Topics in Lattice Sieving

Eamonn W. Postlethwaite

Information Security Group
Royal Holloway, University of London

This dissertation is submitted for the degree of
Doctor of Philosophy

January 2022

This thesis is dedicated to the memory of Maurice, Sonia, and Walter.

Declaration

These doctoral studies were conducted under the supervision of Professor Martin R. Albrecht.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Information Security Group as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Eamonn W. Postlethwaite
January 2022

Acknowledgements

I must first acknowledge my supervisor Martin without whose patience, knowledge, and guidance this thesis would not have been possible. Your support gave me the confidence to ask questions and fail, and to view it as progress. Thank you Martin for making me a 'cryptographer of a sort'.

Other members of the cryptography community have my deep gratitude. Léo Ducas took a chance on, and encouraged, me as an exceptionally green and out of my depth new Ph.D student by inviting me to CWI to collaborate. Arnab Roy and Hart Montgomery patiently explained new cryptographic vistas to me at Fujitsu Research. Thomas Prest welcomed me to PQShield and introduced me to many interesting research projects.

I must also thank my coauthors for their good humour, incredible work ethic, and their ability to put up with such a pedant. Martin, Ben, Amit, Alex, Rachel, Fernando, Thomas, Vlad, John, Elena, Erik, Subhayan, Léo, Gottfried, Marc, Arnab, Charanjit, Rouz, Muslum, Attila, Keitaro, Shu, Thomas, and Bas, may you never have to hear about spaced en dashes, British English, or footnote placement ever again.

I would like to thank Professors Damien Stehlé and Carlos Cid for agreeing to be my examiners and for reading this thesis.

The Centre for Doctoral Training has offered me many opportunities in terms of travel, seminars, socials, training, and interesting whiteboard conversations. I would therefore like to thank all its students, academic staff, and administrative staff (especially Claire!) for the privilege.

No wealth compares to the warmth in the eyes of a friend, and I have been lucky enough over the course of my Ph.D to spend time with some wonderful humans. In no particular order, thanks to friends old and new: Will, Matt, Lucy, Felix, Finlay, Lex, Lukas, Rosemerry, Oscar, Edd, Austeja, Camilla, Sam, Hardy, Jamie, Proctor, Alex,

Greg, Sam, Edwin, Nick, Mo, Sam, Fernando, Feargus, Rob, Ashley, Jeroen, Marcel, Erin, Ben, Jodie, Lenka, Liam, Simon, Balázs, Joe, Amit, Lydia, Ben, Rachel, Alex, Nick, Tabby, Joe, Artem, Eoghan, Maya, Nick, Keele, Reynold, Sally, Trevyn, Marie-Sarah, Paul, Ester, Calum, Anya, Rebeca, Ralph, Robin, James, Thomas, Maxime, Gunjin, Qipeng, Navid, Joey, Clara, Koen, Pandora (and tribe), and many more. I would especially like to thank those who spent more time with me than was perhaps wise.

Finally, the support and steadfast belief my family have shown me during my studies has at times been the only thing keeping me going. To all my family, and especially Dad, Jo, Mum, and Gordon, thank you.

Abstract

In this thesis we discuss the design, cost, and use of algorithms called lattice sieves. These are a class of algorithms that iteratively obtain shorter lattice vectors by taking well chosen combinations from a large set of such. We focus on the cryptanalytical aspects of lattice sieves; their ability to find short lattice vectors allows them to be used in the cryptanalysis of several problems upon which post quantum cryptography is being built, for example the learning with errors problem. The threat model for post quantum cryptography involves the construction and use of large, fault tolerant, quantum computers. This necessitates the analysis of quantum variants of lattice sieves. These quantum variants may require less computation to find short lattice vectors than their classical counterparts, a topic two of the papers below examine. After giving preliminaries we present three publications, each concerned with a different aspect of lattice sieves.

In the first we codify and implement an abstract framework, the General Sieve Kernel, that collates many recent improvements in lattice sieving. This framework moves beyond the idea of using a lattice sieve as a 'one shot' shortest vector problem oracle, and instead imbues it with a state which is maintained between sieving operations. Using our implementation of this framework we are able to design and experiment with new lattice reduction procedures and ultimately solve previously unsolved instances of shortest vector and learning with errors type problems.

In the second we consider time memory trade offs for quantum variants of a large class of lattice sieves called k-sieves. These quantum variants require an exponential amount of qRAM, a type of quantum memory that allows quantum superposition access to classical data. This is an expensive resource, so we are concerned with the optimal time complexity that can be achieved when restricting memory. One sieving procedure we design has a time complexity moderately larger than the state of the art for quantum lattice sieves while requiring significantly less qRAM.

In the third we consider the non asymptotic speedups obtained by applying various quantum search algorithms to classical lattice sieves. In particular we design quantum circuits for these sieves and estimate their cost in a number of cost models. These cost models represent different assumptions about the cost of maintaining an error free quantum memory. In doing so we give a new heuristic analysis of various techniques used in practical lattice sieves and estimates for where quantum sieving will begin to outperform classical sieving under the various cost models.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Modern cryptography is full of statements like 'Y is hard if X is hard', and 'we build Z from Y'. But what are these mysterious X, Y, and Z? What does it mean for X or Y to be hard, or to build Z from Y?

To begin to answer the first of these questions, X and Y are *problems*. Simple examples might include

- given two integers, find their sum,
- given a finite list of real numbers, arrange them in ascending order,
- given an integer, find its prime factorisation,

and so on. These problems all share some structure. One receives an input (a pair of integers, a finite list, an integer) and is asked to perform some task, before giving an output of a particular form (a sum, an ordered list, a prime factorisation). This gives an intuitive idea of what a problem is.

What about a problem being hard? Let us start with the example of adding two integers together. Perhaps $6 + 7 = 13$ is easy, and perhaps $1417 + 2613 = 4030$ is easy too, but the latter is somehow harder. If one applies the addition method we learn as children, to add the ones and carry any tens, add the tens and carry any hundreds, and so forth, there is simply *more to do* when one has more digits to add. When we speak of the hardness of a problem, we are never really talking about a single instance of this problem, but rather how its hardness grows as a function of the input size. To attempt a little formalism, we will measure the input size of the addition problem as the maximum number of base 10 digits required to give either of the summands and call this n. In the $1417 + 2613 = 4030$ example $n = 4$, and if instead we had $1417 + 12613 = 14030$ we would have $n = 5$. We also define some basic operations to

which we assign one unit of cost. For example here we might say that adding two single digit integers together costs one unit of cost, as does calculating the carry, and as does reading a digit of input. In this simple computational model, with a little squinting, it becomes clear that one must read some number in $\{n+1, \dots, 2n\}$ of digits of input and perform no more than $3n$ addition and carry operations. We therefore end up with a cost function of something like

$$C_{\text{add}} \colon \{1, 2, \dots\} \to [0, \infty),\ C_{\text{add}}(n) \leq 5n.$$

Here the domain $\{1, 2, \dots\}$ of the cost function represents the length of the input, and the evaluation of the cost function represents the minimum cost of the addition. Note that we have the inequality because we have described (the add and carry method) only one procedure for addition. This procedure has the cost given as the upper bound. We have *not* shown that addition of two integers cannot be performed with a lower cost – though it cannot be too much lower, as to solve a general instance with input of size n one must sometimes read $2n$ digits. Let us therefore say that the cost of addition grows as something like $5n$ in the input size. This is not very fast! To double this cost we must double the size of the input. Imagining some world where cryptography depends on the 'hardness' of addition, if your preferred adversary doubles their computing power, you have to double to length of your inputs.

Perhaps unsurprisingly therefore, addition is not considered a hard problem. If we let C_X denote the cost function for some problem X, what kind of function should C_X be for us to consider X to be a hard problem? Some reasonable candidates might be

- $C_X(n) = 2^n$, exponential growth,
- $C_X(n) = n!$, factorial growth, otherwise known as total overkill,
- $C_X(n) = n^{\log n}$, something a little more subtle,

and so on. Why are these reasonable candidates? In the first example, to double the cost of the problem one need only increase the input size by one. The factorial in the second example grows even faster.

The final example contains a little subtlety, but is still sufficient. It is less simple to calculate how much one must increase n by to double the cost, and for any $\varepsilon > 0$ eventually $(1+\varepsilon)^n$ will dominate $n^{\log n}$. That is, the fraction $n^{\log n}/(1+\varepsilon)^n$ tends to 0, so $n^{\log n}$ is not an example of exponential growth. However it does grow faster than any polynomial in n; it is *superpolynomial*. In particular take any integer $d \geq 0$ and some positive real numbers $\alpha_0, \dots, \alpha_d$, and consider the polynomial $p(n) =$

$\alpha_0 + \alpha_1 n + \cdots + \alpha_d n^d$. With a little thought one can see that if we let $\log N = d + 1$, as soon as $n \geq N$ we have that $n^{\log n}$ is a polynomial of a higher degree than $p(n)$ and will therefore dominate it.

For various reasons, a problem whose cost grows superpolynomially in the size of its input is a definition of 'hard' which we find useful in cryptography. A problem that is not hard is said to be *easy*, and able to be solved *efficiently*. One reason for considering this definition of hard is that often we are not able to achieve the gold standard of statements introduced at the beginning: 'Y is hard if X is hard'. Instead we settle for 'Y is hard if a somewhat easier version of X is hard'. We call this 'somewhat easier' qualifier the *loss* of such a statement, and in cryptography it is often known quantitatively. Moreover, we often have *polynomial* loss, and so if the cost grows superpolynomially we can account, at least in theory, for the loss without needing to increase the input size too much.

Let us return to our three candidate problems. We have seen that addition is easy. It is left as an interesting Sunday afternoon to the reader to think about how to assign input size and what operations should be assigned unit cost in the case of ordering a set of reals, and to show that it is also easy.

