More Basis Reduction for Linear Codes: Backward Reduction, BKZ, Slide Reduction, and More

Surendra Ghentiyala ✉
Cornell University, Ithaca, NY, USA

Noah Stephens-Davidowitz ✉
Cornell University, Ithaca, NY, USA

Abstract

We expand on recent exciting work of Debris-Alazard, Ducas, and van Woerden [Transactions on Information Theory, 2022], which introduced the notion of *basis reduction for codes*, in analogy with the extremely successful paradigm of basis reduction for lattices. We generalize DDvW's LLL algorithm and size-reduction algorithm from codes over $\mathbb{F}_2$ to codes over $\mathbb{F}_q$, and we further develop the theory of proper bases. We then show how to instantiate for codes the BKZ and slide-reduction algorithms, which are the two most important generalizations of the LLL algorithm for lattices.

Perhaps most importantly, we show a new and very efficient basis-reduction algorithm for codes, called *full backward reduction*. This algorithm is quite specific to codes and seems to have no analogue in the lattice setting. We prove that this algorithm finds vectors as short as LLL does in the worst case (i.e., within the Griesmer bound) and does so in less time. We also provide both heuristic and empirical evidence that it outperforms LLL in practice, and we give a variant of the algorithm that provably outperforms LLL (in some sense) for random codes.

Finally, we explore the promise and limitations of basis reduction for codes. In particular, we show upper and lower bounds on how "good" of a basis a code can have, and we show two additional illustrative algorithms that demonstrate some of the promise and the limitations of basis reduction for codes.

2012 ACM Subject Classification Theory of computation → Error-correcting codes

Keywords and phrases Linear Codes, Basis Reduction

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2024.19

Category APPROX

Related Version *Full Version*: https://arxiv.org/abs/2408.08507 [21]

Funding *Surendra Ghentiyala*: This work is supported in part by the NSF under Grants Nos. CCF-2122230 and CCF-2312296, a Packard Foundation Fellowship, and a generous gift from Google.
Noah Stephens-Davidowitz: This work is supported in part by the NSF under Grants Nos. CCF-2122230 and CCF-2312296, a Packard Foundation Fellowship, and a generous gift from Google. Some of this work was completed while the author was visiting the National University of Singapore and the Centre for Quantum Technologies

1 Introduction

1.1 Codes and lattices

A *linear code* $\mathcal{C} \subseteq \mathbb{F}_q^n$ is a subspace of the vector space $\mathbb{F}_q^n$ over the finite field $\mathbb{F}_q$, i.e., it is the set of all $\mathbb{F}_q$-linear combinations of a linearly independent list of vectors $\boldsymbol{B} := (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k)$,

$$\mathcal{C} := \mathcal{C}(\boldsymbol{B}) := \{z_1\boldsymbol{b}_1 + \cdots + z_k\boldsymbol{b}_k \ : \ z_i \in \mathbb{F}_q\} \ .$$

We call $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k$ a *basis* for the code and k the *dimension* of the code. We are interested in the geometry of the code induced by the Hamming weight $|\boldsymbol{c}|$ for $\boldsymbol{c} \in \mathbb{F}_q^n$, which is simply the number of coordinates of $\boldsymbol{c}$ that are non-zero. For example, it is natural to ask about a

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2024).
Editors: Amit Kumar and Noga Ron-Zewi; Article No. 19; pp. 19:1–19:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

code's *minimum distance*, which is the minimal Hamming weight of a non-zero codeword, i.e.,

$$d_{\min}(\mathcal{C}) := \min_{\boldsymbol{c}\in\mathcal{C}_{\neq\boldsymbol{0}}} |\boldsymbol{c}| \; .$$

At a high level, there are two fundamental computational problems associated with linear codes. In the first, the goal is to find a short non-zero codeword – i.e., given a basis for a code $\mathcal{C}$, the goal is to find a non-zero codeword $\boldsymbol{c} \in \mathcal{C}_{\neq\boldsymbol{0}}$ with relatively small Hamming weight $|\boldsymbol{c}|$. In the second, the goal is to find a codeword that is close to some target vector $\boldsymbol{t} \in \mathbb{F}_q^n$ – i.e., given a basis for a code $\mathcal{C}$ and a target vector $\boldsymbol{t} \in \mathbb{F}_q^n$, the goal is to find a codeword $\boldsymbol{c} \in \mathcal{C}$ such that $|\boldsymbol{c} - \boldsymbol{t}|$ is relatively small. (Here, we are being deliberately vague about what we mean by "relatively small.")

We now describe another class of mathematical objects, which are also ubiquitous in computer science. Notice the striking similarity between the two descriptions.

A *lattice* $\mathcal{L} \subset \mathbb{Q}^n$ is the set of all *integer* linear combinations of linearly independent basis vectors $\boldsymbol{A} := (\boldsymbol{a}_1; \ldots ; \boldsymbol{a}_k) \in \mathbb{Q}^n$, i.e.,

$$\mathcal{L} := \mathcal{L}(\boldsymbol{A}) := \{z_1\boldsymbol{a}_1 + \cdots + z_k\boldsymbol{a}_k \; : \; z_i \in \mathbb{Z}\} \; .$$

We are interested in the geometry of the lattice induced by the Euclidean norm $\|\boldsymbol{a}\| := (a_1^2 + \cdots + a_n^2)^{1/2}$. In particular, it is natural to ask about a lattice's *minimum distance*, which is simply the minimal Euclidean norm of a non-zero lattice vector, i.e.,

$$\lambda_1(\mathcal{L}) := \min_{\boldsymbol{y}\in\mathcal{L}_{\neq\boldsymbol{0}}} \|\boldsymbol{y}\| \; .$$

At a high level, there are two fundamental computational problems associated with lattices. In the first, the goal is to find a short non-zero lattice vector – i.e., given a basis for a lattice $\mathcal{L}$, the goal is to find a non-zero lattice vector $\boldsymbol{y} \in \mathcal{L}_{\neq\boldsymbol{0}}$ with relatively small Euclidean norm $\|\boldsymbol{y}\|$. In the second, the goal is to find a lattice vector that is close to some target vector $\boldsymbol{t} \in \mathbb{Q}^n$ – i.e., given a basis for a lattice $\mathcal{L}$ and a target vector $\boldsymbol{t} \in \mathbb{Z}^n$, the goal is to find a lattice vector $\boldsymbol{y} \in \mathcal{L}$ such that $\|\boldsymbol{y} - \boldsymbol{t}\|$ is relatively small. (Again, we are being deliberately vague here.)

Clearly, there is a strong analogy between linear codes and lattices, where to move from codes to lattices, one more-or-less just replaces a finite field $\mathbb{F}_q$ with the integers $\mathbb{Z}$ and the Hamming weight $|\cdot|$ with the Euclidean norm $\|\cdot\|$. It is therefore no surprise that this analogy extends to many applications. For example, lattices and codes are both used for decoding noisy channels. They are both used for cryptography (see, e.g., [26, 3, 5, 23, 7, 30]; in fact, both are used specifically for *post-quantum* cryptography). And, many complexity-theoretic hardness results were proven simultaneously or nearly simultaneously for coding problems and for lattice problems, often with similar or even identical techniques.[1]

1.2 Basis reduction for lattices

However, the analogy between lattices and codes has been much less fruitful for algorithms. Of course, there are many algorithmic techniques for finding short or close codewords and many algorithmic techniques for finding short or close lattice vectors. But, in many parameter regimes of interest, the best algorithms for lattices are quite different from the best algorithms for codes.

[1] For example, similar results that came in separate works in the two settings include [15] and [14], [4] and [33], [27] and [19], [13, 2] and [32], etc. Works that simultaneously published the same results in the two settings include [9] and [18].

In the present work, we are interested in *basis reduction*, a ubiquitous algorithmic framework in the lattice literature. At a very high level, given a basis $\boldsymbol{A} := (\boldsymbol{a}_1; \ldots; \boldsymbol{a}_k)$ for a lattice $\mathcal{L}$, basis reduction algorithms work by attempting to find a "good" basis of $\mathcal{L}$ (and in particular, a basis whose first vector $\boldsymbol{a}_1$ is short) by repeatedly making "local changes" to the basis. Specifically, such algorithms manipulate the Gram-Schmidt vectors $\widetilde{\boldsymbol{a}}_1 := \boldsymbol{a}_1, \widetilde{\boldsymbol{a}}_2 := \pi^{\perp}_{\{\boldsymbol{a}_1\}}(\boldsymbol{a}_2), \ldots, \widetilde{\boldsymbol{a}}_k := \pi^{\perp}_{\{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{k-1}\}}(\boldsymbol{a}_k)$. Here, $\pi^{\perp}_{\{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{i-1}\}}$ represents orthogonal projection onto the subspace orthogonal to $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}$. Notice that we can view the Gram-Schmidt vector $\widetilde{\boldsymbol{a}}_i$ as a lattice vector in the lower-dimensional lattice generated by the *projected block* $\boldsymbol{A}_{[i,j]} := \pi^{\perp}_{\{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{i-1}\}}(\boldsymbol{a}_i; \ldots; \boldsymbol{a}_j)$. Basis reduction algorithms work by making many "local changes" to $\boldsymbol{A}$, i.e., changes to the block $\boldsymbol{A}_{[i,j]}$ that leave the lattice $\mathcal{L}(\boldsymbol{A}_{[i,j]})$ unchanged. The goal is to use such local changes to make earlier Gram-Schmidt vectors shorter. (In particular, $\widetilde{\boldsymbol{a}}_1 = \boldsymbol{a}_1$ is a non-zero lattice vector. So, if we can make the first Gram-Schmidt vector short, then we will have found a short non-zero lattice vector.) One accomplishes this, e.g., by finding a short non-zero vector $\boldsymbol{y}$ in $\mathcal{L}(\boldsymbol{A}_{[i,j]})$ and essentially replacing the first vector in the block with this vector $\boldsymbol{y}$. (Here, we are ignoring how exactly one does this replacement.)

This paradigm was introduced in the celebrated work of Lenstra, Lenstra, and Lovász [25], which described the polynomial-time LLL algorithm. Specifically, (ignoring important details that are not relevant to the present work) the LLL algorithm works by repeatedly replacing Gram-Schmidt vectors $\widetilde{\boldsymbol{a}}_i$ with a shortest non-zero vector in the lattice generated by the dimension-two block $\boldsymbol{A}_{[i,i+1]}$. The LLL algorithm itself has innumerable applications. (See, e.g., [29].) Furthermore, generalizations of LLL yield the most efficient algorithms for finding short non-zero lattice vectors in a wide range of parameter regimes, including those relevant to cryptography.

