the bound for enumeration it uses the Minkowski bound of the lattice.

Lemma 4 *Algorithm 3.2 is correct and terminates in* $\tilde{O}(2^{n\kappa(n)/4}n^{(n-\kappa(n))/2})$ *steps.*

Proof Correctness will follow inductively if the enumeration calls in Line `4` and `10` behave correctly, i.e. indeed return the shortest vector in the respective lattice. The correctness of the call in Line `4` follows directly from Lemma 3.1. Similarly, correctness of the call in Line `10` follows from Lemma 3.2.

Now we turn to the complexity. We note that all steps in the algorithm are polynomial, with the exception of the enumerations in Line `4` and `10` (and thus the recursive calls). The runtime of the first call to the enumeration procedure can easily be seen to be $2^{O(\kappa(n)^2/4)} = 2^{O(n\kappa(n)/4)}$ since the basis is LLL reduced.

In order to analyze the complexity of the enumeration calls in Line `10`, we use Theorem 3.3 to derive the ζ-reducedness of $\pi_{\mathbf{V}}(\mathbf{B})$. It is plain to see that after the recursive call in Line `6`, $\mathbf{B}$ meets the condition $\|\mathbf{b}_i^*\| \leq \zeta(i)\det(\mathbf{B})^{1/n}$ for all $i \leq \kappa(n)$ and $\pi_{\kappa(n)}(\mathbf{B})$ is HKZ reduced. Additionally, we note that $\mathbf{V}$ always consists of the first vectors of an HKZ reduced basis for $\mathcal{L}(\mathbf{B})$. This means that we can apply Theorem 3.3 to see that in the j-th iteration of the loop $\pi_{\mathbf{V}}(\mathbf{B})$ is ζ-reduced with $\zeta(i \leq \kappa(n)) = \gamma_{n-j+1}^{j/2(n-j)} \cdot 2^{\frac{n-1}{4}}$ and $\zeta(i > \kappa(n)) = \gamma_{n-j+1}^{j/2(n-j)} \cdot \sqrt{n}$. Plugging this into Theorem 3.2, we obtain an upper bound on the runtime of each enumeration of

$$3^{n-j}\gamma_{n-j+1}^{j/2}2^{n\kappa(n)/4}n^{(n-\kappa(n)-j)/2} = 3^n 2^{n\kappa(n)/4}n^{(n-\kappa(n))/2}\left(\frac{\sqrt{\gamma_{n-j+1}}}{3\sqrt{n}}\right)^j = 2^{O(n)}2^{n\kappa(n)/4}n^{(n-\kappa(n))/2} \tag{3}$$

The number of recursive calls is bounded by $2^{O(n)}$ for any κ and η. □

Traditionally, one would now proceed with bounding the bit length of the numbers involved in the algorithm by showing that they are polynomial in the dimension and the maximum bit length of the input numbers. However, in the context of enumeration algorithms this kind of proof is considered classical by now and carries to our algorithm in a straight-forward fashion .

In Appendix A we prove a tighter bound for the algorithm in the typical case, where the norms of the GSO vectors are non-increasing, which achieves the best currently known worst-case bound of

$\tilde{O}(n^{n/2e})$ obtained in [13].

Algorithm 2 Our HKZ Reduction Algorithm

`procedure` reduce ($\mathbf{B}$, κ, η)
Input: A lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, two functions $\kappa, \eta : [n] \to [n]$
Output: An HKZ reduced basis of $\mathcal{L}(\mathbf{B})$

$\mathbf{B} \leftarrow \text{LLL}(\mathbf{B})$
$\Delta \leftarrow \det(\mathbf{B})$, $k \leftarrow \kappa(n)$
`if` $\exists i \leq k : \|b_i^*\| > 2^{\frac{n-1}{2}} \Delta^{\frac{1}{n}}$ `then`
  $\mathbf{v} \leftarrow \text{enum}(\mathbf{B}_{[1,i-1]})$
  `return` $[\mathbf{v} | \pi_{\mathbf{v}}^{-1}(\text{reduce}(\pi_{\mathbf{v}}(\mathbf{B}), \kappa, \eta))]$
$\mathbf{B} \leftarrow [\mathbf{B}_{[1,k]} | \pi_k^{-1}(\text{reduce}(\pi_k(\mathbf{B}), \kappa, \eta))]$
$h \leftarrow \eta(n)$
$\mathbf{V} = [\,]$
`for` $j \in [1, \cdots, \min(h, n)]$
  $\mathbf{V} \leftarrow [\mathbf{V} | \pi_{\mathbf{V}}^{-1}(\text{enum}(\pi_{\mathbf{V}}\mathbf{B}))]$
`return` $[\mathbf{V} | \pi_{\mathbf{V}}^{-1}(\text{reduce}(\pi_{\mathbf{V}}(\mathbf{B})), \kappa, \eta)]$

Algorithm 3 Variant of our HKZ Reduction Algorithm

`procedure` reduce ($\mathbf{B}$, κ)
Input: A lattice basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, a function $\kappa : [n] \to [n]$
Output: An HKZ reduced basis of $\mathcal{L}(\mathbf{B})$

$\mathbf{B} \leftarrow \text{LLL}(\mathbf{B})$
`do`
  $\mathbf{B} \leftarrow [\mathbf{B}_{[1,k]} | \pi_k^{-1}(\text{reduce}(\pi_k(\mathbf{B}), \kappa))]$
  $\mathbf{V} \leftarrow \text{reduce}(\mathbf{B}_{[1,k-1]}, \kappa)$
  `if` $\|\mathbf{v}_1\| \leq \|\mathbf{b}_k^*\|$
    `return` $[\mathbf{v}_1 | \pi_{\mathbf{v}_1}^{-1}(\text{reduce}(\pi_{\mathbf{v}_1}(\mathbf{B}), \kappa))]$
  $\mathbf{B} \leftarrow [\text{dualHKZ}(\mathbf{B}_{[1,k]}) | \mathbf{B}_{[k+1,n]}]$
`while` *change occurred*
$\mathbf{v} \leftarrow \text{enum}(\mathbf{B})$
`return` $[\mathbf{v} | \pi_{\mathbf{v}}^{-1}(\text{reduce}(\pi_{\mathbf{v}}(\mathbf{B}), \kappa))]$