We are therefore left with the final problem, which, despite also being simple to state, is different from the others. The final problem is somehow harder than the first two,[1] in that no one knows, ignoring some caveats to come below, how to do it efficiently. More specifically, if we let the input size be the number of bits of the integer being factorised, i.e. its length base 2, then every known procedure that returns the prime factors has a superpolynomial cost.

The fastest known algorithm is the General Number Field Sieve [LL93], implementations of which [Tea17] are to this day breaking factoring challenges [Zim20]. The General Number Field Sieve solves the factoring problem with asymptotic cost

$$C_{\text{factor}}(n) \in \exp\left(\left((64/9)^{1/3} + o(1)\right)(\log n)^{1/3}(\log\log n)^{2/3}\right),$$

which is a fancy way of saying that it is superpolynomial. Note though that proving a problem is hard is an entirely different kettle of fish to proving it is easy. To prove something is easy it is enough to exhibit a single procedure which has a cost that is less than superpolynomial. To prove something is hard one must show that no such procedure exists, and depending on the problem, this can be significantly more challenging. No one has proven that C_{factor} is superpolynomial, and perhaps it is not.

[1] Although, catch me on a slow day and they are all borderline impossible.

However, as understanding and finding prime factors is both a fundamental problem in pure mathematics, as well as something with huge real world importance (spoiler: factoring plays an important role in modern cryptography), it is a problem that has been considered in a great amount of depth for many years by some very clever people. It is what I like to call a 'problem with a pedigree'. This is one of the most interesting facets of cryptography, often when it is proven that 'Y is hard if X is hard', X is not some unconditionally hard problem, but merely a problem which is much better studied and understood than Y, and ultimately, one in which people have more faith.

If we return to the beginning again, we had a second statement, 'we build Z from Y'. Here Z is not a problem, but some cryptographic object. For example consider a public key encryption scheme. This is some cryptographic object where the public (people who are not me, and in possession of some public information) can encrypt a message such that only I can decrypt it by using some secret information known only to me. To say that Z is built from Y means that someone who is not me, and therefore not in possession of this secret information, must solve an instance of problem Y to decrypt a message, or more generally violate some other abstract security definitions. It should now be clear why we might want Y to be hard.

In general a cryptographic object comes with a semantic description of its inputs and ouputs, as well as security definitions. These definitions are formalisms of statements like 'someone who is not in possession of the secret information cannot decrypt'. It can be difficult enough to build cryptographic objects that rely on problems in this way, nevermind on problems that are well understood and thought to be hard. This is why we want a statement like 'Y is hard if X is hard'; if we can build Z from Y and then show the hardness of Y relies on the hardness of a much more understood or studied problem such as X, then we have a good basis for the security of Z.

1.1 Turing, his machines, and quantum computing

Alan Turing was a British mathematician, cryptanalyst, and a surprisingly good runner. During his studies at Cambridge he developed the idea of a universal computing machine that has come to be known as a Turing machine [Tur37], and in the process answered the 'Entscheidungsproblem' problem set by Hilbert and Ackermann in 1928 [HA28] in the negative. This problem asks whether there is an algorithm that takes as input any mathematical statement and outputs True or False. During the war these ideas on computing were brought to bear on the problem of breaking Axis naval ciphers, building on the crucial work of Polish cryptanalysts and French spies [Tur18, Bak18].

After the war he went on to design several early computers at the National Physical Laboratory and later at Manchester University.

The Churchill post war government prosecuted Turing for the 'gross indecency' of being a gay man. As his punishment he chose chemical castration rather than prison, and he died by cyanide poisoning in 1954.

In recent years attempts have been made to rectify, to some small extent, this national disgrace. In 2013 he received a posthumous pardon, and in 2017 a law was passed by the British parliament that has come to be known as 'Turing's Law'.[2] This law granted a blanket pardon to all gay men for their historical 'offences'. Turing's portrait now adorns the highest denomination bank note printed by the Bank of England (£50), and he is venerated as a national hero. Nonetheless, we should not attempt to separate his treatment as a human being from his rarefied academic achievements, or forget the lessons it can teach us in the present day.

Leaving behind the horrors of history, Turing machines offer a method to formalise the above discussion of problem, cost, and procedure. The full definition and details of Turing machines are not relevant for this introduction, but rather the effect their description had on ideas of computability. Intuitively a Turing machine is formed of an infinitely long one dimensional tape which can be read from and written to by a magnetic head. This tape is separated into cells, each of which contains some symbol from a finite alphabet. The machine has some state, and at the beginning of each step of computation the head reads the symbol from the current cell on the tape. Depending on this read and the current state of the machine, it may erase or overwrite the symbol in the cell, it moves the head either left or right, and it alters its internal state. There is a state called the halting state such that, if the machine ever attains this state, it halts. When a machine halts it has finished its computation.

The complexity of solving a problem on a Turing machine can then be thought of as the number of steps of computation before it halts. What about this physically impossible infinite tape stretching to the horizon in either direction? This represents a cost that we have not yet discussed, the memory cost required to solve a problem. This infinite tape exists only to allow the Turing machine however much memory it requires for its computation. There is a special symbol in the language of the machine which is blank, and this blank symbol is the only one allowed to appear infinitely often on the tape. We call the memory requirement of the Turing machine the largest number of non blank cells on the tape at any step in the computation before it halts. The

[2] `https://www.legislation.gov.uk/ukpga/2017/3/part/9/chapter/1/crossheading/pardons-for-certain-abolished-offences-etc/enacted`.

problems we considered above do not require much memory, but for many considered in this thesis the memory cost will be exponential in the size of the input.

Besides offering rigorous formalism to ideas of algorithms and computability, Turing machines led to an informal statement that has come to be known as the Church–Turing thesis, named with another early pioneer in the theory of computation, Alonzo Church. It can be stated as

Any algorithmic process can be simulated efficiently using a Turing machine.

This is not a formal statement without a definition of 'algorithmic process', but for some definition of algorithmic process for which the above statement is true, the Church–Turing theorem says that no matter what kind of machine we use to perform an algorithm (an abacus, a Babbage engine, a desktop computer, and so forth), we can simulate it efficiently with a Turing machine. In words, 'simulate[d] efficiently' means that any problem that has an efficient solution on e.g. a desktop computer also has an efficient solution on a Turing machine. This means that we need only consider computational costs in the Turing machine model.[3]

It turns out that the above statement of the Church–Turing theorem is not quite strong enough. In the 1970s Solovay and Strassen developed an efficient probabilistic primality test [SS77] for which the use of randomness was essential. Shortly thereafter many algorithms that make fundamental use of randomness were developed, and in many cases no efficient deterministic (i.e. algorithms that do not make use of randomness) equivalent algorithms were known.[4] This leads to the definition of a probabilitistic Turing machine – this is a Turing machine that has a further *random* tape. This is an infinite tape of uniformly distributed bits which can be read from and used to determine actions and state changes in a computation step. For example, if the Turing machine has a particular state and reads a particular symbol from its main tape, it may read a bit from its random tape and move its head left if the bit is 0 and right if the bit is 1. The amount of randomness read from this tape is also considered a cost, alongside steps of computation and memory. We therefore arrive at an adapted version of the Church–Turing thesis.

Any algorithmic process can be simulated efficiently using a probabilistic Turing machine.

[3]For the avoidance of doubt, we never do this. Instead we focus on complexity in asymptotic and circuit based models.

[4]We note that this is no longer the case for primality testing [AKS04].

Having had to alter the statement of this theorem once, why not again? Thoughts like this led Benioff [Ben82], Feynman [Fey82], Deutsch [Deu85], and others in the 1980s to consider the existence of a computational device that could efficiently simulate any physical system. Such a computational device, let us call it `Q`, would be a good candidate for a further strengthening of the Church–Turing theorem,

Any algorithmic process can be simulated efficiently using `Q`.

No prizes for guessing that `Q` stands for 'quantum', which were the physical laws upon which these authors tried building a computing machine. Thus was born the notion of a quantum computer.

The functional importance of the above is that it leaves open the question 'can a quantum computer do things that even a probabilistic Turing machine cannot?', or put more precisely, 'are there algorithmic processes that a quantum computer can perform efficiently, but a probabilistic Turing machine cannot?'. One candidate for such an algorithmic process is the reason this thesis exists.

1.2 Wherefore this thesis

In 1994 Shor gave a quantum algorithm [Sho94] that, to this day, efficiently solves problems not thought to have efficient solutions on probabilistic Turing machines. A quantum algorithm is one that runs on a quantum computer, and as discussed above we model a classical computer by a probabilistic Turing machine. Shor's algorithm can solve a fairly general class of problems, but it is best known for solving a particular few. One such problem is the factorisation of integers into their prime factors, a problem we saw earlier and for which the best known classical algorithm has superpolynomial complexity.

As well as being an enormous theoretical breakthrough, factoring has an important place in modern cryptography. If quantum computers are built that can run Shor's algorithm, whole swathes of modern cryptography (e.g. public key encryption schemes) become insecure. This is not helped by the fact that another problem solved efficiently by Shor's algorithm is the discrete logarithm problem, which will not be discussed in this introduction, but plays a similarly fundamental role in modern cryptography.

Another important quantum algorithm is Grover's quantum mechanical search algorithm [Gro96]. This is an algorithm that square roots the complexity of finding a marked element in an unstructured list. More concretely let us say we have some list of N elements which has no structure (or at least, we know nothing about its structure),

and exactly one element has some special property, or is 'marked'. Using a classical computer we expect to check approximately $N/2$ elements of our list for this property before finding the one that has it. Using a quantum computer the same task can be performed in approximately $\sqrt{N}$ checks, for some appropriately altered definition of a check. Moreover these complexities are known to be optimal in both the classical and quantum cases. Note that this does not immediately mandate an upgrade to the Church–Turing thesis, as the square root of a polynomial function is polynomial, and similarly the square root of a superpolynomial function is superpolynomial. That is, Grover's search algorithm does not give an efficient quantum algorithm where only an inefficient classical algorithm exists. However, the ability to search more quickly for marked elements in lists meaningfully improves several cryptanalytic algorithms, and throughout this thesis Grover's algorithm will be both examined and used to design new algorithms.