Specifically, the Block-Korkine-Zolotarev basis-reduction algorithm (BKZ), originally due to Schnorr [31], is a generalization of the LLL algorithm that works with larger blocks. It works by repeatedly modifying blocks $\boldsymbol{A}_{[i,i+\beta-1]}$ of a lattice basis $\boldsymbol{A} := (\boldsymbol{a}_1; \ldots; \boldsymbol{a}_k)$ in order to ensure that the Gram-Schmidt vector $\widetilde{\boldsymbol{a}}_i$ is a shortest non-zero vector in the lattice generated by the block. Here, the parameter $\beta \geq 2$ is called the *block size*, and the case $\beta = 2$ corresponds to the LLL algorithm (ignoring some technical details). Larger block size β yields better bases consisting of shorter lattice vectors. But, to run the algorithm with block size β, we must find shortest non-zero vectors in a β-dimensional lattice, which requires running time $2^{O(\beta)}$ with the best-known algorithms [6, 1, 12]. So, BKZ yields a tradeoff between the quality of the output basis and the running time of the algorithm. (Alternatively, one can view BKZ as a reduction from an approximate lattice problem in high dimensions to an exact lattice problem in lower dimensions, with the approximation factor depending on how much lower the resulting dimension is.)

BKZ is the fastest known algorithm in practice for the problems relevant to cryptography. However, BKZ is notoriously difficult to understand. Indeed, we still do not have a proof that the BKZ algorithm makes at most polynomially many calls to its β-dimensional oracle, nor do we have a tight bound on the quality of the bases output by BKZ, despite much effort. (See, e.g., [34]. However, for both the running time and the output quality of the basis, we now have a very good *heuristic* understanding [16, 11].)

Gama and Nguyen's slide-reduction algorithm is an elegant alternative to BKZ that is far easier to analyze [20]. In particular, it outputs a basis whose quality (e.g., the length of the first vector) essentially matches our heuristic understanding of the behavior of BKZ, and it provably does so with polynomially many calls to a β-dimensional oracle for finding a shortest non-zero lattice vector. Indeed, for a wide range of parameters (including those relevant to cryptography), [20] yields the fastest algorithm with proven correctness for finding short non-zero lattice vectors.

Dual reduction, and some foreshadowing

One of the key ideas used in Gama and Nguyen's slide-reduction algorithm (as well as in other work, such as [28]) is the notion of a *dual-reduced block* $\boldsymbol{A}_{[i,j]}$. The motivation behind dual-reduced blocks starts with the observation that the product $\|\widetilde{\boldsymbol{a}}_i\| \cdots \|\widetilde{\boldsymbol{a}}_j\|$ does not change if the lattice $\mathcal{L}(\boldsymbol{A}_{[i,j]})$ is not changed. Formally, this quantity is the determinant of the lattice $\mathcal{L}(\boldsymbol{A}_{[i,j]})$, which is a lattice invariant. So, while it is perhaps more natural to think of basis reduction in terms of making earlier Gram-Schmidt vectors in a block shorter, with the ultimate goal of making $\boldsymbol{a}_1$ short, one can more-or-less equivalently think of basis reduction in terms of making *later* Gram-Schmidt vectors *longer.*

One therefore defines a *dual-reduced* block as a block $\boldsymbol{A}_{[i,j]}$ such that the last Gram-Schmidt vector $\widetilde{\boldsymbol{a}}_j$ is as *long* as it can be without changing the associated lattice $\mathcal{L}(\boldsymbol{A}_{[i,j]})$. When $\beta := j - i + 1 > 2$, a dual-reduced block is not the same as a block whose first Gram-Schmidt vector is as short as possible. However, there is still some pleasing symmetry here. In particular, it is not hard to show that the last Gram-Schmidt vector $\widetilde{\boldsymbol{a}}_j$ corresponds precisely to a non-zero (primitive) vector in the *dual* lattice of $\mathcal{L}(\boldsymbol{A}_{[i,j]})$ with length $1/\|\widetilde{\boldsymbol{a}}_j\|$. This of course explains the terminology. It also means that making the last Gram-Schmidt vector $\widetilde{\boldsymbol{a}}_j$ as long as possible corresponds to finding a shortest non-zero vector in the dual of $\mathcal{L}(\boldsymbol{A}_{[i,j]})$, while making the first Gram-Schmidt vector $\widetilde{\boldsymbol{a}}_i$ as short as possible of course corresponds to finding a shortest non-zero vector in $\mathcal{L}(\boldsymbol{A}_{[i,j]})$ itself. Either way, this amounts to finding a shortest non-zero vector in a β-dimensional lattice, which takes time that is exponential in the block size β.

1.3 Basis reduction for codes!

As far as the authors are aware, until very recently there was no work attempting to use the ideas from basis reduction in the setting of linear codes. This changed with the recent exciting work of Debris-Alazard, Ducas, and van Woerden, who in particular showed a simple and elegant analogue of the LLL algorithm for codes [17].

Debris-Alazard, Ducas, and van Woerden provide a "dictionary" ([17, Table 1]) for translating important concepts in basis reduction from the setting of lattices to the setting of codes, and it is the starting point of our work. Below, we describe some of the dictionary from [17], as well as some of the barriers that one encounters when attempting to make basis reduction work for codes.

1.3.1 Projection, epipodal vectors, and proper bases

Recall that when one performs basis reduction on lattices, one works with the Gram-Schmidt vectors $\widetilde{\boldsymbol{a}}_i := \pi^{\perp}_{\{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{i-1}\}}(\boldsymbol{a}_i)$ and the projected blocks $\boldsymbol{A}_{[i,j]} := \pi^{\perp}_{\{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{i-1}\}}(\boldsymbol{a}_i;\ldots;\boldsymbol{a}_j)$, i.e., the *orthogonal projection* of $\boldsymbol{a}_i,\ldots,\boldsymbol{a}_j$ onto the subspace $\{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{i-1}\}^{\perp}$ orthogonal to $\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{i-1}$.

So, if we wish to adapt basis reduction to the setting of linear codes (and we do!), it is natural to first ask what the analogue of *projection* is in the setting of codes. [17] gave a very nice answer to this question.[2] In particular, for a vector $\boldsymbol{x} = (x_1,\ldots,x_n) \in \mathbb{F}_q^n$ we call the set of indices i such that x_i is non-zero the *support* of $\boldsymbol{x}$, i.e.,

[2] [17] formally worked with the case $q = 2$ everywhere. Rather than specialize our discussion here to $\mathbb{F}_2$, we will largely ignore this distinction in this part of the introduction. While the more general definitions that we provide here for arbitrary $\mathbb{F}_q$ are new to the present work, when generalizing to $\mathbb{F}_q$ is straightforward, we will not emphasize this in the introduction.

$$\mathsf{Supp}(\boldsymbol{x}) := \{i \in [n] \ : \ x_i \neq 0\} \ .$$

Then, [17] define $\boldsymbol{z} := \pi^{\perp}_{\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_\ell\}}(\boldsymbol{y})$ as follows. If $i \in \bigcup_j \mathsf{Supp}(\boldsymbol{x}_j)$, then $z_i = 0$. Otherwise, $z_i = y_i$. In other words, the projection simply "zeros out the coordinates in the supports of the $\boldsymbol{x}_j$." This notion of projection shares many (but certainly not all) of the features of orthogonal projection in $\mathbb{R}^n$, e.g., it is a linear contraction mapping that is idempotent.

Armed with this notion of projection, [17] then defined the *epipodal vectors* associated with a basis $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n$ as $\boldsymbol{b}_1^+ := \boldsymbol{b}_1, \boldsymbol{b}_2^+ := \pi^{\perp}_{\{\boldsymbol{b}_1\}}(\boldsymbol{b}_2), \ldots, \boldsymbol{b}_n^+ := \pi^{\perp}_{\{\boldsymbol{b}_1,\ldots,\boldsymbol{b}_{n-1}\}}(\boldsymbol{b}_n)$, in analogy with the Gram-Schmidt vectors. In this work, we go a bit further and define

$$\boldsymbol{B}_{[i,j]} := \pi^{\perp}_{\{\boldsymbol{b}_1,\ldots,\boldsymbol{b}_{i-1}\}}(\boldsymbol{b}_i; \ldots ; \boldsymbol{b}_j) \ ,$$

in analogy with the notation in the literature on *lattice* basis reduction.

Here, [17] already encounters a bit of a roadblock. Namely, the epipodal vectors $\boldsymbol{b}_i^+$ can be zero! E.g., if $\boldsymbol{b}_1 = (1, 1, \ldots, 1)$ is the all-ones vector, then $\boldsymbol{b}_i^+$ will be zero for all $i > 1$![3] This is rather troublesome and could lead to many issues down the road. For example, we might even encounter entire blocks $\boldsymbol{B}_{[i,j]}$ that are zero! Fortunately, [17] shows how to get around this issue by defining *proper* bases, which are simply bases for which all the epipodal vectors are non-zero. They then observe that proper bases exist and are easy to compute. (In Section 4, we further develop the theory of proper bases.) So, this particular roadblock is manageable, but it already illustrates that the analogy between projection over $\mathbb{F}_q^n$ and projection over $\mathbb{R}^n$ is rather brittle.

The LLL algorithm for codes then follows elegantly from these definitions. In particular, a basis $\boldsymbol{B} = (\boldsymbol{b}_1; \ldots ; \boldsymbol{b}_k)$ is *LLL-reduced* if it is proper and if $\boldsymbol{b}_i^+$ is a shortest non-zero codeword in the dimension-two code generated by $\boldsymbol{B}_{[i,i+1]}$ for all $i = 1, \ldots, k - 1$. [17] then show a simple algorithm that computes an LLL-reduced basis in polynomial time. Specifically, the algorithm repeatedly makes simple local changes to any block $\boldsymbol{B}_{[i,i+1]}$ for which this condition is not satisfied until the basis is reduced.

In some ways, this new coding-theoretic algorithm is even more natural and elegant than the original LLL algorithm for lattices. For example, the original LLL algorithm had to worry about numerical blowup of the basis entries. And, the original LLL algorithm seems to require an additional slack factor δ in order to avoid the situation in which the algorithm makes a large number of minuscule changes to the basis. Both of these issues do not arise over finite fields, where all vectors considered by the algorithm have entries in $\mathbb{F}_q$ and integer lengths between 1 and n.