Parameter Selection The algorithm allows two degrees of freedom, κ and η. From Lemma 3.4 it follows that all values up to $\kappa(n) = O(\log n)$ exhibit a worst-case time complexity of $n^{O(n)}$. From a theoretical point of view, $\kappa(n) = 1$ seems best, but in practice larger values might be better to reduce the number of recursive calls, as we will see in the next section. The parameter η is a little more tricky since it does not have an impact on the dominating factors of the worst-case time complexity. From a theoretical perspective, there are two valid arguments leading to conflicting choices of η. On the one hand, setting $\eta(n) = \Theta(n)$ reduces the number of recursive calls to a minimum. As the expensive recursive calls are the main drawback of Kannan's algorithm, this seems to be a good choice. Furthermore, in Equation 3.3 we see that the enumeration in the $(j+1)$-th iteration is easier than the first enumeration by a factor of $(\sqrt{\gamma_{n-j+1}}/3\sqrt{n})^j < 1$. In contrast, after the shortest vector has been found, one can enumerate again to find the second vector of the HKZ reduced basis, which according to Theorem 3.4 will take again $n^{O(n)}$. However, a recursive call in dimension $n-1$ has complexity of only $(n-1)^{O(n-1)}$. Peeking ahead, we note that our experiments indicate that this strategy is superior to the former at least up to moderate dimensions.

3.4 A Variant Based on Dual HKZ Reduction

As mentioned, in [13] the authors were able to improve the analysis of Kannan's algorithm to obtain a worst case bound on the asymptotic running time of $\tilde{O}(n^{n/2e})$ instead of Helfrich's $\tilde{O}(n^{n/2})$ [14]. It is unclear if and how the techniques from [13] can be applied to our algorithm to achieve a similar bound. Here we present a variant of our algorithm, for which we can rigorously prove the same complexity bound as in [13]. In Section 4 we will present experimental results, which show that in practice this variant has efficiency comparable to the previous algorithm.

This variant, presented in Algorithm 3.3, maintains the idea of restricting recursive calls to dimension $n-k$, but potentially makes logarithmically (in n) many of them interleaved with calls

to a k dimensional dual HKZ reduction. The dual HKZ reduction can be realized by calling the algorithm on the dual basis.

We first bound the enumeration step and then the number of loop iterations during the preprocessing in Algorithm 3.3 .

Lemma 5 *Let $\mathbf{B} \in \mathbb{Z}^{m \times n}$ be such that $\mathbf{B}_{[1,k]}$ is dual HKZ reduced and $\pi_{k-1}(\mathbf{B})$ is HKZ reduced for some $k \in [n]$. The shortest vector vector in $\mathcal{L}(\mathbf{B})$ can be computed by enumeration while exploring at most $k^{O(n)} n^{(n-k)/2e+o(n)}$ nodes.*

Proof Let $r_k = \sqrt{k} \det(\mathcal{L}(\mathbf{B}_{[1,k]}))^{1/k}$. Note that by Minkowski's theorem $r_k \geq \lambda_1(\mathcal{L}(\mathbf{B}_{[1,\kappa(n)]})) \geq \lambda_1(\mathcal{L}(\mathbf{B}))$, so $\mathcal{L}(\mathbf{B})$ contains a vector of length at most r_k. Furthermore, note that the shortest vector can be found by enumerating all vectors of length r_k in $\pi_k(\mathcal{L}(\mathbf{B}))$, followed by a CVP computation for each of them. We first bound the number of nodes explored during the first step. Note that due to the dual HKZ reducedness of $\mathbf{B}_{[1,k]}$ we have $1/\|\mathbf{b}_k^*\| \leq \sqrt{k}/\det(\mathcal{L}(\mathbf{B}_{[1,k]}))^{1/k}$, and so $r_k \leq k\|\mathbf{b}_k^*\|$. By Theorem 3 in [13], the number of nodes explored during the first step can be bounded by $k^{n-k} n^{(n-k)/2e+o(n)}$. For each vector found, the CVP problem can be solved exploring at most $k! = k^{O(k)}$ nodes due to the dual HKZ reducedness of $\mathbf{B}_{[1,k]}$ [4]. Multiplying the two bounds gives the result. □

Lemma 6 *Let $\mathbf{B} \in \mathbb{Z}^{m \times n}$ be LLL reduced. Then the number of iterations between Line 3 and 7 is at most logarithmic in $k \cdot n$ with base $\alpha = k/(k-1)$.*

Proof The proof is a generalization of Helfrich's proof of the number of loop iterations performed by Kannan's algorithm ([14], Lemma 3.3). Let $\lambda_1 = \lambda_1(\mathcal{L}(\mathbf{B}))$ and $\bar{\mathbf{B}}$ be such that it solves the densest $(k-1)$-sublattice problem and $\bar{\mathbf{B}}_{[1,k-1]}$ is LLL reduced. We need the following facts:

Fact 1 *If $\mathbf{B} \in \mathbb{Z}^{m \times k}$ is dual HKZ reduced, then* $\det(\mathcal{L}(\mathbf{B}_{[1,k-1]})) \leq \sqrt{\gamma_k} \det(\mathcal{L}(\mathbf{B}))^{(k-1)/k}$.

Proof Can easily be shown by applying Minkowski's theorem to the dual of $\mathbf{B}$. □

Fact 2 *When executing Line 7 of Algorithm 3.3, $\|\mathbf{b}_k^*\| \leq \lambda_1$ holds.*

Proof Let $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ with $\|\mathbf{v}\| = \lambda_1$. If $\pi_{k-1}(\mathbf{v}) > 0$, the fact follows from the HKZ reduction step. Otherwise, $\mathbf{v} \in \mathcal{L}(\mathbf{B}_{[1,k-1]})$ and the fact easily follows from the check in Line 5. □

Fact 3 *If $\mathbf{B}$ is LLL reduced, then* $\det(\mathcal{L}(\mathbf{B}_{[1,k-1]}))/\det(\mathcal{L}(\bar{\mathbf{B}}_{[1,k-1]})) \leq 2^{O(kn)}$.