These quantum threats to cryptography have led to a search for new problems Y from which to build cryptographic objects Z. These problems Y need to be *post quantum*, that is, problems for which no efficient solution exists even given the use of a large scale, fault tolerant quantum computer. In essence, such a quantum computer is one that can run long computations without errors accumulating and invalidating the output. Often these post quantum Y, or the problems X for which we have the statement 'Y is hard if X is hard', have a little less pedigree than factoring integers or finding a discrete logarithm.

Several standards bodies around the world, perhaps most notably NIST (the National Institute for Standards and Technology of the United States), are far into the process of standardising post quantum cryptography [NIS17]. Large technology companies such as Google [Mat16] and Cloudflare [Kri19] are also already trialling post quantum cryptography for internet protocols, to smooth its eventual passage into general usage. Submissions to the NIST process have been solicited from academia and industry, and cryptography built from several classes of candidate post quantum problems is currently under examination. One of these candidate classes is the class of *lattice problems*, such as the learning with errors problem (LWE), or the NTRU problem.[5]

One way to solve these problems is to search for short vectors in mathematical objects called lattices; one way to perform this search is to use an algorithm called a lattice sieve; and one way to understand the effect lattice sieves have on post quantum cryptography is to further their design and analysis. This *finally* is the topic of this

[5]My favourite possibility for this acronym is 'Number Theorists R Us' [Why05].

thesis. It intends to add a smidgen of pedigree to the problem of solving lattice problems via finding short vectors in lattices.

1.3 Contents of this thesis

This thesis contains amended and annotated versions of the following three publications.

1. Albrecht M.R., Ducas L., Herold G., Kirshanova E., Postlethwaite E.W., Stevens M. (2019) The General Sieve Kernel and New Records in Lattice Reduction. In: Ishai Y., Rijmen V. (eds) Advances in Cryptology – EUROCRYPT 2019. EUROCRYPT 2019. Lecture Notes in Computer Science, vol 11477. Springer, Cham.

2. Kirshanova E., Mårtensson E., Postlethwaite E.W., Roy Moulik S. (2019) Quantum Algorithms for the Approximate k-List Problem and Their Application to Lattice Sieving. In: Galbraith S., Moriai S. (eds) Advances in Cryptology – ASIACRYPT 2019. ASIACRYPT 2019. Lecture Notes in Computer Science, vol 11921. Springer, Cham.

3. Albrecht M.R., Gheorghiu V., Postlethwaite E.W., Schanck J.M. (2020) Estimating Quantum Speedups for Lattice Sieves. In: Moriai S., Wang H. (eds) Advances in Cryptology – ASIACRYPT 2020. ASIACRYPT 2020. Lecture Notes in Computer Science, vol 12492. Springer, Cham.

> Throughout the above papers this boxed environment will occasionally appear. It will contain (hopefully) helpful notes or further working that I believe would damage the prose as is. As such these boxes can be safely ignored, unless further details are desired.

The following publications were also written during my studies at Royal Holloway.

1. Albrecht M.R., Curtis B.R., Deo A., Davidson A., Player R., Postlethwaite, E.W., Virdia F., Wunderer T. (2018) Estimate All the {LWE, NTRU} Schemes!. In: Catalano D., De Prisco R. (eds) Security and Cryptography for Networks. SCN 2018. Lecture Notes in Computer Science, vol 11035. Springer, Cham.

2. Behnia R., Postlethwaite E.W., Ozmen M.O., Yavuz A.A. (2020) Lattice-Based Proof-of-Work for Post-Quantum Blockchains. *Accepted at CBT 2021.*

3. Postlethwaite E.W., Virdia F. (2021) On the Success Probability of Solving Unique SVP via BKZ. In: Garay J.A. (eds) Public-Key Cryptography – PKC 2021. PKC 2021. Lecture Notes in Computer Science, vol 12710. Springer, Cham.

4. Hashimoto H., Katsumata S., Postlethwaite E.W., Prest T., Westerbaan B. (2021) A Concrete Treatment of Efficient Continuous Group Key Agreement via Multi-Recipient PKEs. *Accepted at ACM CCS 2021.*

The final paper was written while I was undertaking an internship at PQShield in Oxford, UK, under the pedagogical flair of Thomas Prest.

My research was supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/P009301/1).

1.4 Organisation of this thesis

In Chapter 2 we introduce the notions and mathematical concepts required for this thesis. There is a focus on lattices, lattice reduction algorithms, lattice sieves, and an introduction to quantum notation and quantum search.

In Chapter 3 we present the first paper of this thesis [ADH$^+$19a]. The focus of this paper is how one can use a lattice sieve as the fundamental operation in a stateful machine we call the General Sieving Kernel. Taking this view of lattice sieves allows us to design new algorithmic procedures for lattice reduction tasks, and ultimately for finding short vectors in lattices. We marry this new algorithmic freedom with techniques from the literature and a large implementation effort to solve lattice challenges in higher dimensions than had been previously achieved.

In Chapter 4 we present the second paper of this thesis [KMPM19a]. The focus of this paper is exploring the asymptotic time memory trade offs available to us when considering quantum variants of previously classical lattice sieve designs. Such trade offs are especially important for quantum lattice sieves due to the exponential memory they require needing to be quantumly accessible. This requires a potentially expensive resource called qRAM (quantum random access memory). We develop tailored quantum search routines based on Grover's algorithm and its generalisations to give smooth time memory trade offs for these lattice sieves.

In Chapter 5 we present the final paper of this thesis [AGPS20]. The focus of this paper is the effect that different cost models have on the feasibility and magnitude of complexity improvements given by quantum lattice sieves. Whereas in the above chapter we work asymptotically, where the differences between these cost models disappear, here we focus on concrete quantum circuits for certain lattice sieve tasks. We examine three lattice sieves under several cost models, including one that takes into account the cost of maintaining error free quantum states, and provide software and estimates for their performance.

Chapter 2

Preliminaries

Here we introduce the relevant generic technical ideas and notations used throughout this thesis.

2.1 Basic notation

We use $\mathbb{N} = \{0, 1, \dots\}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ to denote the natural numbers, integers, rationals reals, and complex numbers, respectively. We may write e.g. $\mathbb{R}^+ = \{x \colon x \in \mathbb{R},\, x > 0\}$ to denote the strictly positive entries of such a set, when this is well defined. For $n \in \mathbb{N}$ let $[n]_0 = \{0, 1, \dots, n\}$ and for $n \in \mathbb{N} \setminus \{0\}$ let $[n] = \{1, \dots, n\}$. For a function $f \colon X \to Y$ we denote the image of f as $\mathsf{Im}\, f = \{f(x) \colon x \in X\}$. We denote the absolute value of a real as $|\cdot| : \mathbb{R} \to [0, \infty), x \mapsto x$ if $x \geq 0$, and $x \mapsto -x$ otherwise. For a set X we let $|X|$ denote its cardinality and id_X the identity function on it. By $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ we denote the usual floor and ceiling operators. We denote the rounding function $\lceil \cdot \rfloor \colon \mathbb{R} \to \mathbb{Z}$ which rounds x to the integer y that minimises $|x - y|$. In the case of ties, i.e. $x = n + 1/2$ for $n \in \mathbb{Z}$, we define $\lceil x \rfloor = \lceil n + 1/2 \rfloor = n + 1$. Throughout log is the natural logarithm, and $\log_2$ is the logarithm base 2.

Vectors, norms, angles and subsets of $\mathbb{R}^d$

Let $\mathbb{R}^d$ denote Euclidean space of dimension d and let $\mathbf{e}_i$ represent the i^{th} canonical basis vector. Denote $\mathbf{x} \in \mathbb{R}^d$ as $\mathbf{x} = (x_1, \dots, x_d)^t$, where t denotes transposition. Define the inner product $\langle \cdot, \cdot \rangle : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, (\mathbf{x}, \mathbf{y}) \mapsto \sum_i x_i \cdot y_i$. The inner product implies the Euclidean norm $\|\mathbf{x}\| = \langle \mathbf{x}, \mathbf{x} \rangle^{1/2}$. Denote, for $\arccos \colon [-1, 1] \to [0, \pi]$, the angle

between two vectors as

$$\theta(\mathbf{x}, \mathbf{y}) = \arccos\left(\frac{\langle \mathbf{x}, \mathbf{y}\rangle}{\|\mathbf{x}\|\,\|\mathbf{y}\|}\right).$$

Let

$$B_d(r; \mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^d \colon \|\mathbf{x} - \mathbf{y}\| < r\}, \qquad \bar{B}_d(r; \mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^d \colon \|\mathbf{x} - \mathbf{y}\| \leq r\},$$

denote the open and closed balls of radius r around $\mathbf{x}$, respectively. If $\mathbf{x}$ is omitted it is read as $\mathbf{0}$. Let $S^{d-1} = \{\mathbf{x} \in \mathbb{R}^d \colon \|\mathbf{x}\| = 1\}$.

For a ring R and a given R-module M, we define the span under R of a countable set $S \subset M$ as

$$\operatorname{span}_R(S) = \left\{ \sum_{i \in I \subset \mathbb{N}} \alpha_i \cdot s_i \colon \alpha_i \in R, s_i \in S \right\},$$

where we usually take $M = \mathbb{R}^d$ and $R = \mathbb{Z}$ or $R = \mathbb{R}$. We say S is formed of linearly independent vectors if there does not exist a sequence $(\alpha_i)_{i \in I}$ different from $(0)_{i \in I}$ such that $\sum_{i \in I} \alpha_i \cdot s_i = 0$.