1.3.2 What's a good basis and what is it good for?

Given the incredible importance of the LLL algorithm for lattices, it is a major achievement just to show that one can make sense of the notion of "LLL for codes." But, once [17] have defined an LLL-reduced basis for codes and shown how to compute one efficiently, an obvious next question emerges: what can one do with such a basis?

In the case of lattices, the LLL algorithm is useful for many things, but primarily for the two most important computational lattice problems: finding short non-zero lattice vectors and finding close lattice vectors to a target. In particular, the first vector $\boldsymbol{a}_1$ of an LLL-reduced basis is guaranteed to $\|\boldsymbol{a}_1\| \leq 2^{k-1}\lambda_1(\mathcal{L})$. This has proven to be incredibly useful, despite the apparently large approximation factor.

[3] Of course, similar issues do not occur over $\mathbb{R}^n$, because if $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k \in \mathbb{R}^n$ are linearly independent, then $\pi^{\perp}_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{k-1}}(\boldsymbol{a}_k)$ cannot be zero.

For codes over $\mathbb{F}_2$, [17] show that the same is true, namely that if $\boldsymbol{B} = (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k)$ is an LLL-reduced basis for $\mathcal{C} \subseteq \mathbb{F}_2^n$, then $|\boldsymbol{b}_1| \le 2^{k-1} d_{\min}(\mathcal{C})$. They prove this by showing that if $\boldsymbol{b}_i^+$ has minimal length among the non-zero codewords in $\mathcal{C}(\boldsymbol{B}_{[i,i+1]})$, then $|\boldsymbol{b}_i^+| \le 2|\boldsymbol{b}_{i+1}^+|$. It follows in particular that $|\boldsymbol{b}_1| = |\boldsymbol{b}_1^+| \le 2^{i-1}|\boldsymbol{b}_i^+|$ for all i. One can easily see that $d_{\min}(\mathcal{C}) \ge \min_i |\boldsymbol{b}_i^+|$, from which one immediately concludes that $|\boldsymbol{b}_1^+| \le 2^{k-1} d_{\min}(\mathcal{C})$. A simple generalization of this argument shows that over $\mathbb{F}_q$, we have $|\boldsymbol{b}_1^+| \le q^{k-1} d_{\min}(\mathcal{C})$. (We prove something more general and slightly stronger in the full version [21].)

However, notice that all codewords have length at most n and $d_{\min}(\mathcal{C})$ is always at least 1. Therefore, an approximation factor of q^{k-1} is non-trivial only if $n > q^{k-1}$. Otherwise, literally any non-zero codeword has length less than $q^{k-1} d_{\min}(\mathcal{C})$! On the other hand, if $n > q^{k-1}$, then we can anyway find an exact shortest vector in time roughly $q^k n \lesssim n^2$ by simply enumerating all codewords. (The typical parameter regime of interest is when $n = O(k)$.)

In some sense, the issue here is that the space $\mathbb{F}_q^n$ that codes live in is too "cramped." While lattices are infinite sets that live in a space $\mathbb{Q}^n$ with arbitrarily long and arbitrarily short non-zero vectors, codes are finite sets that live in a space $\mathbb{F}_q^n$ in which all non-zero vectors have integer lengths between 1 and n. So, while for lattices, any approximation factor between one and, say, 2^k is very interesting, for codes the region of interest is simply more narrow.

[17] go on to observe that because $|\boldsymbol{b}_{i+1}^+|$ is an integer, for codes over $\mathbb{F}_2$ an LLL-reduced basis must actually satisfy

$$|\boldsymbol{b}_{i+1}^+| \ge \left\lceil \frac{|\boldsymbol{b}_i^+|}{2} \right\rceil .$$

With this slightly stronger inequality together with the fact that $\sum |\boldsymbol{b}_i^+| \le n$, they are able to show that $\boldsymbol{b}_1$ of an LLL-reduced basis will meet the Griesmer bound [22],

$$\sum_{i=1}^{k} \left\lceil \frac{|\boldsymbol{b}_1|}{2^{i-1}} \right\rceil \le n \; , \tag{1}$$

which is non-trivial. E.g., as long as $k \ge \log n$, it follows that

$$|\boldsymbol{b}_1| - \frac{\lceil \log |\boldsymbol{b}_1| \rceil}{2} \le \frac{n-k}{2} + 1 \tag{2}$$

(We generalize this in the full version [21]to show that the appropriate generalization of LLL-reduced bases to codes over $\mathbb{F}_q$ also meet the q-ary Griesmer bound.)

1.3.2.1 Finding close codewords and size reduction

For lattices, Babai also showed how to use an LLL-reduced basis to efficiently find *close* lattice vectors to a given target vector [10], and like the LLL algorithm itself, Babai's algorithm too has innumerable applications. More generally, Babai's algorithm tends to obtain closer lattice vectors if given a "better" basis, in a certain precise sense. [17] showed an analogous "size-reduction" algorithm that finds close codewords to a given target vector, with better performance given a "better" basis. Here, the notion of "better" is a bit subtle, but essentially a basis is "better" if the epipodal vectors tend to have similar lengths. (Notice that $\sum_i |\boldsymbol{b}_i^+| = |\mathsf{Supp}(\mathcal{C})|$, so we cannot hope for all of the epipodal vectors to be short.)

The resulting size-reduction algorithm finds relatively close codewords remarkably quickly. (Indeed, in nearly linear time.) Furthermore, [17] showed how to use their size-reduction algorithm combined with techniques from information set decoding to speed up some information set decoding algorithms for finding short codewords or close codewords to a target, without significantly affecting the quality of the output. For this, their key observation was the fact that typically most epipodal vectors actually have length one (particularly the later epipodal vectors, as one would expect given that later epipodal vectors by definition have more coordinates "zeroed out" by projection orthogonal to the earlier basis vectors) and that their size-reduction algorithm derives most of its advantage from how it treats the epipodal vectors with length greater than one. They therefore essentially run information set decoding on the code projected onto the support of the epipodal vectors with length one and then "lift" the result to a close codeword using their size-reduction algorithm.

They call the resulting algorithm Lee-Brickell-Babai because it is a hybrid of Babai-style size reduction and the Lee-Brickell algorithm [24]. The running time of this hybrid algorithm is dominated by the cost of running information set decoding on a code with dimension $k - k_1$, where

$$k_1 := |\{i \ : \ |\boldsymbol{b}_i^+| > 1\}|$$

is the number of epipodal vectors that do not have length 1. Indeed, the (heuristic) running time of this algorithm is better than Lee-Brickell by a factor that is exponential in k_1, so that even a small difference in k_1 can make a large difference in the running time. They then show that LLL-reduced bases have $k_1 \gtrsim \log n$ (for random codes) and show that this reduction in dimension can offer significant savings in the running time of information set decoding.

Indeed, though the details are not yet public, the current record in the coding problem challenges [8] was obtained by Ducas and Stevens, apparently using such techniques.

1.4 Our contribution

In this work, we continue the study of basis reduction for codes, expanding on and generalizing the results of [17] in many ways, and beginning to uncover a rich landscape of algorithms.

1.4.1 Expanding on the work of [17]

1.4.1.1 Generalization to $\mathbb{F}_q$

Our first set of (perhaps relatively minor) contributions are generalizations of many of the ideas in [17] from $\mathbb{F}_2$ to $\mathbb{F}_q$, a direction proposed in that work. In fact, they quite accurately anticipated this direction. So, we quote directly from [17, Section 1.3]:

> In principle, the definitions, theorems and algorithms of this article should be generalizable to codes over $\mathbb{F}_q$ endowed with the Hamming metric… Some algorithms may see their complexity grow by a factor $\Theta(q)$, meaning that the algorithms remains polynomial-time only for $q = n^{O(1)}$. It is natural to hope that such a generalised LLL would still match [the] Griesmer bound for $q > 2$. However, we expect that the analysis of the fundamental domain [which is necessary for understanding size reduction]… would become significantly harder to carry out.

In Section 3, we generalize from $\mathbb{F}_2$ to $\mathbb{F}_q$ the definitions of projection, epipodal vectors, and proper bases; the definition of an LLL-reduced bases and the LLL algorithm;[4] and the size-reduction algorithm and its associated fundamental domain. Some of this is admittedly

[4] We actually describe the LLL algorithm as a special case of the more general algorithms that we describe below. See the full version [21].

quite straightforward – e.g., given the definitions in [17] of projection, epipodal vectors, and proper bases for codes over $\mathbb{F}_2$, the corresponding definitions for codes over $\mathbb{F}_q$ are immediate (and we have already presented them in this introduction). And, the definition of an LLL-reduced basis and of size reduction follow more-or-less immediately from these definitions. In particular, we do in fact confirm that LLL over $\mathbb{F}_q$ achieves the Griesmer bound.

As [17] anticipated, the most difficult challenge that we encounter here is in the analysis of the fundamental domain that one obtains when one runs size reduction with a particular basis $\boldsymbol{B}$. We refer the reader to the full version [21]for the details.

(We do not encounter the running time issue described in the quote above – except for our algorithm computing the number of vectors of a given length in $\mathcal{F}(\boldsymbol{B}^+)$. In particular, our versions of the LLL algorithm and the size-reduction algorithm – and even our generalizations like slide reduction – run in time that is proportional to a small polynomial in $\log q$.)

Along the way, we make some modest improvements to the work of [17], even in the case of $\mathbb{F}_2$. In particular, using more careful analysis, we shave a factor of roughly n from the proven running time of LLL. (See the full version [21].)

1.4.1.2 More on the theory of proper bases

In order to develop the basis-reduction algorithms that we will describe next, we found that it was first necessary to develop (in Section 4) some additional tools for understanding and working with *proper bases*, which might be of independent interest. Specifically, we define the concept of a *primitive codeword*, which is a non-zero codeword $\boldsymbol{c}$ such that $\mathsf{Supp}(\boldsymbol{c})$ does not strictly contain the support of any other non-zero codeword. We then show that primitive codewords are closely related to proper bases. For example, we show that $\boldsymbol{c}$ is the first vector in some proper basis if and only if $\boldsymbol{c}$ is primitive, and that a basis is proper if and only if the epipodal vectors are primitive vectors in their respective projections.