Proof W.l.o.g. we can assume that $\|\mathbf{b}_i^*\| \leq 2^{(n-i)/2}\|\mathbf{b}_1\| \leq 2^n \lambda_1$, because otherwise $\lambda_1(\pi_{i-1}(\mathcal{L}(\mathbf{B}))) > \|\mathbf{b}_1\|$ (for brevity we ignored an explicit check in the presentation of Algorithm 3.3). It follows that $\det(\mathcal{L}(\mathbf{B}_{[1,k-1]})) \leq 2^{kn}\lambda_1^{k-1}$. From the properties of LLL reduction we get $\lambda_1 \leq \bar{\mathbf{b}}_1 \leq 2^{(k-2)/4}\det(\mathcal{L}(\bar{\mathbf{B}}_{[1,k-1]}))^{1/(k-1)}$ and so $\det(\mathcal{L}(\bar{\mathbf{B}}_{[1,k-1]})) \geq \lambda_1^{k-1}/2^{(kn)/4}$. Putting the two inequalities together proves the fact. □

We define $r(\mathbf{B}) = \det(\mathbf{B}_{[1,k-1]})/\det(\bar{\mathbf{B}}_{[1,k-1]})$. Note that the HKZ reduction step in Line 3 does not change the value of $r(\mathbf{B})$. Now let $\mathbf{B}$ be the matrix during some iteration of the loop before the dual HKZ reduction step (Line 7) and $\mathbf{B}'$ its result. Then we have

$$\begin{aligned}
r(\mathbf{B}') &= \frac{\det(\mathbf{B}'_{[1,k-1]})}{\det(\bar{\mathbf{B}}_{[1,k-1]})} \\
&\leq \sqrt{\gamma_k}\frac{\det(\mathbf{B}'_{[1,k]})^{(k-1)/k}}{\det(\bar{\mathbf{B}}_{[1,k-1]})} && \textit{Fact 3.1} \\
&\leq \sqrt{\gamma_k}\frac{\det(\mathbf{B}_{[1,k]})^{(k-1)/k}}{\det(\bar{\mathbf{B}}_{[1,k-1]})} \\
&\leq \sqrt{\gamma_k}r(\mathbf{B})^{(k-1)/k}\frac{\|\mathbf{b}_k^*\|^{(k-1)/k}}{\det(\bar{\mathbf{B}}_{[1,k-1]})^{1/k}} \\
&\leq \sqrt{\gamma_k}r(\mathbf{B})^{(k-1)/k}\frac{\lambda_1^{(k-1)/k}}{\det(\bar{\mathbf{B}}_{[1,k-1]})^{1/k}} && \textit{Fact 3.2} \\
&\leq \sqrt{\gamma_k}r(\mathbf{B})^{(k-1)/k}\frac{\lambda_1(\mathcal{L}(\bar{\mathbf{B}}))^{(k-1)/k}}{\det(\bar{\mathbf{B}}_{[1,k-1]})^{1/k}} \\
&\leq \sqrt{\gamma_k}r(\mathbf{B})^{(k-1)/k}\sqrt{\gamma_{k-1}}^{(k-1)/k} \\
&\leq \gamma_k r(\mathbf{B})^{1/\alpha}
\end{aligned}$$

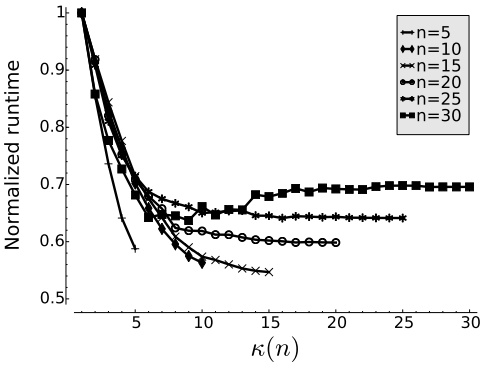
Now denote the value of $r(\mathbf{B})$ after the i-th iteration by r_i. We have just shown that $r_{i+1} \leq \gamma_k r_i^{1/\alpha}$ and it follows that $r_i \leq \gamma_k^i r_1^{1/\alpha^i} \leq \gamma_k^i 2^{O(kn)/\alpha^i}$ by Fact 3.3. So after $i = \log_\alpha O(kn)$ iterations $r_i \leq 2 \cdot O(kn)^{\log_\alpha \gamma_k}$. Introducing a slack $\delta > 1$ for the dual HKZ reduction step (which does not affect the bound given in Lemma 3.5), there are at most $\log_\alpha(\gamma_k)\log_\delta(O(kn))$ more iterations. □

4 Experimental Results

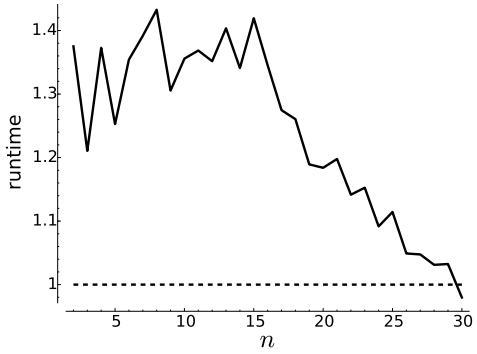
In this section we present our initial experimental evaluation of our new enumeration algorithm, and compare its practical performance to the algorithms of Fincke and Pohst [8] and lightweight Kannan (cf. Section 3). Our implementation of the basic enumeration procedure closely follows the algorithm of Schnorr and Euchner [29], and includes the following standard optimizations: (1) we update the search radius dynamically each time a shorter vector is found during the enumeration; (2) we enumerate the values on each level in increasing order of the distance of the resulting lattice point to the target; (3) we use the symmetry of the lattice to reduce the search space by a factor 2 when solving SVP; and (4) we preprocess the lattice using the LLL algorithm with parameter $\delta = 0.99$, rather than the $\delta = 3/4$ factor assumed in the theoretical analysis. We also consider preprocessing with the BKZ block basis reduction algorithm with various block sizes. We remark that the asymptotic analysis presented in Section 3 applies also to this optimized version of the algorithm, but does not lead to substantially better theoretical bounds. However, these optimizations result in better practical performance, allowing us to collect more experimental data

in a given amount of time. Of course, for fairness, the same optimizations have been applied also to our implementation of the algorithms of Fincke and Pohst and Kannan we compare against. In order to achieve flexibility in our experimental framework, and present the results in a less technology-dependent way, we chose not to perform any code-level optimization, and carried out our experiments with a fast prototyped implementation of the algorithm, whose performance is measured by counting the number of nodes explored during all enumerations. The same approach is commonly used in the analysis of enumeration algorithms [13, 14, 31].

Experimental results reported in this section are averaged over 20 random lattices, where we use the same lattices for each parameter set. We use the same distribution on *random lattices* as was used in [13], i.e. subset sum lattices with numbers of bit length $100 \cdot n$, where n is the lattice dimension, but we expect the results to hold for a wide range of probability distributions. To circumvent numerical issues, we ran `NTL`'s floating point version of LLL with $\delta = 0.5$ on the lattices before feeding them to our algorithm. We remark that our experiments are limited to dimension 50, which is much smaller than current records for SVP challenges[3]. This is because running full enumeration without further heuristic improvements is very costly. It was already noted in [11] that full enumeration beyond dimension 70 is problematic in practice and results of experiments with traditional Fincke-Pohst in [13] did not exceed dimension 52. Without any further low level optimizations it turned out that $n = 50$ was the largest dimension for which we could obtain statistically meaningful results within the available time frame. We will present estimates for larger dimension based on extrapolation of our experimental results.