Matrices

Let $\mathbf{I}_n(R)$ denote the $n \times n$ identity matrix over ring R, which will be left implicit. Let $\mathrm{M}_{n \times m}(R)$ be the set of $n \times m$ matrices over the ring R, if $n = m$ we write $\mathrm{M}_n(R)$. Let $\mathbb{1}_n(R) \in \mathrm{M}_{n \times 1}(R)$ denote the all ones vector, for which we write $\mathbb{1}$ and assume the dimension and ring are clear from context. We let the determinant be the unique function $\det \colon \mathrm{M}_n(R) \to R$ such that it is multilinear, alternating, and $\det(\mathbf{I}_n) = 1$. The general linear group $(\mathrm{GL}_n(R), \cdot)$ is the group under matrix multiplication of all $\mathbf{A} \in \mathrm{M}_n(R)$ such that $\mathbf{A}$ is invertible over R, or equivalently that the determinant of $\mathbf{A}$ is a unit of R, i.e. $\det(\mathbf{A}) \in R^\times$. The special linear group $(\mathrm{SL}_n(R), \cdot)$ is a normal subgroup of the general linear group, in particular the kernel of the homomorphism $\det \colon \mathrm{GL}_n(R) \to R^\times$, so that $\mathbf{A} \in \mathrm{SL}_n(R)$ has $\det(\mathbf{A}) = 1_R$. We write the inverse of $\mathbf{A}$ as $\mathbf{A}^{-1}$ and $\mathbf{A}^{-1} \cdot \mathbf{A} = \mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}_n$. For any free R-module M of rank n, usually $M = R^n$ in our case, the transform $\mathbf{A} \colon M \to M, m \mapsto \mathbf{A} \cdot m$ for $\mathbf{A} \in \mathrm{GL}_n(R)$ is an automorphism of M. We note the special case $R = \mathbb{Z}$ where $R^\times = \{-1, 1\}$ and so the elements of $\mathrm{GL}_n(\mathbb{Z})$ are *unimodular*, i.e. their determinants are in $\{-1, 1\}$.We recall that the determinant is multiplicative, and for $\mathbf{A} \in \mathrm{GL}_n(R)$ we have $\det(\mathbf{A}^{-1}) = 1/\det(\mathbf{A})$.

For a ring R and a matrix $\mathbf{B} \in \mathrm{M}_{d \times n}(R)$ we usually write $\mathbf{B} \in R^{d \times n}$. We write $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n)$ as its column vectors, and let $\mathbf{B}[i] = \mathbf{b}_i$ and $\mathbf{B}[i : j] = (\mathbf{b}_i \cdots \mathbf{b}_{j-1})$. If $\mathbf{b}_i =$

$(b_1^i, \ldots, b_d^i)^t$ then let $\mathbf{B}[i,j] = b_j^i$. If $\mathbf{B} \in \mathbb{R}^{d\times n}$ then we define $\|\mathbf{B}\| = \max\{\|\mathbf{b}_i\| : i \in [n]\}$.

Measure and topology

We consider Lebesgue measurable sets in $\mathbb{R}^d$ and denote the measure of some such $S \subset \mathbb{R}^d$ as $\mathrm{vol}(S)$. Sometimes we prefer to refer to the specific measure and denote by L^d the Lebesgue measure on $\mathbb{R}^d$ and write $L^d(S)$ in place of $\mathrm{vol}(S)$. We denote L^1 by L. If X is a topological space and $S \subset X$, then the interior of S is defined as the union of all open sets of X contained as subsets of S. In Euclidean space this is equivalent to all points $\mathbf{s} \in S \subset \mathbb{R}^d$ such that there exists a radius $r_\mathbf{s} > 0$ with $B_d(r_\mathbf{s}; \mathbf{s}) \subset S$. If X is a topological space and $Y \subset X$ is equipped with the subspace topology, then we say Y is compact if every open cover of Y has a finite subcover. In Euclidean space this is equivalent to Y being closed and bounded, e.g. $\bar{B}_d(r; \mathbf{x}) \subset \mathbb{R}^d$. A convex set is a subset X of a vector space V over an ordered field F such that for all $x, y \in X$ the line segment $l_{x,y} = \{(1-t)\cdot x + t \cdot y \colon t \in [0,1] \cap F\}$ that joins x and y has $l_{x,y} \subset X$. A convex function is a real valued function f whose epigraph (the points 'above the curve') is a convex set. A real valued function f is concave if and only if $-f$ is convex. For a subset X of a vector space V over a field F we define $\alpha X = \{\alpha \cdot x \colon x \in X\}$ for any $\alpha \in F$. A subset X of a vector space V is said to be symmetric around 0, the additive identity of the vector space, if for all $x \in X$ we have $X = (-1)\cdot X$. Here -1 is the additive inverse of the scalar multiplicative identity of V.

The function $\Gamma(z) = \int_0^\infty x^{z-1}e^{-x}\,\mathrm{d}x$ for $z \in \mathbb{C}$ with positive real part is the classical gamma function. The volume of an open or closed ball of radius r in $\mathbb{R}^d$ is given by

$$\mathrm{vol}(B_d(r;\mathbf{x})) = \mathrm{vol}(\bar{B}_d(r;\mathbf{x})) = \frac{\pi^{d/2}}{\Gamma(1+d/2)} \cdot r^d,$$

for any $\mathbf{x} \in \mathbb{R}^d$, and we define $V_d(r) = \mathrm{vol}(B_d(r))$. In particular we define $v_d = V_d(1)$ as the volume of the unit ball in dimension d.

Probability

Let (Ω, F, P) be a probability space, so that Ω is a non empty set, $F \subset 2^\Omega$ is a σ-algebra of the power set of Ω, and $P \colon F \to [0,1]$ is a probability measure.

We consider random variables $X \colon \Omega \to S$, which are measurable functions between the measure spaces (Ω, F) and $(S, \mathscr{B}(S))$ where $\mathscr{B}(S)$ denotes the Borel σ-algebra of the topological space S. To concretise a little, we may think of e.g. $S = \mathbb{Z}_q^{n\times m}$,

the finite set of all n by m matrices over the ring $\mathbb{Z}_q$. Since S is countable, if we equip S with the discrete topology then $\mathscr{B}(S) = 2^S$; the Borel σ-algebra is just the power set of S. The random variable X is then any map from Ω to S such that $X^{-1}(A) = \{\omega \in \Omega \colon X(\omega) \in A\}$ is measurable in Ω, i.e. $X^{-1}(A) \in F$, for all $A \in \mathscr{B}(S)$. If the image of X is countable (as in the example above) we call it a discrete random variable, else it is a continuous random variable. For $A \in \mathscr{B}(S)$, i.e. the measurable $A \subset S$, we define $\Pr[X \in A] = P(\{\omega \in \Omega \colon X(\omega) \in A\})$. In particular for $\{a\} \in \mathscr{B}(S)$ we write $\Pr[X = a] = P(\{\omega \in \Omega \colon X(\omega) = a\})$.

If $S = \mathbb{R}$ then we call $X \colon \Omega \to \mathbb{R}$ a real valued random variable. A real valued random variable defines a cumulative distribution function (cdf) $F_X \colon \mathbb{R} \to [0, 1]$ as $F_X(a) = \Pr[X \in (-\infty, a]]$, or $\Pr[X \leq a]$ for short. For a discrete real valued random variable X, F_X uniquely defines a probability mass function (pmf),

$$f_X \colon \mathsf{Im}\, X \to [0, 1],\ x \mapsto \Pr[X \leq x] - \Pr[X < x].$$

The pmf f_X has the property that for any measurable A, $\Pr[X \in A] = \sum_{x \in A \cap \mathsf{Im}\, X} f_X(x)$. For a continuous real valued random variable X, F_X does not uniquely define a probability density function (pdf), the continuous analogue of a pmf. Instead we call any function $f_X \colon \mathbb{R} \to \mathbb{R}$ a pdf, or density, for X when

$$\Pr[X \in A] = \int_A f_X(x)\ \mathrm{d}L,$$

for all measurable A.

Let $X \colon \Omega \to S$ be a random variable, then we write $x \leftarrow X$ to mean a sample taken from $\mathsf{Im}\, X$ with probability $\Pr[X = x]$. A collection $X_1, \ldots, X_n \colon \Omega \to S$ of random variables are said to be i.i.d. if they are pairwise independent and for all measurable A they have $\Pr[X_1 \in A] = \cdots = \Pr[X_n \in A]$. We will often refer to samples from a given random variable as being i.i.d. If these samples are $x_1, \ldots, x_n$ then this shorthand means that each was sampled as $x_i \leftarrow X_i$ for i.i.d. $X_1, \ldots, X_n$. We denote by $\mathsf{U}(S)$ the uniform random variable over a finite set S. We denote by $\mathsf{N}(\mu, \sigma^2)$ the normal random variable with mean μ and variance σ^2. We may write e.g. $X \sim \mathsf{N}(\mu, \sigma^2)$ to denote that X is a random variable distributed as $\mathsf{N}(\mu, \sigma^2)$. Let $\mathbb{E}[X]$ and $\mathbb{V}[X]$ denote the expectation and variance of a random variable X.

We may not always abide by the notational conventions above for probability. For example if we fix some $n \in \mathbb{Z}^+$ and an $a \in [n]$ then

$$\Pr_{b \leftarrow \mathsf{U}([n])}[b > a] = 1 - a/n$$

represents the probability that a uniformly sampled b from $[n]$ is larger than some fixed $a \in [n]$. One could define $\Omega = [n]$ and an indicator random variable $X_a \colon \Omega \to \{0, 1\}$ per a such that $X_a(b) = 1$ if $b > a$ and $X_a(b) = 0$ otherwise. Given X_a then the above is equivalent to $\Pr[X_a = 1]$, but (I think) more readable.