We find this perspective to be quite useful for thinking about proper bases and basis reduction in general. In particular, we use this perspective to develop algorithms that perform basic operations on proper bases, such as inserting a primitive codeword into the first entry of a basis without affecting properness. The resulting algorithmic tools seems to be necessary for the larger-block-size versions of basis reduction that we describe below, in which our algorithms must make more complicated changes to a basis.

1.4.2 Backward reduction and redundant sets

Our next contribution is the notion of *backward reduction*, described in Section 5. Recall that in the context of lattices, a key idea is the notion of a *dual-reduced* block $\boldsymbol{A}_{[i,j]}$, in which the *last* Gram-Schmidt vector $\widetilde{\boldsymbol{a}}_j$ is as *long* possible, while keeping $\mathcal{L}(\boldsymbol{A}_{[i,j]})$ fixed.

Backward-reduced blocks are what we call the analogous notion for codes. Specifically, we say that a block $\boldsymbol{B}_{[i,j]}$ is *backward reduced* if the last epipodal vector $\boldsymbol{b}_j^+$ is as *long* as possible, while keeping $\mathcal{C}(\boldsymbol{B}_{[i,j]})$ fixed. Just like in the case of lattices, this idea is motivated by an invariant. Here, the invariant is $|\boldsymbol{b}_i^+| + \cdots + |\boldsymbol{b}_j^+|$, which is precisely the support of the code $\mathcal{C}(\boldsymbol{B}_{[i,j]})$. So, if one wishes to make earlier epipodal vectors shorter (and we do!), then one will necessarily make later epipodal vectors longer, and vice versa. In particular, in the case of LLL, when the block size $\beta := j - i + 1$ is equal to 2, there is no difference between minimizing the length of the first epipodal vector and maximizing the length of the second epipodal vector. So, if one wishes, one can describe the LLL algorithm in [17] as working by repeatedly backward reducing blocks $\boldsymbol{B}_{[i,i+1]}$.

The above definition of course leads naturally to two questions. First of all, how do we produce a backward-reduced block (for block size larger than 2)? And, second, what can we say about them? Specifically, what can we say about the length $|\boldsymbol{b}_j^+|$ of the last epipodal vector in a backward-reduced block $\boldsymbol{B}_{[i,j]}$?

One might get discouraged here, as one quickly discovers that long last epipodal vectors *do not* correspond to short non-zero codewords in the dual code. So, the beautiful duality that arises in the setting of lattices simply fails in our new context. (The *only* exception is that last epipodal vectors with length *exactly* two correspond to dual vectors with length *exactly* two.) This is why we use the terminology "backward reduced" rather than "dual reduced." One might fear that the absence of this correspondence would make backward-reduced blocks very difficult to work with.

Instead, we show that long last epipodal vectors $\boldsymbol{b}_j^+$ in a block $\boldsymbol{B}_{[i,j]}$ have a simple interpretation. They correspond precisely to large *redundant sets of coordinates* of the code $\mathcal{C}(\boldsymbol{B}_{[i,j]})$. In the special case when $q = 2$, a redundant set $S \subseteq [n]$ of coordinates is simply a set of coordinates in the support of the code such that for every $a, b \in S$ and every codeword $\boldsymbol{c}$, $c_a = c_b$. For larger q, we instead have the guarantee that $c_a = zc_b$ for fixed non-zero scalar $z \in \mathbb{F}_q^*$ depending only on a and b. In particular, maximal redundant sets correspond precisely to the non-zero coordinates in a last epipodal vector. (See Lemma 8.)

This characterization immediately yields an algorithm for backward reducing a block. (See Algorithm 2.) In fact, finding a backward-reduced block boils down to finding a set of most common elements in a list of at most n non-zero columns, each consisting of $\beta := j-i+1$ elements from $\mathbb{F}_q$. One quite surprising consequence of this is that one can actually find backward-reduced blocks efficiently, even for large β! (Compare this to the case of lattices, where finding a dual-reduced block for large β is equivalent to the NP-hard problem of finding a shortest non-zero vector in a lattice of dimension β.)

Furthermore, this simple combinatorial characterization of backward-reduced blocks makes it quite easy to prove a simple tight lower bound on the length of $\boldsymbol{b}_j^+$ in a backward-reduced block $\boldsymbol{B}_{[i,j]}$. (See the full version [21].) Indeed, such a proof follows immediately from the pigeonhole principle. This makes backward-reduced blocks quite easy to analyze. In contrast, as we will see below, *forward-reduced* blocks, in which the first epipodal vector $\boldsymbol{b}_i^+$ is as short as possible, are rather difficult to analyze for $\beta > 2$.

1.4.3 Fully backward-reduced bases

With this new characterization of backward-reduced blocks and the realization that we can backward reduce a block quite efficiently, we go on to define the notion of a *fully backward-reduced basis.* We say that a basis is *fully backward reduced* if *all* of the prefixes $\boldsymbol{B}_{[1,j]}$ are backward reduced for all $1 \leq j \leq k$.[5] In fact, we are slightly more general than this, and consider bases that satisfy this requirement for all j up to some threshold $\tau \leq k$.

We show that a fully backward-reduced basis achieves the Griesmer bound (Equation (1) for $q = 2$), just like an LLL-reduced basis. This is actually unsurprising, since it is not difficult to see that when the threshold $\tau = k$ is maximal, a fully backward-reduced basis is also LLL reduced. However, even when $\tau = \log_q n$, we still show that a backward-reduced basis achieves the Griesmer bound. (See Theorem 16.)

[5] Notice that this implies that $\boldsymbol{B}_{[i,j]}$ is also backward reduced for any $1 \leq i < j$. So, such bases really are *fully* backward reduced.

We then show a very simple and very efficient algorithm for computing fully backward-reduced bases. In particular, if the algorithm is given as input a *proper* basis, then it will convert it into a fully backward-reduced basis up to threshold τ in time $O(\tau^2 n \operatorname{polylog}(n, q))$. Notice that this is *extremely* efficient when $\tau \le \operatorname{poly}(\log_q n)$.[6] Indeed, for most parameters of interest, this running time is in fact less than the time $O(nk \log q)$ needed simply to read the input basis $\boldsymbol{B} \in \mathbb{F}_q^{k \times n}$. (Of course, this is possible because the algorithm only looks at the first τ rows of the input basis.) So, if one already has a proper basis, one can convert it into a fully backward-reduced basis nearly for free.[7]

In contrast, the LLL algorithm runs in time $O(kn^2 \log^2 q)$. One *can* perform a similar "threshold" trick and run the LLL algorithm only on the first τ basis vectors for $\tau = \lceil \log_q n \rceil$ (which would still imply that $|\boldsymbol{b}_1|$ must be bounded by the Griesmer bound). But, this would still yield a running time of $\Omega(\tau n^2 \log^2 q)$ in the worst case. The speedup that we achieve from fully backward reduction comes from the combination of this threshold trick together with the fact that fully backward reduction runs in time proportional to $\tau^2 n$, rather than τn^2.

Furthermore, we show empirically that the resulting algorithm tends to produce better bases than LLL in practice. (See the full version [21].)

(It seems unlikely that any similar algorithm exists for lattices for two reasons. First, in the setting of lattices, computing a dual-reduced basis for large block sizes is computationally infeasible. Second, while for codes it is not unreasonable to look for a short non-zero codeword in the subcode generated by the first τ basis vectors, for lattices the lattice generated by the first $k-1$ basis vectors often contains no shorter non-zero vectors than the basis vectors themselves, even when the full lattice contains much shorter vectors.)

1.4.3.1 Heuristic analysis of full backward reduction

We also provide heuristic analysis of full backward reduction, providing a compelling heuristic explanation for why its performance in practice seems to be much better than what worst-case analysis suggests. In particular, recall that we essentially characterize the length of the last epipodal vector of a backward-reduced block $\boldsymbol{B}_{[i,j]}$ in terms of the maximal number of times that a column in $\boldsymbol{B}_{[i,j]}$ repeats. We then naturally use the pigeonhole principle to argue that for suitable parameters there must be a column that repeats many times.

E.g., for $q = 2$, there must be at least one repeated non-zero column if the number of non-zero columns s is larger than the number of possible non-zero columns $2^\beta - 1$, where $\beta := j - i + 1$ is the length of a column. This analysis is of course tight in the worst case. However, in the average case, we know from the birthday paradox that we should expect to see a repeated column even if s is roughly $2^{\beta/2}$, rather than 2^β.

So, under the (mild but unproven) heuristic that the blocks $\boldsymbol{B}_{[1,j]}$ in a fully backward-reduced basis behave like a random matrices for all j (in terms of the number of redundant coordinates), it is easy to see that $k_1 \gtrsim 2 \log_q n$, which is significantly better than what LLL achieves (both in the worst case and empirically).

This heuristic argument is backed up by experiments. (See the full version [21].) We also show (in the full the version [21]) a less natural variant of this algorithm that provably achieves $k_1 \gtrsim 2 \log_q n$ when its input is a random matrix. This variant works by carefully

[6] We argue (in the full version [21]) that there is not much point in taking τ significantly greater than $\log_q^2(n)$.

[7] Computing a proper basis seems to require time $O(nk^2 \log^2 q)$ (without using fast matrix multiplication algorithms), but in many contexts the input basis is in systematic form and is therefore proper.

"choosing which coordinates to look at" for each block, in order to maintain independence. We view this as an additional heuristic explanation for full backward reduction's practical performance, since one expects an algorithm that "looks at all coordinates" to do better than one that does not.

This result about k_1 for backward-reduced bases also compares favorably with the study of the LLL algorithm in [17]. In particular, in [17], they proved that LLL achieves $k_1 \gtrsim \log n$ for a random code for $q = 2$, but in their experiments they observed that LLL combined with a preprocessing step called EpiSort actually seems to achieve $k_1 \approx c \log n$ for some constant $1 < c \leq 2$. However, the behavior of LLL and EpiSort seems to be much more subtle than the behavior of full backwards reduction. We therefore still have no decent explanation (even a heuristic one) for why LLL and EpiSort seem to achieve $k_1 \approx c \log n$ or for what the value of this constant c actually is.

1.4.4 BKZ and slide reduction for codes

Our next set of contributions are adaptations of the celebrated BKZ and slide-reduction algorithms to the setting of codes.