(a) Normalized average runtime of the algorithm depending on preprocessing overhead ($\kappa(n)$) in several lattice dimensions n

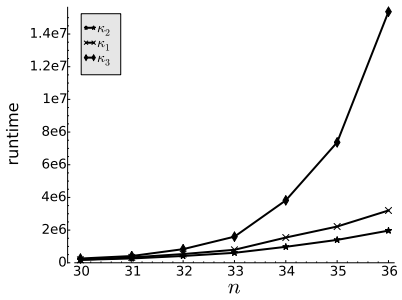


(b) Ratio of the average runtime of our parameter prediction ($\kappa(n) = \log(n)$) to Fincke-Pohst ($\kappa(n) = n$)

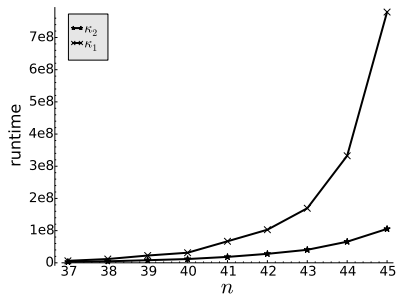
Figure 1: Runtime of the algorithm for $n = \dim(\Lambda) \leq 30$

Parameter Optimization for Low Dimensions We recall that our algorithm is parametrized by two functions κ, η that are used to pick the values of $k = \kappa(n)$ and $h = \eta(n)$. The first question we set to answer is what values of k, h lead to the best performance in practice. To this end, we performed an extensive set of experiments in low dimension (up to $n \leq 30$) for all possible values of $1 \leq k, h \leq n$, using dynamic programming to optimize the functions κ and η: for increasing

[3] `http://www.latticechallenge.org/svp-challenge`

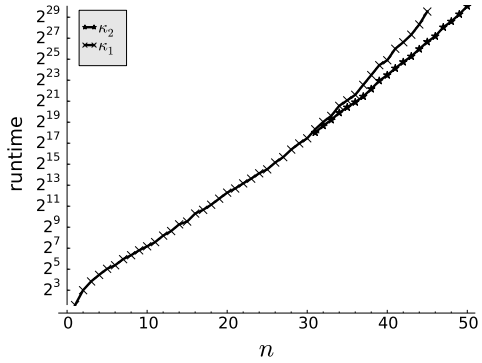


(a) κ_1, κ_2, and κ_3 for $30 \leq n \leq 36$

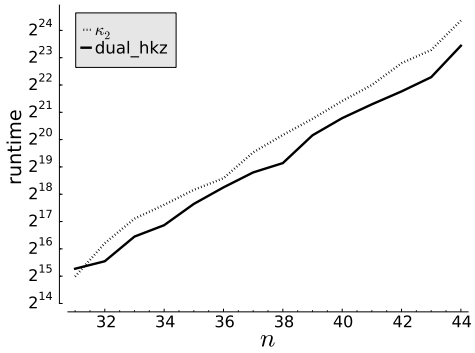


(b) κ_1 and κ_2 for $37 \leq n \leq 45$

Figure 2: Average running times of Fincke-Pohst(κ_1), our parameter selection(κ_2), and lightweight Kannan(κ_3) for $30 \leq n = \dim(\Lambda) \leq 45$

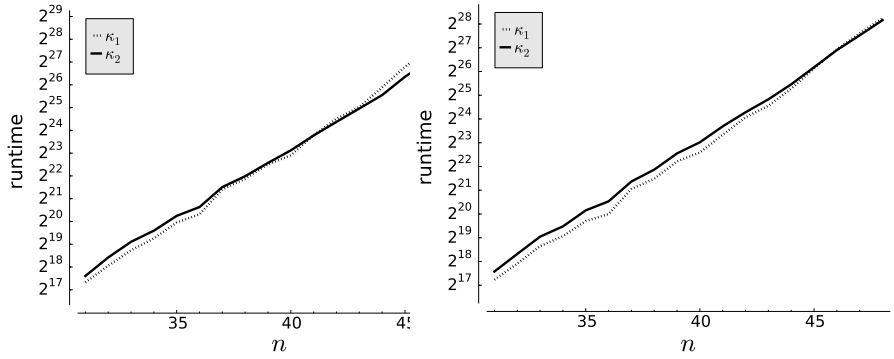


(a) Runtime of Fincke-Pohst(κ_1) and our parameter selection (κ_2) in logarithmic scale for $n = \dim(\Lambda) \leq 45$



(b) Runtime of top level enumeration of our algorithm (κ_2) and the variant based on dual HKZ reduction in logarithmic scale for $30 < n = \dim(\Lambda) \leq 45$

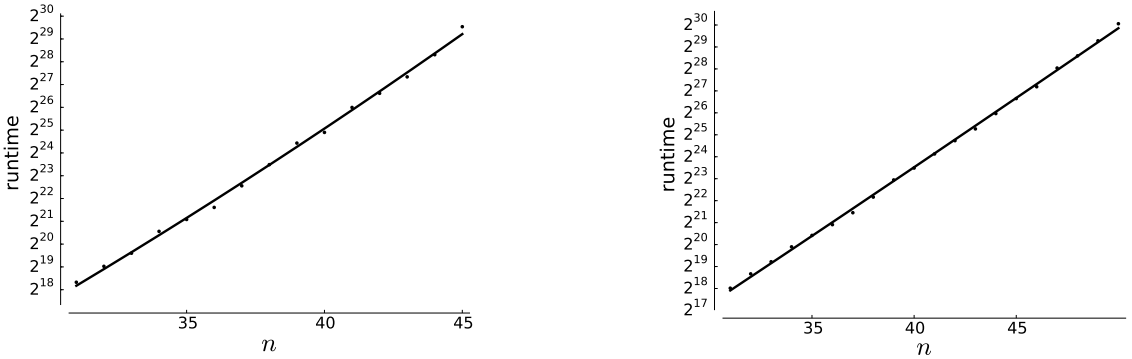
Figure 3: Comparison of the runtimes of our algorithm and other algorithms



(a) BKZ-10 (b) BKZ-20 (c) BKZ-30

Figure 4: Runtime of Fincke-Pohst (κ_1) and our algorithm (κ_2) with BKZ preprocessing (logarithmic scale)

dimension n, we explore the entire parameter space $(\kappa(n), \eta(n)) \in [n]^2$, using previously obtained values for κ and η for the recursive calls.