Asymptotic notation

We define the Bachmann–Landau asymptotic notation following [Knu97, CLRS09]. Let f and g be real valued functions defined on some unbounded subset of the positive real numbers $S \subset \mathbb{R}^+$. Then we define

- $f \in O(g) \Leftrightarrow \exists\, M > 0, s_0 \in S$ such that for all $s \geq s_0$ we have $f(s) \leq M \cdot |g(s)|$, i.e. f is eventually bounded above by a constant multiple of g,
- $f \in o(g) \Leftrightarrow$ for all $M > 0, \exists\, s_0 \in S$ such that for $s \geq s_0$ we have $|f(s)| < M \cdot |g(s)|$, i.e. f is eventually dominated by g,
- $f \in \Omega(g) \Leftrightarrow g \in O(f)$, i.e. f is eventually bounded below by a constant mutiple of g,
- $f \in \omega(g) \Leftrightarrow g \in o(f)$, i.e. f eventually dominates g,
- $f \in \Theta(g) \Leftrightarrow f \in O(g)$ and $f \in \Omega(g)$, i.e. f is eventually bounded above and below by (different) constant multiples of g. Equivalently, $g \in \Theta(f)$.

We also define a class $\mathrm{poly}(n)$ of functions. If S is any unbounded subset of $\mathbb{N}$ then $f \colon S \to \mathbb{N}$ is such that $f(n) \in \mathrm{poly}(n)$ if there exists an $s_0 \in S$ and $p(x) \in \mathbb{Z}[x]$ such that for all $s \geq s_0$ we have $|f(s)| \leq p(s)$.

A function $f \colon \mathbb{N} \to \mathbb{R}$ is negligible (in n) if for any $c \in \mathbb{N}$ there exists an $N_c \in \mathbb{N}$ such that for all $n \geq N_c$ we have $f(n) \leq 1/n^c$. Let $f(n)$ be the probability of the n^{th} event occurring in some sequence of events parameterised by n. The probability of this sequence is said to be negligible if f is negligible. The probability of this sequence is said to be overwhelming if the probability of them not happening, i.e. $1 - f(n)$, is negligible.

Quantum notation

For some $\mathbf{U} \in \mathbb{C}^{n\times n}$ we denote by $\mathbf{U}^\dagger$ the conjugate transpose, i.e. $\mathbf{U}^\dagger[i,j] = \overline{\mathbf{U}[j,i]}$. We also, in the relevant sections, make use of braket notation. We work over the Hilbert space $\mathbb{C}^n$ and let $|\psi\rangle$ represent a column vector and $\langle\varphi|$ represent a row vector. Then $\langle\varphi|\psi\rangle$ and $|\psi\rangle\langle\varphi|$ are the standard Hermitian inner and outer product, respectively. Concretely

$$\langle\varphi|\psi\rangle = \varphi^\dagger\psi \in \mathbb{C}, \quad |\psi\rangle\langle\varphi| = \begin{pmatrix}\psi_1\\ \vdots \\ \psi_n\end{pmatrix}\begin{pmatrix}\overline{\varphi_1} & \cdots & \overline{\varphi_n}\end{pmatrix} \in \mathbb{C}^{n\times n}.$$

Miscellaneous

When considering elements of the ring $\mathbb{Z}_q$ we will sometimes take a representative element of them in $\mathbb{Z}$. For even q we call $\{-q/2, \ldots, q/2-1\}$ the balanced representation of $\mathbb{Z}_q$ in $\mathbb{Z}$, and similarly $\{-(q-1)/2, \ldots, (q-1)/2\}$ for odd q.

For $n \in \mathbb{Z}^+$ we let S_n denote the permutation group on n points.

2.2 Lattices

In this thesis we consider Euclidean lattices, these are discrete additive subgroups of a Euclidean space. In particular we work with the Euclidean space $\mathbb{R}^d$, see e.g. [Cas97, Sie89] for a thorough introduction.

Definition 2.2.1 (Euclidean lattices in $\mathbb{R}^d$)**.** A lattice is a discrete additive subgroup $(\Lambda, +) < \left(\mathbb{R}^d, +\right)$, that is, it is an additive subgroup and there exists a neighbourhood U of the identity $\mathbf{0}$ such that $U \cap \Lambda = \{\mathbf{0}\}$.

We drop the group theoretic notation from this point on. The Euclidean distance on $\mathbb{R}^d$ allows us to consider it as a metric space, and so we may take this neighbourhood U as some open Euclidean ball with positive radius. An equivalent way [Cas97, III.4] to define a lattice Λ is as the integer span of a basis.

Definition 2.2.2 (Lattice basis)**.** A set of vectors $B \subset \mathbb{R}^d$ is a basis for Λ if $\mathrm{span}_{\mathbb{Z}}(B) = \Lambda$ and B is formed of linearly independent vectors. We say that B generates Λ.

Definition 2.2.3 (Lattice rank and dimension)**.** If B of size n is a basis for $\Lambda \subset \mathbb{R}^d$ we call d the dimension of Λ and n the rank of Λ. If $n = d$ we call the lattice full rank.

When working with a basis we may write the vectors of B as the column vectors of some matrix $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n) \in \mathbb{R}^{d \times n}$.[1] We make heavy use of matrix notation for bases, and where clear from context switch between the set B and matrix $\mathbf{B}$ at will. We will use the terminology of Definition 2.2.3 also for B and $\mathbf{B}$.

Definition 2.2.4 (Matrix form)**.** The lattice generated by a set of linearly independent vectors $B = \{\mathbf{b}_i\}_{i=1}^n$ is

$$\Lambda = \operatorname{span}_{\mathbb{Z}}(B) = \left\{ \sum_{i=1}^{n} v_i \cdot \mathbf{b}_i \colon v_i \in \mathbb{Z} \right\} = \{ \mathbf{B} \cdot \mathbf{v} \colon \mathbf{v} \in \mathbb{Z}^n \},$$

for $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n) \in \mathbb{R}^{d \times n}$. We may write $\Lambda(\mathbf{B})$ to define this lattice.

For every lattice $\Lambda = \Lambda(\mathbf{B})$ of dimension d and rank greater than one, there are a countably infinite number of bases that generate Λ. Indeed for any $\mathbf{U} \in \mathrm{GL}_n(\mathbb{Z})$, a *unimodular* matrix, $\Lambda(\mathbf{B}) = \Lambda(\mathbf{B} \cdot \mathbf{U})$, and $|\mathrm{GL}_n(\mathbb{Z})|$ is countably infinite for $n \geq 2$. For example consider the integer matrix $\Lambda = \mathbb{Z}^2$, one possible basis is $\mathbf{B} = \mathbf{I}_2$, but it also has the following bases

$$\mathbf{B} \cdot \mathbf{U} = \mathbf{U}, \text{ where } \mathbf{U} = \begin{pmatrix} 1 & i \\ 0 & 1 \end{pmatrix},$$

for all $i \in \mathbb{Z}$. Each such $\mathbf{U} \in \mathbb{Z}^{2\times 2}$ and has $\det(\mathbf{U}) = 1$ and one can easily check that for any $v_1, v_2 \in \mathbb{Z}$

$$\mathbf{B} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \mathbf{B} \cdot \mathbf{U} \cdot \left(\mathbf{U}^{-1} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \right) = \mathbf{B} \cdot \mathbf{U} \cdot \begin{pmatrix} v_1 - iv_2 \\ v_2 \end{pmatrix},$$

so that $\Lambda(\mathbf{B}) \subset \Lambda(\mathbf{B} \cdot \mathbf{U})$, and

$$\mathbf{B} \cdot \mathbf{U} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \mathbf{B} \cdot \begin{pmatrix} v_1 + iv_2 \\ v_2 \end{pmatrix},$$

for the reverse inclusion. In general, $\mathbf{U}$ being unimodular is equivalent to it being invertible over the integers, and therefore the maps $\mathbf{v} \mapsto \mathbf{U} \cdot \mathbf{v}$ and $\mathbf{v} \mapsto \mathbf{U}^{-1} \cdot \mathbf{v}$ are bijections of $\mathbb{Z}^n$. For any two bases $\mathbf{B}, \mathbf{C}$ such that $\Lambda(\mathbf{B}) = \Lambda(\mathbf{C})$ we have $|B| = |C| = n \leq d$. There cannot be more than d vectors in either as they would not be linearly independent, and if $|B| < |C|$ then one can show the vectors of C are not

[1] In Chapter 3 we index from 0 to better reflect the implementation.

linearly independent. Furthermore, there exists a $\mathbf{U} \in \mathrm{GL}_n(\mathbb{Z})$ such that $\mathbf{C} = \mathbf{B} \cdot \mathbf{U}$. Indeed we may write $\mathbf{c}_i = x_{i,1}\mathbf{b}_1 + \cdots x_{i,n}\mathbf{b}_n$ with $x_{i,j} \in \mathbb{Z}$, and hence

$$\mathbf{C} = \mathbf{B} \cdot \mathbf{X}, \text{ for } \mathbf{X}[i,j] = x_{j,i}$$

and similarly we may write $\mathbf{b}_i = y_{i,1}\mathbf{c}_1 + \cdots + y_{i,n}\mathbf{c}_n$ with $y_{i,j} \in \mathbb{Z}$ and

$$\mathbf{B} = \mathbf{C} \cdot \mathbf{Y}, \text{ for } \mathbf{Y}[i,j] = y_{j,i}.$$

Therefore we have

$$\mathbf{C} = \mathbf{B} \cdot \mathbf{X} = \mathbf{C} \cdot \mathbf{Y} \cdot \mathbf{X},$$

where $\mathbf{Y}$ and $\mathbf{X}$ are integer matrices and the columns of $\mathbf{C}$ are linearly independent, as it is a basis. Therefore $\mathbf{Y} \cdot \mathbf{X} = \mathbf{I}_n$ and $\mathbf{Y} = \mathbf{X}^{-1}$, so $\mathbf{X} \in \mathrm{GL}_n(\mathbb{Z})$. Together these statements say that there are countably many bases for a lattice of rank greater than one, and that they are all related by unimodular transforms.