1.4.4.1 BKZ for codes

Our analogue of the BKZ algorithm for codes is quite natural.[8] Specifically, our algorithm works by repeatedly checking whether the epipodal vector $\boldsymbol{b}_i^+$ is a shortest non-zero codeword in the code generated by the block $\boldsymbol{B}_{[i,i+\beta-1]}$. If not, it updates the basis so that this is the case (using the tools that we have developed to maintain properness). The algorithm does this repeatedly until no further updates are possible. At least intuitively, a larger choice of β here requires a slower algorithm because the resulting algorithm will have to find shortest non-zero codewords in β-dimensional codes. But, larger β will result in a better basis.

As we mentioned above, in the setting of lattices, the BKZ algorithm is considered to be the best performing basis-reduction algorithm in most parameter regimes, but it is notoriously difficult to analyze. We encounter a roughly similar phenomenon in the setting of linear codes. In particular, we run experiments that show that the algorithm performs quite well in practice. (Though it requires significantly more running time than full backward reduction to achieve a similar profile. See the full version [21].) However, we are unable to prove that it terminates efficiently, except in the special case of $\beta = 2$, in which case we recover the LLL algorithm of [17]. For $\beta > 2$, we offer only an extremely weak bound on the running time. As in the case of lattices, the fundamental issue is that it is difficult to control the effect that changing $\boldsymbol{b}_i^+$ can have on the other epipodal vectors $\boldsymbol{b}_{i+1}^+, \ldots, \boldsymbol{b}_{i+\beta-1}^+$ in the block.

Here, we encounter an additional issue as well. In the case of lattices, there is a relatively simple tight bound on the minimum distance of the lattice generated by the block $\boldsymbol{A}_{[i,i+\beta-1]}$ to the lengths of the Gram-Schmidt vectors $\|\widetilde{\boldsymbol{a}}_i\|, \ldots, \|\widetilde{\boldsymbol{a}}_{i+\beta-1}\|$ in the block. In particular, Minkowski's celebrated theorem tells us that $\lambda_1(\mathcal{L}(\boldsymbol{A}_{[i,i+\beta-1]})) \leq C\sqrt{\beta}(\|\widetilde{\boldsymbol{a}}_i\| \cdots \|\widetilde{\boldsymbol{a}}_{i+\beta-1}\|)^{1/\beta}$ for some constant $C > 0$, and one applies this inequality repeatedly with different i to understand the behavior of basis reduction for lattices.

[8] We note that the name "BKZ algorithm for codes" is perhaps a bit misleading. In the case of lattices, the BKZ algorithm is named after Korkine and Zolotarev due to their work on Korkine-Zolotarev-reduced bases (which can be thought of as BKZ-reduced bases with maximal block size $\beta = k$, and is sometimes also called a *Hermite*-Korkine-Zolotarev-reduced basis). A *Block*-Korkine-Zolotarev-reduced basis is (unsurprisingly) a basis in which each block $\boldsymbol{B}_{[i,i+\beta-1]}$ is a Korkine-Zolotarev-reduced basis. For codes, the analogous notion of a Korkine-Zolotarev-reduced basis was called a Griesmer-reduced basis in [17]. So, we should perhaps call our notion "Block-Griesmer-reduced bases" and the associated algorithm "the block-Griesmer algorithm." However, the authors decided to use the term "BKZ" here in an attempt to keep terminology more consistent between lattices and codes.

However, in the case of codes, there is no analogous simple tight bound on $d_{\min}(\mathcal{C}(\boldsymbol{B}_{[i,i+\beta-1]}))$ in terms of the lengths of the epipodal vectors $|\boldsymbol{b}_i^+|, \ldots, |\boldsymbol{b}_{i+\beta-1}^+|$, *except* in the special case when $\beta = 2$. Instead, there are many known incomparable upper bounds on $d_{\min}$ in terms of the dimension β and the support size $s := |\boldsymbol{b}_i^+| + \cdots + |\boldsymbol{b}_{i+\beta-1}|$ (and, of course, the alphabet size q). Each of these bounds is tight or nearly tight for some support sizes s (for fixed β) but rather loose in other regimes. The nature of our basis-reduction algorithms is such that different blocks have very different support sizes s, so that we cannot use a single simple bound that will be useful in all regimes. And, due to the relatively "cramped" nature of $\mathbb{F}_q^n$, applying loose bounds on $d_{\min}$ can easily yield trivial results, or results that do offer no improvement over the $\beta = 2$ case. As a result, the bound that we obtain on the length of $\boldsymbol{b}_1$ for a BKZ-reduced basis does not have a simple closed form. (Since the special case of $\beta = 2$ yields a very simple tight bound $d_{\min} \leq (1 - 1/q)s$, this is not an issue in the analysis of the LLL algorithm in [17].)

In fact, we do not even know if the worst-case bound on $|\boldsymbol{b}_1|$ for a BKZ-reduced basis is efficiently computable, even if one knows the optimal minimum distance of β-dimensional codes for all support sizes. However, we do show an efficiently computable bound that is nearly as good. And, we show empirically that in practice it produces quite a good basis. (See the full version [21].)

1.4.4.2 Slide reduction for codes

Given our difficulties analyzing the BKZ algorithm, it is natural to try to adapt Gama and Nguyen's slide-reduction algorithm [20] from lattices to codes. In particular, recall that in the case of lattices, the slide-reduction algorithm has the benefit that (unlike BKZ) it is relatively easy to prove that it terminates efficiently.

In fact, recall that the idea for backward-reduced bases was inspired by dual-reduced bases for lattices, which are a key component of slide reduction. We therefore define slide-reduced bases for codes by essentially just substituting backward-reduced blocks for dual-reduced blocks in Gama and Nguyen's definition for lattices. Our slide-reduction algorithm (i.e., an algorithm that produces slide-reduced bases) follows similarly.

We then give a quite simple proof that this algorithm terminates efficiently. Indeed, our proof is a direct translation of Gama and Nguyen's elegant potential-based argument from the case of lattices to the case of codes. (Gama and Nguyen's proof is itself a clever variant of the beautiful original proof for the case when $\beta = 2$ in [25].)

Finally, we give an efficiently computable upper bound on $|\boldsymbol{b}_1|$ for a slide-reduced basis in a similar spirit to our upper bound on BKZ. Here, we again benefit from our analysis of backward-reduced blocks described above. Indeed, the behavior of the epipodal vectors in our backward-reduced blocks is quite easy to analyze. However, our bound does not have a simple closed form because the behavior of the forward-reduced blocks still depends on the subtle relationship between the minimal distance of a code and the parameters n and k, as we described in the context of BKZ above.

In our experiments (in the full version [21]), slide reduction is far faster than BKZ but does not find bases that are as good.

1.4.5 Two illustrative algorithms

In the full version [21], we show yet two more basis-reduction algorithms for codes. We think of the importance of these algorithms as being less about their actual usefulness and more about what they show about the potential and limitations of basis reduction for codes. We explain below.

1.4.5.1 One-block reduction

The one-block-reduction algorithm is quite simple. It finds a short non-zero codeword in a code $\mathcal{C}$ generated by some basis $\boldsymbol{B}$ by first ensuring that $\boldsymbol{B}$ is proper, and then by simply finding a shortest non-zero codeword in the subcode $\mathcal{C}(\boldsymbol{B}_{[1,\beta]})$ generated by the prefix basis $\boldsymbol{B}_{[1,\beta]}$. Notice that if $\beta \leq O(\log_q n)$, then this algorithm runs in polynomial time. In particular, enumerating all codewords in the subcode can be done in time roughly $O(nq^\beta \log q)$.

Furthermore, it is not hard to see that when $\beta = \lceil \log_q n \rceil$, this simple algorithm actually meets the Griesmer bound! (See the full version [21].) At a high level, this is because (1) the worst case in the Griesmer bound has $|\boldsymbol{b}_i^+| = 1$ for all $i \geq \beta$; and (2) the resulting bound is certainly not better than the minimum distance of a code with dimension β and support size $n-(k-\beta)$. Here, the $k-\beta$ term comes from the fact that $\mathsf{Supp}(\boldsymbol{B}_{[1,\beta]}) = n-|\boldsymbol{b}_{\beta+1}^+|-\cdots-|\boldsymbol{b}_k^+|$. (Similar logic explains why full backward reduction achieves the Griesmer bound with $\tau \approx \log_q n$.)

More generally, it seems unlikely that a basis-reduction algorithm will be able to find $\boldsymbol{b}_1$ that is shorter than what is achieved by this simple approach if we take $\beta \geq \max\{k_1^*, \beta'\}$, where β' is the size of the largest block in the basis reduction algorithm and k_1^* is the maximal index of an epipodal vector that has length larger than one. (In practice, k_1^* is almost never much larger than k_1.) In particular, for a basis reduction algorithm to do better than this, it must manage to produce a block $\boldsymbol{B}_{[1,\beta]}$ that has minimum distance less than what one would expect given its support size.

We therefore think of this algorithm as illustrating two points.

First, the existence of this algorithm further emphasizes the importance of the parameter k_1^* (and the closely related parameter k_1) as a sort of "measure of non-triviality." If an algorithm achieves large k_1^*, then the above argument becomes weaker, since we must take $\beta \geq k_1^*$. Indeed, if β is significantly larger than $2\log_q k$, then the running time of one-block reduction (if implemented by simple enumeration) becomes significantly slower.

Second, the existence of the one-block-reduction algorithm illustrates that we should be careful not to judge basis-reduction algorithms *entirely* based on $|\boldsymbol{b}_1|$. We certainly think that $|\boldsymbol{b}_1|$ is an important measure of study, and indeed it is the main way that we analyze the quality of our bases in this work. However, the fact that one-block-reduction exists shows that this should not be viewed as the only purpose of a basis-reduction algorithm.

Of course, the algorithms that we have discussed thus far are in fact non-trivial, because they (1) find short non-zero codewords *faster* than one-block reduction; and (2) find whole reduced bases and not just a single short non-zero codeword. Such reduced bases have already found exciting applications in [17] and [8], and we expect them to find more.

1.4.5.2 Approximate Griesmer reduction

Recall that [17] calls a basis $\boldsymbol{B} \in \mathbb{F}_q^{k\times n}$ *Griesmer reduced* if $\boldsymbol{b}_i^+$ is a shortest non-zero codeword in $\mathcal{C}(\boldsymbol{B}_{[i,k]})$ for all i. And, notice that, if one is willing to spend the time to find shortest non-zero codewords in codes with dimension at most k, then one can compute a Griesmer-reduced basis iteratively, by first setting $\boldsymbol{b}_1$ to be a shortest non-zero codeword in the whole code, then projecting orthogonal to $\boldsymbol{b}_1$ and building the rest of the basis recursively. (Griesmer-reduced bases are the analogue of Korkine-Zolotarev bases for lattices. We discuss Griesmer-reduced bases more below.)