(a) Fincke-Pohst ($f_1(n) = 2^{a_1 n^2 + a_2 n}$)

(b) Our algorithm ($f_2(n) = n^{b_1 n} \cdot 2^{b_2 n}$)

Figure 5: Experimental data and fitted curves (logarithmic scale). Parameters of f_1 and f_2 as in Table 3 for LLL preprocessing.

For η, we observed that $\eta(n) = 1$ (i.e. projecting and recursing immediately after one enumeration) is always optimal or at least close to optimal, independently of the value chosen for $\kappa(n)$. So, in the rest of the paper we always assume $h = 1$. For κ, the results of our experiments are shown in Figure 1a, which depicts the normalized runtime of the algorithm as a function of $k = \kappa(n)$ for several lattice dimensions n. We see that for small n the graphs are monotonically decreasing, supporting previous claims that Kannan's algorithm (corresponding to $\kappa(n) = 1$) is not competitive in practice, and the common practice of running the (asymptotically inferior) Fincke-Pohst algorithm (corresponding to $\kappa(n) = n$). However, it is also evident that as n increases, most of the improvement over Kannan's algorithm is achieved by increasing k from 1 to a value only slightly larger than 1, after which the curves become rather flat. Moreover, for $n = 30$ we see that the parameter $\kappa(n) = 5 = \lceil \log(n) \rceil$ already gives slightly better performance than Fincke-Pohst, and increasing k beyond that value makes the algorithm slower. This is supported by Figure 1b, which shows the ratio between the running times of the algorithm (as a function of the lattice dimension n), when $\kappa(n) = \log(n)$ or $\kappa(n) = n$. The figure shows that Fincke-Pohst is slightly better than setting $k = \log(n)$ in low dimension $n < 30$, but $n = 30$ is the cross-over point, where using $\kappa(n) = \log(n)$ becomes faster.

Results in Moderate Dimensions For dimensions $n > 30$ the dynamic programming approach to optimize k, h is rather slow. So in our preliminary experimentation we focused on the most promising candidate values for the parameters, as predicted by the analytical arguments described in Section 3.3. Specifically, we compare the performance of the algorithm when $k = \log n$ to two standard benchmarks corresponding to the Fincke-Pohst ($k = n$) and the lightweight version of Kannan's algorithm ($k = 1$). More formally, our tests correspond to the execution of our algorithm with three possible functions κ_i defined as $\kappa_i(n \leq 30) = n$, and $\kappa_1(n > 30) = n$ (Fincke-Pohst), $\kappa_2(n > 30) = \log n$ (our candidate), $\kappa_3(n > 30) = 1$ (lightweight Kannan). Our results for $30 \leq n \leq 36$ are shown in Figure 2a. After obtaining these results we discarded the lightweight Kannan algorithm (κ_3), which is clearly inferior, and continued the experiments for the remaining two algorithms. Figure 2b shows the results and demonstrates that $n = 30$ is the cross-over point between our new algorithm and the traditional method of Fincke-Pohst. The new algorithm with $k = \log n$ clearly outperforms Fincke-Pohst for all $n \leq 45$ with exponentially growing speedup factor, which is already larger than 7 in dimension 45.

Figure 3a combines the results from Figure 2a and 2b for κ_1 and κ_2 on a logarithmic scale, where we extended the results for Fincke-Pohst (κ_1) with the results from the previous paragraph. Furthermore, the results are extended for κ_2 up to dimension $n = 50$, which we were able to obtain due to its superior performance. It shows that the running time of Fincke-Pohst starts to increase more rapidly beyond dimension 30 and there seems to be a qualitative difference between the running times of the algorithms, with the Fincke-Pohst algorithm exhibiting a clear superexponential slowdown, and the graph for our new algorithm much closer to a straight line (corresponding to an almost linear exponent in the running time, i.e. $2^{O(n \log n)}$).

Finally, we also ran experiments with our variant based on dual HKZ reduction, presented in Section 3.4, with the same parameter κ_2. Figure 3b compares the runtime of the top level enumeration of both algorithms. It shows that by spending more time during the preprocessing one can improve enumeration time by roughly a factor 2, but it does not seem like there is a qualitative difference in the runtime, as the lines are almost parallel. This reinforces our theoretical result in Appendix A that at least in the typical case our algorithm behaves optimally in the sense of [13].

Effect of BKZ It is common practice to use Fincke-Pohst in conjunction with BKZ preprocessing instead of mere LLL reduction. To evaluate the effect of BKZ on Fincke-Pohst and our algorithm, we ran similar experiments as described above. We used `NTL`'s BKZ routine with blocksize $\beta \in \{10, 20, 30\}$ as preprocessing and called Fincke-Pohst (κ_1) and our algorithm (κ_2) on the resulting basis. The results are shown in Figure 4. It is evident that the BKZ preprocessing increases the cross-over point from previously $n = 30$ to $n = 42, 46, 48$ for $\beta = 10, 20, 30$, respectively, but the improvement decays drastically with increasing blocksize. Further evidence based on the same data to support this claim is given in the next section.

Extrapolation and Comparison to other Algorithms Even though already stated, we emphasize that our experimental analysis of the algorithm is only preliminary as the tested dimensions are fairly low compared to what is possible with today's state of the art (see, e.g. the SVP challenge). Nonetheless, to get a sense of how well our algorithm performs in higher dimensions, we used standard statistical methods to fit curves to the data collected in our experiments. To determine the practical complexity of the enumeration on reduced bases, we selected a model based on our theoretical analysis, $f(n) = 2^{c_1 n^2} \cdot n^{c_2 n} \cdot 2^{c_3 n}$, and fitted it to the data collected only during the top level enumeration, i.e. ignoring pre- and postprocessing. The resulting constants for Fincke-Pohst and our algorithm are shown in Table 2. For Fincke-Pohst, the running time (in low dimension) is dominated by the single exponential component $2^{c_3 n}$, but there is also a quadratic term $2^{c_1 n^2}$ which becomes dominant in sufficiently high dimension. (Notice also the complete absence of a quasilinear exponent $2^{c_2 n \log n}$.) So, the runtime of Fincke-Pohst is clearly $2^{O(n^2)}$ also in practice when the dimension is sufficiently large. On the other hand, for our algorithm the dominating factor is the quasilinear exponent associated to c_2. Apart from the small constant c_1 in the case of our algorithm, which we attribute to noise, this is consistent with the theoretical analysis. For further exploration, we selected the model $f_1(n) = 2^{a_1 n^2 + a_2 n}$ for the Fincke-Pohst algorithm and $f_2(n) = n^{b_1 n} \cdot 2^{b_2 n}$ for our algorithm and fitted them to the entire runtimes, i.e. with pre- and postprocessing, including the variants using BKZ. The result is shown in Table 3 and visualized in Figure 5 (for LLL preprocessing). The table demonstrates again, that the improvement due to BKZ decreases with increasing blocksize.