When considering a non full rank lattice $\Lambda = \Lambda(\mathbf{B})$ with basis $\mathbf{B} \in \mathbb{R}^{d\times n}$, we may always instead consider it as a full rank lattice in $\mathbb{R}^n$ such that none of its properties with respect to the Euclidean distance are altered. See [Gal12, Lem. 16.1.5] for the proof, modulo minor notational differences.

Lemma 2.2.1 (Projection to full rank)**.** *Let the non full rank lattice Λ have basis $\mathbf{B} \in \mathbb{R}^{d\times n}$. There exists a linear map $P\colon \mathbb{R}^d \to \mathbb{R}^n$ such that $P(\Lambda)$ is a full rank lattice. Furthermore, if $S = \mathrm{span}_{\mathbb{R}}(B)$ and $\mathbf{x}_1, \mathbf{x}_2$ are arbitrary elements of S, then $\|P(\mathbf{x}_1)\| = \|\mathbf{x}_1\|$ and $\langle P(\mathbf{x}_1), P(\mathbf{x}_2)\rangle = \langle \mathbf{x}_1, \mathbf{x}_2\rangle$. Finally, for $\mathbf{x} \in \mathbb{R}^d \setminus S$, $P(\mathbf{x}) = \mathbf{0}$.*

In particular, the above tells us that (Euclidean) lengths are invariant under the transformation P. If we are considering a measurable set $X \subset S$ then its d dimensional volume will always be 0, that is $L^d(X) = 0$. Instead we implicitly set $\mathrm{vol}(X) = L^n(P(X))$. Due to the invariance of lengths and inner products under P this is the same volume as if one considered the n dimensional volume implied by $S \cong \mathbb{R}^n$. Hence, from this point forward, provided we are only considering Euclidean properties, we may consider full rank lattices. Given the group structure of lattices, we may consider subgroups. These are called sublattices.

Definition 2.2.5 (Sublattice)**.** Given a lattice Λ, any $\Lambda' \subset \Lambda$ which is also a lattice is called a sublattice of Λ.

If Λ is a rank n lattice then the rank of Λ' is $n' \in [n]_0$, e.g. $\Lambda = \mathbb{Z}^2$ has sublattices $\{\mathbf{0}\}, \mathbb{Z} \times \{\mathbf{0}\}$, and $\mathbb{Z}^2$. If $\mathbf{B} \in \mathbb{R}^{d\times n}$ and $\Lambda' = \Lambda(\mathbf{B}')$ is a sublattice of $\Lambda = \Lambda(\mathbf{B})$ then

$\mathbf{B}' = \mathbf{B} \cdot \mathbf{T}$ for some full rank $\mathbf{T} \in \mathbb{Z}^{n \times n'}$ with $n' \leq n$. If $n = n'$ the rank is unchanged. We also define a primitive sublattice. This is a type of sublattice that we will later *project* the full lattice against, and this property ensures the resulting object is a lattice.

Definition 2.2.6 (Primitive sublattice)**.** A sublattice Λ' of Λ is primitive when any basis $\mathbf{B}' = (\mathbf{b}_1 \cdots \mathbf{b}_{n'})$ of Λ' can be completed into a basis $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_{n'}\, \mathbf{b}_{n'+1} \cdots \mathbf{b}_n)$ of Λ.

If $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n)$ is a basis of Λ, there are some natural candidates for primitive sublattices.

Definition 2.2.7. Let $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n) \in \mathbb{R}^{d \times n}$ be a basis for $\Lambda = \Lambda(\mathbf{B})$, then for $i \in [n]$ define $\mathbf{B}_i = (\mathbf{b}_1 \cdots \mathbf{b}_i)$ and $\Lambda_i = \Lambda(\mathbf{B}_i)$.

Indeed, all bases of Λ_i are of the form $\mathbf{B}_i \cdot \mathbf{U}_i$ for some $\mathbf{U}_i \in \mathrm{GL}_i(\mathbb{Z})$. We may always complete $\mathbf{B}_i \cdot \mathbf{U}_i$ with $(\mathbf{b}_{i+1} \cdots \mathbf{b}_n)$ since $\mathbf{B} \cdot \mathbf{U}$ is a basis of Λ when we take $\mathbf{U}$ to be the element of $\mathrm{GL}_n(\mathbb{Z})$ formed as

$$\mathbf{U} = \begin{pmatrix} \mathbf{U}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{n-i} \end{pmatrix}.$$

2.2.1 Invariants of a lattice

There are several useful quantities that, while they can be computed with respect to a particular basis of some lattice Λ, are in fact invariants of Λ. This means they depend only on Λ and not on the representation of it. The first we will define is the volume, which gives some notion of density for a lattice; the bigger the volume, the less dense the lattice. The second is the collection of lattice minima, each is the length of a shortest non trivial lattice vector not included in the span of the previous minima. Finding these lengths, and in particular lattice vectors that have them, will be a fundamental problem considered in this thesis. The final invariant is the Hermite factor of a lattice. While the volume gives an idea of density, it is simple to alter it in uninteresting ways via e.g. scaling the lattice. The Hermite factor of a lattice is a function of the volume and the first minimum (and is therefore trivially an invariant itself) that is subject to a famous upper bound of Minkowski.

Volume

As a lattice Λ is a subgroup of $\mathbb{R}^d$ we can consider the quotient group $\mathbb{R}^d/\Lambda$ of cosets $\mathbf{x} + \Lambda$ for $\mathbf{x} \in \mathbb{R}^d$. For any two vectors in such a coset, $\mathbf{y}, \mathbf{z} \in \mathbf{x} + \Lambda$, we write

$\mathbf{y} \equiv \mathbf{z} \bmod \Lambda$, and there exists a $\mathbf{w} \in \Lambda$ such that $\mathbf{y} - \mathbf{z} = \mathbf{w}$. A fundamental domain for Λ is a subset of $\mathbb{R}^d$ containing exactly one element of each such coset.

Definition 2.2.8. A fundamental domain of Λ is some $\mathcal{F} \subset \mathbb{R}^d$ such that for all $\mathbf{x} \in \mathbb{R}^d$ the coset $\mathbf{x} + \Lambda$ intersects $\mathcal{F}$ in a unique point. That is, for all $\mathbf{x} \in \mathbb{R}^d$, there exists a unique $\mathbf{x}'$ such that $\mathcal{F} \cap (\mathbf{x} + \Lambda) = \{\mathbf{x}'\}$.

A consequence of this definition is that for any point $\mathbf{x} \in \mathbb{R}^d$ there exists a unique pair $(\mathbf{x}', \mathbf{w}_\mathbf{x}) \in \mathcal{F} \times \Lambda$ such that $\mathbf{x} = \mathbf{x}' + \mathbf{w}_\mathbf{x}$. That is, for any $\mathbf{x} \in \mathbb{R}^d$ there is a unique $\mathbf{x}'$ such that $\mathbf{x} \equiv \mathbf{x}' \bmod \Lambda$ and $\mathbf{x}' \in \mathcal{F}$. Note that for any $\mathbf{y} \in \mathbb{R}^d$ the translation $\mathbf{y} + \mathcal{F}$ of a fundamental domain $\mathcal{F}$ is itself a fundamental domain.

A fundamental parallelotope of a lattice is a region described by some basis.

Definition 2.2.9. Given a basis $\mathbf{B}$ of lattice Λ the (closed) fundamental parallelotope of this basis is

$$\bar{\mathcal{P}}(\mathbf{B}) = \left\{ \sum_{i=1}^{n} \alpha_i \cdot \mathbf{b}_i \colon \alpha_i \in [0,1] \right\}.$$

It is not quite a fundamental domain since $\mathbf{0} \in \bar{\mathcal{P}}(\mathbf{B})$ and $\mathbf{b}_1 \in \bar{\mathcal{P}}(\mathbf{B})$, and so setting $\mathbf{x} = \mathbf{0}$ means $\mathbf{x} + \Lambda$ intersects $\bar{\mathcal{P}}(\mathbf{B})$ at least twice. If we associate the opposite facets of $\bar{\mathcal{P}}(\mathbf{B})$ we have a torus, each point of which can be uniquely represented a point in the open parallelotope.

Definition 2.2.10. Given a basis $\mathbf{B}$ of lattice Λ the fundamental parallelotope of this basis is

$$\mathcal{P}(\mathbf{B}) = \left\{ \sum_{i=1}^{n} \alpha_i \cdot \mathbf{b}_i \colon \alpha_i \in [0,1) \right\}.$$

If $\mathbf{B}$ is full rank, then $\mathcal{P}(\mathbf{B})$ *is* an example of a fundamental domain for Λ.

Remark 2.2.1. If $\mathbf{B} \in \mathbb{R}^{d \times n}$ is not full rank then $\mathcal{P}(\mathbf{B})$ fails to be a fundamental domain for any $\mathbf{x} \in \mathbb{R}^d \setminus \operatorname{span}_{\mathbb{R}}(B)$, since the intersection of $\mathcal{P}(\mathbf{B})$ and $\mathbf{x} + \Lambda$ is empty. Assume we are in this non full rank case. Then we will always implicitly alter Definition 2.2.8 to quantify only over $\mathbf{x} \in \operatorname{span}_{\mathbb{R}}(B)$. Given this, $\mathcal{P}(\mathbf{B})$ is once again a fundamental domain for Λ. Since the d dimensional volume $L^d(\mathcal{P}(\mathbf{B})) = 0$ we make use of the convention given below Lemma 2.2.1; we set $\operatorname{vol}(\mathcal{P}(\mathbf{B})) = L^n(P(\mathcal{P}(\mathbf{B})))$. Finally, for any $\mathbf{y} \in \operatorname{span}_{\mathbb{R}}(B)$ the translation $\mathbf{y} + \mathcal{P}(\mathbf{B})$ is a fundamental domain for Λ, but for $\mathbf{y} \in \mathbb{R}^d \setminus \operatorname{span}_{\mathbb{R}}(B)$ it is not.