Our approximate-Griesmer-reduction algorithm is a simple variant of this idea. In particular, it is really a family of algorithms parameterized by a subprocedure that finds short (but not necessarily shortest) non-zero codewords in a code. Given such a subprocedure, the algorithm first finds a short non-zero codeword $\boldsymbol{b}_1$ in the input code $\mathcal{C}$. It then projects the code orthogonally to $\boldsymbol{b}_1$ and builds the rest of the basis recursively. (To make sure that we end up with a proper basis, care must be taken to assure that $\boldsymbol{b}_1$ is primitive. We ignore this in the introduction. See the full version [21].)

The running time of this algorithm and the quality of the basis produced of course depends on the choice of subprocedure. Given the large number of algorithms for finding short non-zero codewords with a large variety of performance guarantees for different parameters (some heuristic and some proven), we do not attempt here to study this algorithm in full generality. We instead simply instantiate it with the Lee-Brickell-Babai algorithm from [17] (an algorithm which itself uses [17]'s LLL algorithm as a subroutine). Perhaps unsurprisingly, we find that this produces significantly better basis profiles (e.g., smaller $|\boldsymbol{b}_1|$ and larger k_1 and k_1^*) than all of the algorithms that we designed here. The price for this is, of course, that the subprocedure itself must run in enough time to find non-zero short codewords in dimension k codes.

We view this algorithm as a proof of concept, showing that at least in principle one can combine basis-reduction techniques with other algorithms for finding short codewords to obtain bases with very good parameters. This meshes naturally with the Lee-Brickell-Babai algorithm in [17], which shows how good bases can be combined with other algorithmic techniques to find short non-zero codewords. Perhaps one can merge these techniques more in order to show a way to use a good basis to find a better basis, which itself can be used to find a better basis, etc?

1.4.6 On "the best possible bases"

Finally, in the full version [21], we prove bounds on "the best possible bases" in terms of the parameters k_1 and k_1^*. Indeed, recall that the (heuristic) running time of [17]'s Lee-Brickell-Babai algorithm beats Lee-Brickell by a factor that is exponential in k_1. And, we argued above that k_1^* can be viewed as a measure of the "non-triviality" of a basis reduction algorithm. So, it is natural to ask how large k_1 and k_1^* can be in principle.

In the full version [21], we show that *any* code over $\mathbb{F}_2$ has a basis with $k_1^* \geq \Omega(\log k^2)$, even if the support size is as small as $n = k + \sqrt{k}$. For this, we use Griesmer-reduced bases (not to be confused with the approximate-Griesmer-reduced bases described above; note in particular that it is NP-hard to compute a Griesmer-reduced basis). Notice that this is a factor of $\Omega(\log k)$ better than the logarithmic k_1^* achieved by all known efficient basis-reduction algorithms.

Here, we use the parameter k_1^* and not k_1 because it is easy to see that in the worst case a code can have arbitrarily large support but still have no proper basis with $k_1 > 1$.[9] Typically, of course, one expects k_1^* and k_1 to be very closely related, so that one can view this as heuristic evidence that typical codes have bases with $k_1 \geq \Omega(\log^2 k)$.

In the full version [21], we argue under a mild heuristic assumption that any basis for a random code over $\mathbb{F}_2$ has $k_1 \leq k_1^* \leq O(\log^2 k)$, even if the support size n is a large polynomial in the dimension k.

Taken together, these results suggest that the best possible bases that we should expect to find in practice should have $k_1 \approx k_1^* = \Theta(\log^2 k)$ for typical settings of parameters. Such a basis would (heuristically) yield a savings of $k^{\Theta(\log k)}$ in [17]'s Lee-Brickell-Babai algorithm. So, it would be very exciting to find an efficient algorithm that found such a basis.

[9] For example, take that code $\mathbb{F}_2^{k-1} \cup (\mathbb{F}_2^{k-1} + \boldsymbol{c})$ where $\boldsymbol{c} := (1, 1, \ldots, 1) \in \mathbb{F}_2^n$. Any proper basis of this code must have $k - 1$ vectors with length one and therefore must have $k_1 = 1$.

On the other hand, our (heuristic) upper bound on k_1 suggests a limitation of basis reduction for codes. In particular, we should not expect any improvement better than $k^{\Theta(\log k)}$ in Lee-Brickell-Babai. And, the upper bound also suggests that basis-reduction algorithms are unlikely to outperform the simple one-block-reduction algorithm for block sizes larger than $\Omega(\log^2 k)$.

2 Preliminaries

2.1 Some notation

Logarithms are base two unless otherwise specified, i.e., $\log(2^x) = x$. We write $\boldsymbol{I}_m$ for the $m \times m$ identity matrix.

If $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k \in \mathbb{F}_q^n$, then $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k) \in \mathbb{F}_q^{n \times k}$ denotes the matrix where each $\boldsymbol{b}_i$ is a column and $(\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k) \in \mathbb{F}_q^{k \times n}$ denotes the matrix where each $\boldsymbol{b}_i$ is row i of $\boldsymbol{B}$.

We say that a matrix $\boldsymbol{B} \in \mathbb{F}_q^{k \times n}$ is *in systematic form* if $\boldsymbol{A} = (\mathbf{I}_k, \boldsymbol{X})\boldsymbol{P}$, where $\boldsymbol{P}$ is a permutation matrix (i.e., if k contains the columns $\boldsymbol{e}_1^T, \ldots, \boldsymbol{e}_k^T$).

For any basis $\boldsymbol{B} \in \mathbb{F}_q^{k \times n}$ and any subset $S \subseteq [n]$ with $|S| = k$ such that $\boldsymbol{B}_S$ has full rank, we call the process of replacing $\boldsymbol{B}$ by $(\boldsymbol{B}|_S)^{-1}$ *systematizing $\boldsymbol{B}$ with respect to S*. When the set S is not important, we simply call this *systematizing $\boldsymbol{B}$*. This procedure is useful at least in part because it results in a proper basis.

We define two notions of the support of a vector. Specifically, we write

$$\mathsf{Supp}(\boldsymbol{x}) := \{i \in [\![1, n]\!] : x_i \neq 0\} ,$$

and similarly

$$\overrightarrow{\mathbf{Supp}}(\boldsymbol{x})_i := \begin{cases} 0 & x_i = 0 \\ 1 & x_i \neq 0 \,. \end{cases}$$

We can also define the support of an $[n, k]_q$ code $\mathcal{C}$ by extending the definitions of Supp and $\overrightarrow{\mathbf{Supp}}$,

$$\mathsf{Supp}(\mathcal{C}) \triangleq \bigcup_{\boldsymbol{c} \in \mathcal{C}} \mathsf{Supp}(\boldsymbol{c}) \qquad\qquad \overrightarrow{\mathbf{Supp}}(\mathcal{C}) = \bigvee_{\boldsymbol{c} \in \mathcal{C}} \overrightarrow{\mathbf{Supp}}(\boldsymbol{c}) \,,$$

and we define the support of a matrix $\boldsymbol{B} \in \mathbb{F}_q^{k \times n}$ as the support of the code generated by the matrix.

If $\boldsymbol{A} \in \mathbb{F}_q^{m \times n}$, $\boldsymbol{B} \in \mathbb{F}_q^{r \times s}$, then the direct sum of $\boldsymbol{A}$ and $\boldsymbol{B}$, denoted $\boldsymbol{A} \oplus \boldsymbol{B} \in \mathbb{F}_q^{(m+r) \times (n+s)}$, is

$$\boldsymbol{A} \oplus \boldsymbol{B} = \begin{pmatrix} \boldsymbol{A} & \mathbf{0}_{m \times s} \\ \mathbf{0}_{n \times r} & \boldsymbol{B} \end{pmatrix}$$

We will often use the following important property regarding matrix direct sums. If $\boldsymbol{A} \in \mathbb{F}_q^{m \times n}$, $\boldsymbol{B} \in \mathbb{F}_q^{r \times s}, \boldsymbol{x} \in \mathbb{F}_q^n, \boldsymbol{y} \in \mathbb{F}_q^s$, then

$$(\boldsymbol{A} \oplus \boldsymbol{B}) \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}\boldsymbol{x} \\ \boldsymbol{B}\boldsymbol{y} \end{pmatrix} \,.$$

3 Generalizing epipodal vectors, size reduction, and the fundamental domain to $\mathbb{F}_q$

In this section, we generalize many of the fundamental concepts in [17] from codes over $\mathbb{F}_2$ to codes over $\mathbb{F}_q$. Specifically, we generalize the notions of projection, epipodal matrices, and the size-reduction algorithm. We then study the geometry of the fundamental domain that one obtains by running the size-reduction algorithm on a given input basis.

Much of this generalization is straightforward (once one knows the theory developed for $\mathbb{F}_2$ in [17]). So, one might read much of this section as essentially an extension of the preliminaries. The most difficult part, in the full version [21], is the analysis of the fundamental domain (which is not used in the rest of the paper).

3.1 Projection and epipodal vectors

The notions of projection and epipodal vectors extend naturally to $\mathbb{F}_q$ from the notions outlined in [17]. However, to ensure that this work is as self-contained as possible, we will now explicitly outline how some of those notions extend to $\mathbb{F}_q$. Notice that these operations are roughly analogous to orthogonal projection maps over $\mathbb{R}^n$.

▶ **Definition 1.** *If $\boldsymbol{x}_1 = (x_{1,1}, \ldots, x_{1,n}), \ldots, \boldsymbol{x}_k = (x_{k,1}, \ldots, x_{k,n}) \in \mathbb{F}_q^n$, the function $\pi_{\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_k\}} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ is defined as follows:*

$$\pi_{\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_k\}}(\boldsymbol{y})_i = \begin{cases} y_i & x_{1,i} \neq 0 \vee \cdots \vee x_{k,i} \neq 0 \\ 0 & \textit{otherwise.} \end{cases}$$

We call this "projection onto the support of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$."