Algorithm	c_1	c_2	c_3
Fincke-Pohst	0.0068	0.0000	0.3195
Ours	0.0009	0.0839	0.0508

Table 2: Parameters of model $2^{c_1 n^2} \cdot n^{c_2 n} \cdot 2^{c_3 n}$ after curve fitting to top level enumeration

	$f_1(n) = 2^{a_1 n^2 + a_2 n}$		$f_2(n) = n^{b_1 n} \cdot 2^{b_2 n}$	
Preprocessing	a_1	a_2	b_1	b_2
LLL	0.0045	0.4469	0.0280	0.4389
BKZ-10	0.0024	0.4830	0.0245	0.4504
BKZ-20	0.0019	0.4962	0.0237	0.4518
BKZ-30	0.0016	0.5028	0.0225	0.4574

Table 3: Parameters of models f_1 (for Fincke-Pohst) and f_2 (for our algorithm) after curve fitting

In [9] the authors claim that enumeration can be sped up heuristically by a factor of $2^{n/2}$ with extreme pruning. Recall that the basic idea of extreme pruning is to heavily cut down on the enumeration and compensate the sacrificed success probability by a large number of repetitions with randomized inputs. Fincke-Pohst is an ideal candidate for extreme pruning due to the cheap preprocessing, which has to be applied in each iteration. After introducing the corresponding speed-up into f_1 we can estimate the dimension n at which our algorithm becomes more efficient than Fincke-Pohst with extreme pruning by computing the cross-over point between (the modified) f_1 and f_2. This point is reached for $n = 155$, which seems already very close to practically tractable dimensions. It is not immediately clear if extreme pruning can be applied to our algorithm due to its heavier preprocessing. However, in the same work the authors of [9] show that a speed-up of $2^{n/4}$ can be achieved by non-extreme pruning, where the enumeration is pruned but only sacrificing very little success probability. This non-extreme form of pruning can readily be applied to our algorithm and we expect it to result in similar speed-ups. If this is indeed the case the cross-over point at which our algorithm becomes more efficient than Fincke-Pohst with extreme pruning would drop to $n = 95$ – well below the limit of today's tractability.

Applying the same approach to variants that use BKZ and extreme pruning is a little problematic, as extreme pruning is likely to suffer from heavy preprocessing. The precise impact of BKZ on the practical complexity of extreme pruning has to the best of our knowledge not been investigated in detail. However, the authors of [9] give a prediction for the number of nodes that have to be enumerated by extreme pruning in dimension $n = 110$ using BKZ-32 preprocessing, which is $2.5 \cdot 10^{13}$. Using our model, we expect the number of nodes to be enumerated by our algorithm with BKZ-30 and (non-extreme) pruning to be about $8.4 \cdot 10^{11}$. This already corresponds to a speed-up of about 30 even without considering the overhead incurred by the repeated application of BKZ-32 necessary for extreme pruning. We expect this speed-up factor to increase rapidly in larger dimensions due to the superior asymptotics of our algorithm.

We can use our model to estimate at which point the asymptotically superior sieving becomes more efficient. Even though sieving algorithms have the drawback of exponential memory requirements, we will ignore this fact and focus on the runtime. In [19] the running time of HashSieve, a very recent sieving variant and to the best of our knowledge the most efficient sieving algorithm in practice to date, was estimated to be $2^{0.45n-19}$ seconds in dimension n in practice. In order to compare our algorithm meaningfully to this estimate, we need to convert the number of nodes calculated by f_2 to computing time on a comparable computer that was used in [19]. For this we use the observation made in [9] that an efficient implementation of enumeration can process a node in approximately 200 clock cycles. Computing the intersection of f_2 with the estimate for sieving of $2^{0.45n-19}$ suggests that sieving is more efficient than our algorithm starting already in dimension

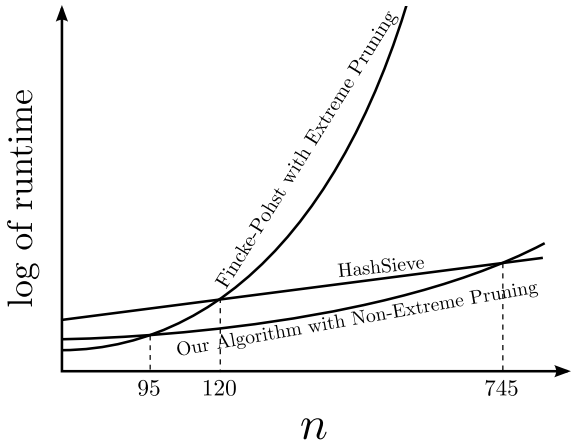


Figure 6: Diagram of estimated runtimes and cross-over points (not to scale). All Algorithms use only LLL preprocessing.

$n = 35$. However, if introducing the potential speed-up of $2^{n/4}$ to f_2 due to pruning, this cross-over point rises to $n = 745$. Using for f_2 the parameters resulting from the application of BKZ-30 to the basis before running our algorithm (cf. Table 3) and including the pruning heuristic, we expect our algorithm to be more efficient up to dimension $n = 1895$. The latter two values for n are both by far out of reach for today's state of the art. We remark that the estimate from [19] does not take block reduction as preprocessing into account. This is likely to improve HashSieve somewhat, but we do not believe that it will change the picture significantly.

Remark about Fincke-Pohst and Sieving Although not directly relevant to the algorithm proposed in this work, we can use the model for Fincke-Pohst and compare it to sieving. Without pruning, Fincke-Pohst is more efficient than sieving up to dimension $n = 31$, with non-extreme pruning up to $n = 71$ and with extreme pruning up to $n = 120$. This is consistent with results of the SVP challenge, which gives us confidence in our model. Furthermore, these numbers suggest that in the search for fast practical algorithms, in particular in dimensions relevant to currently unbroken challenges, one should focus on (non-extremely) pruned enumeration or sieving rather than extreme pruning.

An illustration of the cross-over points is given in Figure 6. We reiterate that this analysis is preliminary and more experimental evidence is necessary to support our hypotheses. For this a more efficient implementation is essential. To take a first step in this direction we modified `fpLLL` to use our preprocessing and are starting to run similar experiments with it. Preliminary results are shown in Figure[4] 7 and demonstrate that our algorithm can be run up to dimension almost 70 in reasonable time using one core on a standard PC (Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz).