Let $\mathbf{x} = \alpha_1 \cdot \mathbf{b}_1 + \cdots + \alpha_n \cdot \mathbf{b}_n$ for $\alpha_i \in \mathbb{R}$, then the map $\phi \colon \operatorname{span}_{\mathbb{R}}(B) \to \mathcal{P}(\mathbf{B})$ given by

$$\mathbf{x} \mapsto (\alpha_1 - \lfloor \alpha_1 \rfloor) \cdot \mathbf{b}_1 + \cdots + (\alpha_n - \lfloor \alpha_n \rfloor) \cdot \mathbf{b}_n \in \mathcal{P}(\mathbf{B}),$$

gives the unique representative of the lattice coset $\mathbf{x} + \Lambda$ in $\mathcal{P}(\mathbf{B})$. This map preserves lattice cosets as only integer multiples of basis vectors are subtracted, and demonstrates that $\mathcal{P}(\mathbf{B})$ is a fundamental domain for Λ. Note that the volumes of $\bar{\mathcal{P}}(\mathbf{B})$ and $\mathcal{P}(\mathbf{B})$, and any translates of either, are equal.

If we let $\mathbf{y} = -(\mathbf{b}_1 + \cdots + \mathbf{b}_d)/2$ and consider $\mathbf{y} + \mathcal{P}(\mathbf{B})$ then we have a fundamental domain centred around $\mathbf{0}$.

Definition 2.2.11. Given a basis $\mathbf{B}$ of lattice Λ the centred fundamental parallelotope of this basis is $\mathcal{P}_{1/2}(\mathbf{B}) = \mathbf{y} + \mathcal{P}(\mathbf{B})$ for $\mathbf{y} = -(\mathbf{b}_1 + \cdots + \mathbf{b}_d)/2$, or equivalently

$$\mathcal{P}_{1/2}(\mathbf{B}) = \left\{ \sum_{i=1}^{n} \alpha_i \cdot \mathbf{b}_i \colon \alpha_i \in [-1/2, 1/2) \right\}.$$

For any fundamental domain $\mathcal{F}$ of $\Lambda = \Lambda(\mathbf{B})$ it follows from Definition 2.2.8 that we can partition $\mathrm{span}_{\mathbb{R}}(B)$ with sets $\mathbf{w} + \mathcal{F}$ for $\mathbf{w} \in \Lambda$. We therefore have the partition

$$\mathrm{span}_{\mathbb{R}}(B) = \bigcup_{\mathbf{w} \in \Lambda} \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}),$$

which intuitively tells us that the volume of a fundamental domain can be thought of as the volume that can be given to a single lattice vector; the larger this volume, the less dense the lattice. We therefore calculate the volume of $\mathcal{P}(\mathbf{B})$ and call it the volume of Λ.

Definition 2.2.12 (Volume)**.** Given a lattice Λ and a basis $\mathbf{B} \in \mathbb{R}^{d \times n}$ that generates it, the volume of Λ is

$$\mathrm{vol}(\Lambda) = \sqrt{\det(\mathbf{B}^t \mathbf{B})}.$$

Note that in the case of full rank lattices $\mathrm{vol}(\Lambda) = |\det(\mathbf{B})|$.

The volume is an invariant of the lattice, in that, though we define it with respect to a given basis, it is equal for all bases of Λ. For $\mathbf{U} \in \mathrm{GL}_n(\mathbb{Z})$,

$$\sqrt{\det((\mathbf{B} \cdot \mathbf{U})^t (\mathbf{B} \cdot \mathbf{U}))} = \sqrt{\det(\mathbf{U}^t) \cdot \det(\mathbf{B}^t \mathbf{B}) \cdot \det(\mathbf{U})} = \sqrt{\det(\mathbf{B}^t \mathbf{B})}.$$

It is also often called the determinant of Λ, given the above definition. If $\Lambda' \subset \Lambda$ is a sublattice such that $\Lambda' = \Lambda(\mathbf{B}')$, $\Lambda = \Lambda(\mathbf{B})$, and $\mathbf{B}' = \mathbf{B} \cdot \mathbf{T}$ then we know $\mathrm{vol}(\Lambda') = \sqrt{\det(\mathbf{T}^t \cdot \mathbf{T})} \cdot \mathrm{vol}(\Lambda)$.

Lattice minima

The next invariants are the n minima of a rank n lattice. The first minimum of Λ is defined as $\lambda_1(\Lambda) = \min\{\|\mathbf{w}\| : \mathbf{w} \in \Lambda \setminus \{\mathbf{0}\}\}$. The second minimum wants to somehow encode the length of the 'next shortest' vector of Λ, but note that, as in $\Lambda = \mathbb{Z}^n$, there can be many linearly independent vectors of length $\lambda_1(\Lambda)$. We also do not want the second minimum to return the length of some multiple of a vector $\mathbf{w}$ such that $\|\mathbf{w}\| = \lambda_1(\Lambda)$, e.g. in $\Lambda = \mathbb{Z} \times 3\mathbb{Z}$ we want $\lambda_1(\Lambda) = 1$ from $\mathbf{w} = (1, 0)^t$ and $\lambda_2(\Lambda) = 3$ from $\mathbf{w}' = (0, 3)^t$, and not $\lambda_2(\Lambda) = 2$ from $2 \cdot \mathbf{w}$. We therefore take the following definition that concerns itself with the next shortest vectors that are not in the span of vectors that satisfy the previous minima.

Definition 2.2.13 (Minima of Λ)**.** For a rank n lattice $\Lambda \subset \mathbb{R}^d$ and $i \in \{1, \ldots, n\}$, the i^{th} minimum is

$$\lambda_i(\Lambda) = \inf\left\{r \colon r \in \mathbb{R}^+,\ \operatorname{span}_{\mathbb{R}}\left(B_d\left(r\right) \cap \Lambda\right) \cong \mathbb{R}^j,\ \text{for } j \geq i\right\}.$$

So the lattice $\Lambda = \mathbb{Z} \times \mathbb{Z} \times 3\mathbb{Z}$ has $\lambda_1(\Lambda) = \lambda_2(\Lambda) = 1$ and $\lambda_3(\Lambda) = 3$, as we expect. We have some immediate bounds on the minima of a Λ given any basis for it. Indeed

$$\lambda_i(\Lambda) \leq \max\{\mathbf{b}_j \colon j \leq i\},\ \text{for any } 1 \leq i \leq n, \tag{2.1}$$

since the maximum is taken over a set of i linearly independent vectors.

Hermite factor

Finally we introduce the Hermite factor of a lattice. While it is true that we may partition space such that each vector in some Λ is given $\operatorname{vol}(\Lambda)$ volume, we cannot bound this volume, and we may take any lattice and scale it to give it any positive volume we desire; $\operatorname{vol}(\alpha \cdot \Lambda) = \alpha^n \cdot \operatorname{vol}(\Lambda)$, for $\alpha > 0$. We make the following definition, of the ratio of the first minimum of a lattice and the n^{th} root of its volume, to disallow such 'artifical' manipulations.

Definition 2.2.14 (Hermite Factor)**.** For a lattice Λ of rank n we define its Hermite factor as

$$h(\Lambda) = \frac{\lambda_1(\Lambda)}{\operatorname{vol}(\Lambda)^{1/n}}.$$

While it is clear that this quantity cannot be lower bounded, e.g. $\Lambda = \mathbb{Z} \times \alpha\mathbb{Z}$ has $h(\Lambda) = 1/\sqrt{\alpha}$ for $\alpha \geq 1$, Minkowski proved [Min96] a famous theorem on convex

bodies that leads to an upper bound on the Hermite factor. For this we appeal to Lemma 2.2.1 to consider full rank lattices.

Theorem 2.2.2 (Minkowski's First)**.** *Let $\Lambda \subset \mathbb{R}^d$ be a full rank lattice, and let C be a measurable subset of $\mathbb{R}^d$ such that it is convex, symmetric around $\mathbf{0}$ and has volume greater than $2^d \cdot \mathrm{vol}(\Lambda)$. Then C contains a non zero vector of Λ.*

We note that the requirement to have the volume of C be strictly greater than $2^d \cdot \mathrm{vol}(\Lambda)$ can be relaxed to greater than or equal if C is also compact, see e.g. [Hen02] and note that elements of the class $\mathcal{K}_0^d$ used therein[2] are compact. By taking $C = \bar{B}_d(r)$ such that $\bar{B}_d(r)$ has volume $2^d \cdot \mathrm{vol}(\Lambda)$, we have $\lambda_1(\Lambda) \leq r$. Solving for r gives the following.

Corollary 2.2.3. *For any full rank lattice $\Lambda \subset \mathbb{R}^d$ we have*

$$\lambda_1(\Lambda) \leq 2 \cdot v_d^{-1/d} \cdot \mathrm{vol}(\Lambda)^{1/d},$$

and consequently the following upper bound on the Hermite factor of any full rank $\Lambda \subset \mathbb{R}^d$

$$h(\Lambda) \leq 2 \cdot v_d^{-1/d},$$

where $v_d = V_d(1)$.

We note a useful upper bound, $2 \cdot v_d^{-1/d} < \sqrt{d}$. Minkowski's second theorem [Min96] is a strengthening of his first in that it relates the volume of C to the product of all d minima, not just the first. To state it we must define a more general notion of minima.

Definition 2.2.15. Let C be a bounded measurable subset of $\mathbb{R}^d$ which has non empty interior, is convex, and is symmetric around $\mathbf{0}$. We define

$$\lambda_i(C, \Lambda) = \inf\left\{r \in \mathbb{R}^+ : \mathrm{span}_{\mathbb{R}}(rC \cap \Lambda) \cong \mathbb{R}^j, \text{ for } j \geq i\right\}$$

Setting $C = \bar{B}_d(1)$ retrieves Definition 2.2.13, i.e. $\lambda_i(\Lambda) = \lambda_i(\bar{B}_d(1), \Lambda)$ for all i.