▶ **Definition 2.** *If $\boldsymbol{x}_1 = (x_{1,1}, \ldots, x_{1,n}), \ldots, \boldsymbol{x}_k = (x_{k,1}, \ldots, x_{k,n}) \in \mathbb{F}_q^n$, the function $\pi^{\perp}_{\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_k\}} : \mathbb{F}_q^n \to \mathbb{F}_q^n$ is defined as follows:*

$$\pi^{\perp}_{\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_k\}}(\boldsymbol{y})_i = \begin{cases} y_i & x_{1,i} = 0 \wedge \cdots \wedge x_{k,i} = 0 \\ 0 & \textit{otherwise.} \end{cases}$$

We call this "projection orthogonal to $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$."

We will often simply write $\pi_{\boldsymbol{x}}$ to denote $\pi_{\{\boldsymbol{x}\}}$ and $\pi^{\perp}_{\boldsymbol{x}}$ to denote $\pi^{\perp}_{\{\boldsymbol{x}\}}$

We now define the epipodal matrix of a basis for a code, which is the analogue of the Gram–Schmidt matrix.

▶ **Definition 3.** *Let $\boldsymbol{B} = (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k) \in \mathbb{F}_q^{k \times n}$ be a matrix with elements from $\mathbb{F}_q$. The ith projection associated to the matrix $\boldsymbol{B}$ is defined as $\pi_i := \pi^{\perp}_{\{\boldsymbol{b}_1,\ldots,\boldsymbol{b}_{i-1}\}}$, where π_1 denotes the identity.*

The ith epipodal vector *is defined as $\boldsymbol{b}_i^+ := \pi_i(\boldsymbol{b}_i)$. The matrix $\boldsymbol{B}^+ := (\boldsymbol{b}_1^+; \ldots; \boldsymbol{b}_k^+) \in \mathbb{F}_q^{k \times n}$ is called the epipodal matrix of $\boldsymbol{B}$.*

The following notation for a projected block will be helpful in defining our reduction algorithms. (The same notation is used in the lattice literature.)

▶ **Definition 4.** *For a basis $\boldsymbol{B} = (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k) \in \mathbb{F}_q^{k \times n}$ and $i, j \in [1, k]$ where $i \leq j$, we use the notation $\boldsymbol{B}_{[i,j]}$ as shorthand for $(\pi_i(\boldsymbol{b}_i); \ldots; \pi_i(\boldsymbol{b}_j))$. Furthermore, for $i \in [1, k]$ and $j > k$, we define $\boldsymbol{B}_{[i,j]} = \boldsymbol{B}_{[i,k]}$ for all $j > k$.*

We will often write ℓ_i to denote $|\boldsymbol{b}_i^+|$ when the basis $\boldsymbol{B} = (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k)$ is clear from context.

3.1.1 Basic operations on blocks

See the full version [21].

3.2 Size reduction and its fundamental domain

See the full version [21].

4 Proper bases and primitivity

We will primarily be interested in bases that are *proper* in the sense that the epipodal vectors should all be non-zero.

▶ **Definition 5.** *A basis is said to be* proper *if all its epipodal vectors* $\boldsymbol{b}_i^+$ *are non-zero.*

[17] observed that, given an arbitrary basis $\boldsymbol{B} \in \mathbb{F}_q^{k \times n}$ for a code, we can efficiently compute a proper basis $\boldsymbol{B}'$ for the same code by systematizing $\boldsymbol{B}$. In particular, let $\boldsymbol{A} \in \mathbb{F}_q^{k \times k}$ be an invertible matrix formed from k columns of $\boldsymbol{B}$ (which must exist because $\boldsymbol{B}$ is a full-rank matrix). Then, $\boldsymbol{B}' := \boldsymbol{A}^{-1}\boldsymbol{B}$ is a proper basis for the code generated by $\boldsymbol{B}$. In particular, every code has a proper basis. From this, we derive the following simple but useful fact.

See the full version [21].

5 Redundant sets of coordinates, the last epipodal vector, and backward reduction

We are now ready to develop the theory behind backward-reduced bases. A backward-reduced basis is one in which the last epipodal vector $\boldsymbol{b}_k^+$ is as *long* as possible. In the context of lattices, such bases are called dual-reduced bases and the maximal length of the last Gram-Schmidt vector has a simple characterization in terms of λ_1 of the dual lattice. For codes, the maximal length of the last epipodal vector behaves rather differently, as we will explain below. In particular, we will see how to find a backward-reduced basis quite efficiently. In contrast, finding a dual-reduced basis is equivalent to finding a shortest non-zero vector in a lattice and is therefore NP-hard.

On our way to defining backward reduction, we first define the notion of *redundant* coordinates. Notice that we only consider coordinates in the support of $\mathcal{C}$.

▶ **Definition 6.** *For a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, we say that a set $S \subseteq [n]$ of coordinates is* redundant for $\mathcal{C}$ *if $S \subseteq \mathsf{Supp}(\mathcal{C})$ and for every $\boldsymbol{c} \in \mathcal{C}$ and all $i, j \in S$, $c_i = 0$ if and only if $c_j = 0$.*

The following simple claim explains the name "redundant." In particular, for any codeword $\boldsymbol{c} \in \mathcal{C}$, if we know c_i for some $i \in S$, then we also know c_j for any $j \in S$.

▷ Claim 7. For a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, a set $S \subseteq \mathsf{Supp}(\mathcal{C})$ is redundant for $\mathcal{C}$ if and only if for every $i, j \in S$, there exists a non-zero scalar $a \in \mathbb{F}_q^*$ such that for all $\boldsymbol{c} \in \mathcal{C}$, $c_j = ac_i$.

Furthermore, to determine whether S is a set of redundant coordinates, it suffices to check whether the latter property holds for all $\boldsymbol{c} := \boldsymbol{b}_i$ in a basis $(\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k)$ of $\mathcal{C}$.

Proof. See the full version [21]. ◁

Next, we show that redundancy is closely connected with the last epipodal vector in a basis.

▶ **Lemma 8.** *For a code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with dimension k and $S \subseteq [n]$, there exists a basis $\boldsymbol{B} := (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k)$ of $\mathcal{C}$ with $S \subseteq \mathsf{Supp}(\boldsymbol{b}_k^+)$ if and only if S is redundant.*

Furthermore, if S is redundant, then there exists a proper basis with this property.

Proof. See the full version [21]. ◀

The above motivates the following definition.

▶ **Definition 9.** *For a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, the repetition number of $\mathcal{C}$, written $\eta(\mathcal{C})$, is the maximal size of a redundant set $S \subseteq \mathsf{Supp}(\mathcal{C})$.*

In particular, notice that by Lemma 8, $\eta(C)$ is also the maximum of $|\boldsymbol{b}_k^+|$ over all bases $(\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k)$ *and* this maximum is achieved by a proper basis. The next lemma gives a lower bound on $\eta(\mathcal{C})$, therefore showing that codes with sufficiently large support and sufficiently low rank must have bases whose last epipodal vector is long.

▶ **Lemma 10.** *For any code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with dimension k,*

$$\eta(\mathcal{C}) \geq \left\lceil \frac{q-1}{q^k - 1} \cdot |\mathsf{Supp}(\mathcal{C})| \right\rceil .$$

Proof. See the full version [21]. ◀

We present in Algorithm 1 a simple algorithm that finds the largest redundant set of a code $\mathcal{C}$. (The algorithm itself can be viewed as a constructive version of the proof of Lemma 10.)

Algorithm 1 Max Redundant Set.

```
Input: A basis $\boldsymbol{B} = (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k) \in \mathbb{F}_q^{k \times n}$ for $\mathcal{C}$
Output: A redundant set $S$ for $\mathcal{C}$ with $|S| = \eta(\mathcal{C})$
for $j \in [n]$ do
    $a_j \leftarrow B_{i,j}^{-1}$, where $i \in [k]$ is minimal such that $B_{i,j} \neq 0$.
end
Find $S \subseteq \mathsf{Supp}(\mathcal{C})$ with maximal size such that for all $j_1, j_2 \in S$ and all $i$,
  $a_{j_1} B_{i,j_1} = a_{j_2} B_{i,j_2}$.
return $S$
```

▷ Claim 11. Algorithm 1 outputs a redundant set S for $\mathcal{C}$ with $|S| = \eta(\mathcal{C})$. Furthermore, Algorithm 1 runs in time $O(kn \log(q) \log(qn))$ (when implemented appropriately).

Proof. See the full version [21]. ◁

5.1 Backward reduction

We are now ready to present our definition of backward-reduced bases.

▶ **Definition 12.** *Let $\boldsymbol{B} = (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k) \in \mathbb{F}_q^{k \times n}$ be a basis of a code $\mathcal{C}$. We say that $\boldsymbol{B}$ is* backward reduced *if it is proper and $|\boldsymbol{b}_k^+| = \eta(\mathcal{C}(\boldsymbol{B}))$.*

Finally, we give an algorithm that finds a backward-reduced basis. See Algorithm 2.

▷ Claim 13. On input a proper basis $\boldsymbol{B}$, Algorithm 2 correctly outputs an invertible matrix $\boldsymbol{A}$ such that $\boldsymbol{AB}$ is backward reduced. Furthermore, Algorithm 2 runs in time $O(nk \log(q) \log(qn))$.

Proof. See the full version [21]. ◁

Algorithm 2 Backward Reduction.

```
Input: A proper basis $\boldsymbol{B} = (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k) \in \mathbb{F}_q^{k \times n}$ for $\mathcal{C}$
Output: An invertible matrix $\boldsymbol{A} \in \mathbb{F}_q^{k \times k}$ such that $\boldsymbol{AB}$ is backward reduced.
$\{j_1, \ldots, j_t\} \leftarrow \mathsf{MaxRedundantSet}(\boldsymbol{B})$
Let $m$ be minimal such that $B_{m,j_1} \neq 0$.
for $i \in [m+1, k]$ do
    $\boldsymbol{b}_i \leftarrow \boldsymbol{b}_i - B_{m,j_1}^{-1} B_{i,j_1} \boldsymbol{b}_m$
end
$(\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k) \leftarrow (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_{m-1}; \boldsymbol{b}_{m+1}; \ldots; \boldsymbol{b}_k; \boldsymbol{b}_m)$
return the matrix corresponding to the linear transformation done to $\boldsymbol{B}$.
```

5.2 Full backward reduction

Since backward reduction can be done efficiently, it is natural to ask what happens when we backward reduce many prefixes $\boldsymbol{B}_{[1,i]}$ of a basis. We could simply do this for all $i \in [k]$, but it is natural to be slightly more fine-grained and instead only do this for $i \leq \tau$ for some threshold τ. In particular, since the last $k - \mathrm{poly}(\log n)$ epipodal vectors tend to have length one even in very good bases (see the full version [21]to understand why), it is natural to take $\tau \leq \mathrm{poly}(\log n)$ to be quite small, which leads to very efficient algorithms. This suggests the following definition.