5 Future Work

Much work remains to be done to evaluate the theoretical and practical impact of known heuristics (most notably, block reduction [28, 29, 10] and pruning [27, 30, 9]) to the new enumeration method.

[4]While the results for $n \leq 65$ are again averaged over 20 instances, the runtimes for larger dimensions only represent one data point due to time constraints.

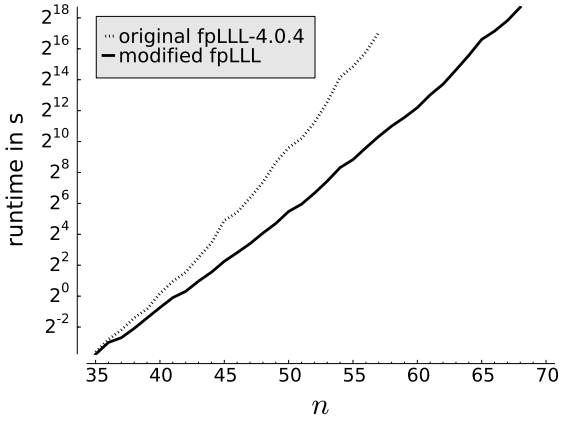


Figure 7: Comparison of the runtime of original `fpLLL` and our modified version

Still, we believe our new algorithm offers a better starting point for realistic experimentation and meaningful extrapolation, than previously known enumeration algorithms.

We conclude the paper by noting that the parameter setting $\kappa(n) = \log(n)$ and $\eta(n) = 1$ used in our experiments also suggests a possible variant that combines classic enumeration techniques with recent theoretical advances on CVP algorithms [23]. Notice that the last k levels of the enumeration tree with basis $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$ correspond precisely to a CVP computation on the lattice $\mathbf{B}_{[1,k]}$. So, we can replace the last k levels of the enumeration tree with any other CVP algorithm. We recall that [23] gives a CVP algorithm that after computing the Voronoi cell of the lattice (in time $2^{O(n)}$,) allows to quickly solve CVP. The algorithm is considered primarily of theoretical interest because storing the Voronoi cell takes exponential space. However, when applied to $\mathbf{B}_{[1,k]}$ for $k = O(\log n)$, both the time and space complexity $2^{O(k)} = n^{O(1)}$ of the algorithm of [23] is polynomial in the original lattice dimension n. Moreover, the Voronoi cell of $\mathbf{B}_{[1,k]}$ only needs to be computed once, and can then be reused for all CVP instances corresponding to the leaves of the truncated enumeration tree for $\pi_k(\mathbf{B})$. So, the algorithm can be usefully employed to possibly speed up the enumeration procedure, while keeping the overall space complexity of the algorithm polynomial in n. The high level structure of the hybrid algorithm would be the following: (1) apply LLL or other block basis reduction algorithm to $\mathbf{B}$, (2) recursively HKZ-reduce the projected lattice $\pi_k(\mathbf{B})$, (3) compute the Voronoi cell of $\mathbf{B}_{[1,k]}$, (4) find the shortest vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ by enumeration in $\pi_k(\mathbf{B})$ and using the Voronoi cell to solve CVP in $\mathbf{B}_{[1,k]}$, (5) recursively HKZ-reduce $\pi_{\mathbf{v}}(\mathbf{B})$. Exploring the theoretical and practical significance of this hybrid algorithm is left to future work.

References

[1] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, Aug. 2002.

[2] M. Ajtai. Generating hard instances of lattice problems. *Complexity of Computations and Proofs, Quaderni di Matematica*, 13:1–32, 2004. Preliminary version in STOC 1996.

[3] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of STOC*, pages 266–275. ACM, July 2001.

[4] J. Blömer. Closest vectors, successive minima, and dual hkz-bases of lattices. In *Proceedings of the 17th ICALP*, pages 248 – 259. Springer, 2000.

[5] J. Blömer and S. Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. *Theoretical Computer Science*, 410(18):1648–1665, Apr. 2009. Preliminary version in ICALP 2007.

[6] J. Buchmann and M. Pohst. Computing a lattice basis from a system of generating vectors. In J. H. Davenport, editor, *Proceedings of the european conference on computer algebra*, volume 378 of *Lecture Notes in Computer Science*, pages 54–63. Springer, June 1989.

[7] Y. Chen and P. Q. Nguyen. Bkz 2.0: Better lattice security estimates. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.

[8] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44:463–471, 1985.

[9] N. Gama, P. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In H. Gilbert, editor, *Advances in Cryptology EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 257–278. Springer Berlin / Heidelberg, 2010.

[10] N. Gama and P. Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In *Proceedings of STOC*, pages 207–216. ACM, May 2008.

[11] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *Proceedings of Eurocrypt*, volume 4965 of *LNCS*, pages 31–51. Springer, 2008.

[12] O. Goldreich, D. Micciancio, S. Safra, and J.-P. Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Information Processing Letters*, 71(2):55–61, 1999.

[13] G. Hanrot and D. Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In *Proceedings of Crypto*, volume 4622 of *LNCS*, pages 170–186. Springer, Aug. 2007.

[14] B. Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. *Theoretical Computer Science*, 41(2–3):125–139, Dec. 1985.

[15] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the fifteenth annual ACM symposium on theory of computing - STOC '83*, pages 193–206. ACM, Apr. 1983. Journal version in Math. of Operation Research 12(3):415-440, (1987).

[16] R. Kannan. *Annual Review of Computer Science*, volume 2, chapter Algorithmic Geometry of numbers, pages 231–267. Annual Review Inc., Palo Alto, California, 1987.

[17] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operation research*, 12(3):415–440, Aug. 1987. Prelim. version in STOC 1983.

[18] S. Khot. Hardness of approximating the shortest vector problem in lattices. *Journal of the ACM*, 52(5):789–808, Sept. 2005. Preliminary version in FOCS 2004.

[19] T. Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. Cryptology ePrint Archive, Report 2014/744, 2014. `http://eprint.iacr.org/`.

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

[21] D. Micciancio. Inapproximability of the shortest vector problem: Toward a deterministic reduction. *Theory of Computing*, 8(1):487–512, 2012.

[22] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measure. *SIAM Journal on Computing*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.

[23] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proceedings of STOC*, pages 351–358, 2010.

[24] D. Micciancio and P. Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *Proceedings of SODA*, pages 1468–1480. ACM/SIAM, Jan. 2010.

[25] D. Micciancio and B. Warinschi. A linear space algorithm for computing the Hermite Normal Form. In *Proceedings of ISSAC '96*, pages 231–236. ACM Press, July 2001.