▶ **Definition 14.** *For some threshold $\tau \leq k$, a basis $\boldsymbol{B} \in \mathbb{F}_q^{k \times n}$ is* fully backward reduced up to τ *if it is proper and $\boldsymbol{B}_{[1,i]}$ is backward reduced for all $1 \leq i \leq \tau$.*

We now show how to easily and efficiently compute a fully backward-reduced basis, using the backward-reduction algorithm (Algorithm 2) that we developed above. We present the algorithm in Algorithm 3 and then prove its correctness and efficiency. Notice in particular that the algorithm only changes each prefix $\boldsymbol{B}_{[1,i]}$ (at most) once.

Algorithm 3 Full Backward Reduction.

```
Input: A proper basis $\boldsymbol{B} := (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k) \in \mathbb{F}_q^{k \times n}$ for a code $\mathcal{C}$ and a threshold
        $\tau \in [1, k]$
Output: A basis for $\mathcal{C}$ that is totally backward reduced up to $\tau$.
for $i = \tau, \ldots, 1$ do
    $\boldsymbol{A} \leftarrow \mathrm{BackwardReduction}(\boldsymbol{B}_{[1,i]})$
    $\boldsymbol{B} \leftarrow (\boldsymbol{A} \oplus \boldsymbol{I}_{k-i})\boldsymbol{B}$
end
return $\boldsymbol{B}$
```

▶ **Theorem 15.** *On input a proper basis $\boldsymbol{B} := (\boldsymbol{b}_1; \ldots; \boldsymbol{b}_k) \in \mathbb{F}_q^{k \times n}$ for a code $\mathcal{C}$ and a threshold $\tau \in [1, k]$, Algorithm 3 correctly outputs a basis $\boldsymbol{B}' \in \mathbb{F}_q^{k \times n}$ for $\mathcal{C}$ that is fully backward reduced up to τ. Furthermore, the algorithm runs in time $O(\tau^2 n \log(q) \log(qn))$.*

Proof. See the full version [21]. ◀

We next bound $|\boldsymbol{b}_1|$ of a fully backward-reduced basis. In fact, when $\tau \geq \lceil \log_q n \rceil$, this bound matches the Griesmer bound. In fact, it is not hard to see that with $\tau = k$, a fully backward-reduced basis is in fact LLL-reduced as well. But, the below theorem shows that we do not need to go all the way to $\tau = k$ to achieve the Griesmer bound. This is because in the worst case, $|\boldsymbol{b}_i^+| = 1$ for all $i \geq \log_q n$ anyway.

▶ **Theorem 16.** *For any positive integers k, $n \geq k$, and $\tau \leq k$, a basis $\boldsymbol{B} \in \mathbb{F}_q^{k \times n}$ of a code $\mathcal{C}$ that is fully backward reduced up to τ satisfies*

$$\sum_{i=1}^{\tau} \left\lceil \frac{|\boldsymbol{b}_1|}{q^{i-1}} \right\rceil \leq n - k + \tau \ .$$

Proof. See the full version [21]. ◀

5.3 Heuristic analysis suggesting better performance in practice

Recall that our analysis of backward-reduced bases in Section 5 relied crucially on the repitition number $\eta(\mathcal{C})$, which is the maximum over all bases of $\mathcal{C}$ of the last epipodal vector. We showed that $\eta(\mathcal{C})$ can be equivalently thought of as the maximal set of redundant coordinates. E.g., when $q = 2$, $\eta(\mathcal{C})$ is precisely the number of repeated columns in the basis $\boldsymbol{B}$ for $\mathcal{C}$.

Our analysis of fully backward-reduced bases then relies on the lower bound on $\eta(\mathcal{C})$ in Lemma 10. The proof of Lemma 10 simply applies the pigeonhole principle to the (normalized, non-zero) columns of a basis $\boldsymbol{B}$ for $\mathcal{C}$ to argue that, if there are enough columns, then one of them must be repeated many times. Of course, the pigeonhole principle is tight in general and it is therefore easy to see that this argument is tight in the worst case.

However, in the average case, this argument is not tight. For example, if the number n of (non-zero) columns in our basis $\boldsymbol{B} \in \mathbb{F}_2^{k \times n}$ is smaller than the number of possible (non-zero) columns $2^k - 1$, then it is certainly possible that no two columns will be identical. But, the birthday paradox tells us that even with just $n \approx 2^{k/2}$, a random matrix $\boldsymbol{B} \in \mathbb{F}_2^{k \times n}$ will typically have a repeated column. More generally, if a code $\mathcal{C}$ is generated by a random basis $\boldsymbol{B} \in \mathbb{F}_q^{k \times n}$, then we expect to have $\eta(\mathcal{C}) > 1$ with probability at least $1 - 1/\operatorname{poly}(n)$, provided that, say, $n \geq 10\log(n)q^{k/2}$, or equivalently, provided that

$$k \leq 2(\log_q n - \log_q(10\log(n))) \ .$$

We could now make a heuristic assumption that amounts to saying that the prefixes $\boldsymbol{B}_{[1,i]}$ behave like random matrices with suitable parameters (in terms of the presence of repeated non-zero columns). We could then use such a heuristic to show that we expect the output of Algorithm 3 to achieve

$$k_1 > (2 - o(1)) \log_q n \ .$$

We choose instead to present in the full version [21]a variant of Algorithm 3 that provably achieves the above. This variant is identical to Algorithm 3 except that instead of looking at all of $\boldsymbol{B}_{[1,i]}$ and choosing the largest set of redundant coordinates in order to properly backward reduce $\boldsymbol{B}_{[1,i]}$, the modified algorithm chooses the largest set of redundant coordinates *from some small subset* of all of the coordinates. In other words, the modified algorithm ignores information. Because the algorithm ignores this information, we are able to rigorously prove that the algorithm achieves $k_1 \gtrsim 2\log_q n$ when its input is a random matrix (by arguing that at each step the algorithm has sufficiently many fresh independent random coordinates to work with).

We think it is quite likely that Algorithm 3 performs better (and certainly not much worse) than this information-ignoring variant. We therefore view this as strong heuristic evidence that Algorithm 3 itself achieves $k_1 \gtrsim 2\log_q n$. (This heuristic is also confirmed by experiment. See the full version [21].)

5.3.1 Backward reducing without all of the columns

See the full version [21].

References

1 Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete Gaussian sampling. In *STOC*, 2015.

2 Divesh Aggarwal and Noah Stephens-Davidowitz. (Gap/S)ETH hardness of SVP. In *STOC*, 2018.

3 Miklós Ajtai. Generating hard instances of lattice problems. In *STOC*, 1996.

4 Miklós Ajtai. The Shortest Vector Problem in L2 is NP-hard for randomized reductions. In *STOC*, 1998.

5 Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC*, 1997.

6 Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the Shortest Lattice Vector Problem. In *STOC*, pages 601–610, 2001.

7 Michael Alekhnovich. More on average case vs approximation complexity. In *FOCS*, pages 298–307, 2003.

8 Nicolas Aragon, Julien Lavauzelle, and Matthieu Lequesne. decodingchallenge.org, 2019. URL: `http://decodingchallenge.org`.

9 Sanjeev Arora, László Babai, Jacques Stern, and Z. Sweedyk. The Hardness of Approximate Optima in Lattices, Codes, and Systems of Linear Equations. *J. Comput. Syst. Sci.*, 54(2):317–331, 1997.

10 L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

11 Shi Bai, Damien Stehlé, and Weiqiang Wen. Measuring, simulating and exploiting the head concavity phenomenon in BKZ. In *Asiacrypt*, 2018.

12 Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, 2016.

13 Huck Bennett, Alexander Golovnev, and Noah Stephens-Davidowitz. On the quantitative hardness of CVP. In *FOCS*, 2017.

14 E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

15 Peter van Emde Boas. Another NP-complete problem and the complexity of computing short vectors in a lattice. Technical report, University of Amsterdam, 1981.

16 Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Asiacrypt*, 2011.

17 Thomas Debris-Alazard, Léo Ducas, and Wessel P. J. van Woerden. An algorithmic reduction theory for binary codes: LLL and more. *IEEE Transactions on Information Theory*, 68(5):3426–3444, 2022. URL: `https://eprint.iacr.org/2020/869`.

18 Irit Dinur, Guy Kindler, Ran Raz, and Shmuel Safra. Approximating CVP to within almost-polynomial factors is NP-hard. *Combinatorica*, 23(2):205–243, 2003.

19 I. Dumer, D. Micciancio, and M. Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Transactions on Information Theory*, 49(1):22–37, 2003.

20 Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In *STOC*, 2008.

21 Surendra Ghentiyala and Noah Stephens-Davidowitz. More basis reduction for linear codes: backward reduction, BKZ, slide reduction, and more, 2024.

22 J. H. Griesmer. A bound for error-correcting codes. *IBM Journal of Research and Development*, 4(5):532–542, 1960.

23 Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.

24 P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *Eurocrypt*, 1988.

25 Arjen K. Lenstra, Hendrik W. Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, December 1982.

26 Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. DSN Progress Report, Jet Propulsion Laboratory, 1978.

27 Daniele Micciancio. The Shortest Vector Problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, 2001.

28 Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In *Eurocrypt*, 2016. URL: http://eprint.iacr.org/2015/1123.

29 Phong Q. Nguyen and Brigitte Vallée, editors. *The LLL Algorithm: Survey and Applications*. Springer-Verlag, 2010.

30 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):Art. 34, 40, 2009. doi:10.1145/1568318.1568324.

31 Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53(23):201–224, 1987.

32 Noah Stephens-Davidowitz and Vinod Vaikuntanathan. SETH-hardness of coding problems. In *FOCS*, 2019.

33 Alexander Vardy. Algorithmic complexity in coding theory and the Minimum Distance Problem. In *STOC*, 1997.

34 Michael Walter. Lattice blog reduction: The Simons Institute blog. https://blog.simons.berkeley.edu/2020/04/lattice-blog-reduction-part-i-bkz/, 2020.