[26] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of ACM*, 56(6):34, Sept. 2009. Preliminary version in STOC 2005.

[27] C. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1-3):181–199, 1994.

[28] C.-P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2–3):201–224, Aug. 1987.

[29] C.-P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1-3):181–199, Aug. 1994. Preliminary version in FCT 1991.

[30] C.-P. Schnorr and H. H. Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In *Proceedings of EUROCRYPT '95*, volume 921 of *LNCS*, pages 1–12. Springer, May 1995.

[31] J. van de Pol and N. P. Smart. Estimating key sizes for high dimensional lattice based systems. Cryptology ePrint Archive, Report 2013/630, 2013. `http://eprint.iacr.org/`.

A Better Asymptotics for Non-Increasing Bases

In this section we will show that we can proof a tighter bound on the enumeration after preprocessing if the GSO norms of the input basis are non-increasing, which is typically the case. What follows is an adaptation of the proof of Theorem 3 in [13], which can be found more explicitly in the extended version[5]. We show the result for $k = 1$, which yields the best asymptotic runtime in our previous analysis. The proof can easily be adapted to larger k and yields the expected runtime.

We use the same starting point as the authors of [13], where it is proved that the number of nodes processed during a SVP enumeration with arbitrary bound r on an arbitrary basis $\mathbf{B}$ can be bounded by

$$2^{O(n)} \max_{I \subset [n]} \left(\frac{r^{|I|}}{\sqrt{n}^{|I|} \prod_{i \in I} \|\mathbf{b}_i^*\|} \right)$$

up to polynomial factors. Since our enumeration uses the bound $r = \sqrt{n} \det(\mathbf{B})^{1/n}$, this is equivalent to

$$2^{O(n)} \prod_I \frac{\det(\mathbf{B})^{1/n}}{\|\mathbf{b}_i^*\|}$$

where $I = \{i : \|\mathbf{b}_i^*\| < \det(\mathbf{B})^{1/n}\}$. Note that in the case of non-increasing bases, I is of the form $I = [k, \ldots, n]$ for some $k \in [n]$. The following lemma shows the result:

Lemma 7 *Let $\mathbf{B} \in \mathbb{Z}^{m \times n}$ with $\|\mathbf{b}_1\| \leq 2^{(n-1)/4} \det(\mathbf{B})^{1/n}$, $\pi_1(\mathbf{B})$ be HKZ reduced, and $\|\mathbf{b}_1\| \geq \|\mathbf{b}_2^*\| \geq \cdots \geq \|\mathbf{b}_n^*\|$. Then for all $k \in [n]$*

$$\prod_{i=k}^{n} \frac{\det(\mathbf{B})^{1/n}}{\|\mathbf{b}_i^*\|} \leq 2^n n^{n/2e}.$$

Proof Let

$$\tilde{\Gamma}_n(k) = 2 \cdot \Gamma_{n-1}(k-1)$$

where $\Gamma_n(k) = \prod_{i=n-k}^{n-1} (\gamma_{i+1})^{1/2i}$ from Definition 2 of [13] and $\Gamma_n(0) = 1$. From Lemma 2 in [13] we immediately obtain $\tilde{\Gamma}_n(k) \leq 2\sqrt{n}^{\log\left(\frac{n}{n-k}\right)}$. Akin to the proof in the extended version of [13], we let $\pi_{[k,n]} = \prod_{i=k}^{n} \|\mathbf{b}_i^*\|^{1/(n-k+1)}$ and first proof the inequality

$$\pi_{[1,k]} \leq \tilde{\Gamma}_n(k)^{n/k} \pi_{[k+1,n]} \tag{4}$$

for all k by induction on k. For $k = 1$ the inequality follows from the assumption $\|\mathbf{b}_1\| \leq 2^{(n-1)/4} \det(\mathbf{B})^{1/n}$. The rest of the proof is identical to the one of Hanrot and Stehlé and shamelessly copied for completeness. Assume that the inequality holds for $k \geq 1$ and rewrite it as

$$\pi_{[1,k+1]}^{\frac{k+1}{k}} \cdot \|\mathbf{b}_{k+1}^*\|^{-\frac{1}{k}} \leq \tilde{\Gamma}_n(k)^{\frac{n}{k}} \cdot \pi_{[k+2,n]}^{\frac{n-k-1}{n-k}} \cdot \|\mathbf{b}_{k+1}^*\|^{\frac{1}{n-k}}$$

which is equivalent to

$$\pi_{[1,k+1]}^{\frac{k+1}{k}} \leq \tilde{\Gamma}_n(k)^{\frac{n}{k}} \cdot \pi_{[k+2,n]}^{\frac{n-k-1}{n-k}} \cdot \|\mathbf{b}_{k+1}^*\|^{\frac{n}{k(n-k)}}$$

[5] `http://perso.ens-lyon.fr/damien.stehle/KANNAN_EXTENDED.html`

From the HKZ reducedness assumption it follows that $\|\mathbf{b}_{k+1}^*\| \leq \sqrt{\gamma_{n-k}}^{\frac{n-k}{n-k-1}} \cdot \pi_{[k+2,n]}$, which gives

$$\pi_{[1,k+1]}^{\frac{k+1}{k}} \leq \tilde{\Gamma}_n(k)^{\frac{n}{k}} \sqrt{\gamma_{n-k}}^{\frac{n}{k(n-k-1)}} \cdot \pi_{[k+2,n]}^{\frac{k+1}{k}} = \tilde{\Gamma}_n(k+1)^{\frac{n}{k}} \cdot \pi_{[k+2,n]}^{\frac{k+1}{k}}$$

which yields the induction step after raising to the power $k/(k+1)$.

From Inequality A.1 we obtain the inequality

$$\pi_{[k+1,n]} \geq \frac{\det(\mathbf{B})^{1/n}}{\tilde{\Gamma}_n(k)} \tag{5}$$

by raising A.1 to the power of k/n, multiplying it with $\pi_{[k+1,n]}^{(n-k)/n}$, and using the identity $\det(\mathbf{B}) = \pi_{[1,k]}^k \cdot \pi_{[k+1,n]}^{n-k}$. From A.2 we get

$$\left(\frac{\det(\mathbf{B})^{1/n}}{\pi_{[k+1,n]}}\right)^{n-k} \leq \tilde{\Gamma}_n(k)^{n-k} \leq 2^{n-k}\sqrt{n}^{(n-k)\cdot\log\left(\frac{n}{n-k}\right)} \leq 2^n\sqrt{n}^{n/e}. \qquad \square$$