Theorem 2.2.4 (Minkowski's Second)**.** *Let $\Lambda \subset \mathbb{R}^d$ be a full rank lattice, and let C be a measurable subset of $\mathbb{R}^d$ such that it is convex, symmetric around $\mathbf{0}$ and has volume greater than $2^d \cdot \mathrm{vol}(\Lambda)$. Then $\lambda_1(C, \Lambda) \cdots \lambda_d(C, \Lambda) \leq 1$.*

Once again we can appeal to compactness, and therefore to closed balls, to remove the strictly greater than requirement on the volume of C. An immediate rewrite of the above in the same manner as for Minkowski's first theorem gives

[2] These are convex *bodies* which are compact by definition, as opposed to general convex *sets*.

Corollary 2.2.5. *For any full rank lattice $\Lambda \subset \mathbb{R}^d$ and $1 \leq j \leq d$ we have*

$$h(\Lambda) \leq \left(\prod_{i=1}^{j} \lambda_i(\Lambda)\right)^{1/j} \mathrm{vol}(\Lambda)^{-1/d} \leq 2 \cdot v_d^{-1/d}.$$

We may take j in this range as the products are non decreasing as j increases. The $j = 1$ case equals $h(\Lambda)$ by definition, and Minkowski's second theorem gives the maximal $j = d$ case. Indeed, letting $C = \bar{B}_d(r)$ such that $V_d(r) = 2^d \cdot \mathrm{vol}(\Lambda)$ we have $\lambda_1(C, \Lambda) \cdots \lambda_d(C, \Lambda) \leq 1$, or equivalently that $\lambda_1(\Lambda) \cdots \lambda_d(\Lambda) \leq r^d$. We calculate $r^d = 2^d \cdot v_d^{-1} \cdot \mathrm{vol}(\Lambda)$, and the corollary follows.

We also define Hermite's constant, which is the supremum of $h(\Lambda)^2$ over lattices Λ of a given rank.

Definition 2.2.16 (Hermite's constant)**.** Let $\mathbf{\Lambda}_d$ be the class of all full rank lattices $\Lambda \subset \mathbb{R}^d$. The Hermite constant in dimension d is

$$\gamma_d = \sup_{\Lambda \in \mathbf{\Lambda}_d} \frac{\lambda_1(\Lambda)^2}{\mathrm{vol}(\Lambda)^{2/d}}.$$

The bound on $h(\Lambda)$ from Corollary 2.2.3 therefore tells us $\gamma_d \leq 4 \cdot v_d^{-2/d}$, as it is independent of Λ. A better bound due to Blichfeldt [Bli29] is

Theorem 2.2.6 (Blichfeldt's bound on γ_d)**.** *For all $d \geq 1$ we have*

$$\gamma_d \leq \left(\frac{2}{\pi}\right) \cdot \Gamma(2 + d/2)^{2/d}.$$

Hermite constants are known exactly only for a few small dimensions, see Table 2.1, but there exist tight asymptotic bounds (and less tight non asymptotic bounds [WCW19]) that show it is linear in d. For example [Ngu10],

$$\frac{d}{2\pi e} + \frac{\log(\pi d)}{2\pi e} + o(1) \leq \gamma_d \leq \frac{1.744d}{2\pi e}(1 + o(1)).$$

Interestingly, it is not known whether $(\gamma_d)_d$ is increasing.

dimension	1	2	3	4	5	6	7	8	24
γ_d	1	$2/\sqrt{3}$	$2^{1/3}$	$\sqrt{2}$	$8^{1/5}$	$(64/3)^{1/6}$	$64^{1/7}$	2	4

Table 2.1 The known exact values of Hermite's constant. See [Mar03] for $d \leq 8$ and [CK04] for $d = 24$.

2.2.2 Some lattices

Modular lattices

Here we give the construction of two types of lattice that appear frequently in lattice based cryptography. They are both modular lattices, meaning that elements of them are periodic with respect to some modulus.

Definition 2.2.17 (Modular lattices)**.** A lattice $\Lambda \subset \mathbb{R}^d$ is modular if there exists some positive integer q such that

$$q\mathbb{Z}^d \subseteq \Lambda \subseteq \mathbb{Z}^d.$$

Note that this definition requires Λ to be full rank, since it includes $q\mathbb{Z}^d$. In particular, $\Lambda \subset \mathbb{Z}^d$ is an *integer* lattice and, by letting Q equal all lattice points with coordinates in $\{0, \ldots, q-1\}$, can be represented as $Q + q\mathbb{Z}^d$. Equivalently, checking whether $\mathbf{w} \in \Lambda$ is equivalent to checking whether there exists a $\mathbf{w}' \in Q$ such that $\mathbf{w} \equiv \mathbf{w}' \bmod q$.

The two lattices are constructed from a matrix of the form $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ with $n \leq m$, which we note is not a basis for either lattice. We break the convention used thus far of n representing the rank in this construction, to better reflect the use of these lattices in cryptography; the lattices formed will be full rank and have rank and dimension equal to m. The first lattice is the lattice of the kernel of $\mathbf{A}$,

$$\Lambda_q^{\perp}(\mathbf{A}) = \{\mathbf{w} \in \mathbb{Z}^m \colon \mathbf{A} \cdot \mathbf{w} \equiv \mathbf{0} \bmod q\}.$$

We note that this is indeed a lattice via the group theoretic definition; the group properties are immediate and the discreteness follows from $\Lambda_q^{\perp}(\mathbf{A}) \subset \mathbb{Z}^m$. It is also clearly modular and full rank, as it contains $q\mathbb{Z}^m$. The second lattice is the row span of $\mathbf{A}$ modulo q,

$$\Lambda_q(\mathbf{A}) = \{\mathbf{w} \in \mathbb{Z}^m \colon \mathbf{w} \equiv \mathbf{A}^t \cdot \mathbf{v} \bmod q,\ \mathbf{v} \in \mathbb{Z}^n\}.$$

This too is a clearly a full rank modular lattice. We can determine the volumes of these two lattices directly by constructing bases for them. Note that permuting the columns of $\mathbf{A}$ permutes the order of the coordinates of $\mathbf{w} \in \mathbf{\Lambda}_q^\perp(\mathbf{A})$ or $\mathbf{w} \in \Lambda_q(\mathbf{A})$. This is *not* the same as permuting the columns of some basis. A basis of $\mathbb{Z} \times 2\mathbb{Z}$ is $\mathbf{B} = (\mathbf{e}_1\ 2\mathbf{e}_2)$, if we swap the columns to form basis $\mathbf{B}'$ we still have $\Lambda(\mathbf{B}') = \mathbb{Z} \times 2\mathbb{Z}$ and not $2\mathbb{Z} \times \mathbb{Z}$. We can however remember the permutation of the columns of $\mathbf{A}$ and apply its inverse to the coordinates of $\mathbf{w}$ to retrieve our original lattice. We therefore find a basis for any column permutation of $\mathbf{A}$. We first set $\mathbf{A} = (\mathbf{A}_1|\mathbf{A}_2)$ with $\mathbf{A}_1 \in \mathbb{Z}_q^{n\times n}$ an element of $\mathrm{GL}_n(\mathbb{Z}_q)$, and $\mathbf{A}_2 \in \mathbb{Z}_q^{n\times(m-n)}$. What is the probability we can form such an invertible $\mathbf{A}_1$? Using [Han06, Cor. 2.9], if $q = p_1^{e_1} \dots p_s^{e_s}$ is the prime factorisation of q, we know the proportion of invertible matrices is given by

$$\frac{|\mathrm{GL}_n(\mathbb{Z}_q)|}{|\mathrm{M}_n(\mathbb{Z}_q)|} = \prod_{i=1}^{s}\left(\prod_{j=0}^{n-1}\left(1-p_i^{j-n}\right)\right).$$

Each bracketed product concerning p_i is bounded below by the q-Pochhammer symbol $(1/p_i;1/p_i)_\infty$ [Koe14, p. 27] which in our case represents the limit as $n \to \infty$. For each $\alpha \in (0, 1/2]$ we have that $(\alpha;\alpha)_\infty > 1/4$, and so the total product is greater than 4^{-s}, and in reality is larger as $(\alpha;\alpha)_\infty$ tends towards 1 as $\alpha \to 0$. If we sample $\mathbf{A} \leftarrow \mathsf{U}(\mathbb{Z}_q^{n\times m})$ we have $\lfloor m/n \rfloor$ independent uniform choices for $\mathbf{A}_1$, each of which has probability greater than 4^{-s} of being invertible. There are in fact $\binom{m}{n}$ possibilities for $\mathbf{A}_1$, though they are not independent and some may repeat. We note that q is often a prime power or a small polynomial in n in lattice based cryptography, so $s \in O(\log n)$ in the worst case, and often $s = 1$. Certainly therefore $m \in \mathrm{poly}(n)$ is sufficient to ensure we can form an invertible $\mathbf{A}_1$ and in the case where q is prime one can take $m = 2n$ to guarantee $\mathbf{A}_1$ is full rank with overwhelming probability [Mic11, Sec. 3.1].

Let us therefore assume we have an invertible $\mathbf{A}_1$. We may now form the following two bases in $\mathbb{Z}^{m\times m}$

$$\mathbf{B}_{\mathbf{A}}^\perp = \begin{pmatrix} q\mathbf{I}_n & -\mathbf{A}_1^{-1}\cdot\mathbf{A}_2 \\ \mathbf{0} & \mathbf{I}_{m-n} \end{pmatrix}, \qquad \mathbf{B}_{\mathbf{A}} = \begin{pmatrix} \mathbf{I}_n & \mathbf{0} \\ \left(\mathbf{A}_1^{-1}\cdot\mathbf{A}_2\right)^t & q\mathbf{I}_{m-n} \end{pmatrix} \tag{2.2}$$

and check that they are such that $\Lambda_q^\perp(\mathbf{A}) = \Lambda(\mathbf{B}_{\mathbf{A}}^\perp)$ and $\Lambda_q(\mathbf{A}) = \Lambda(\mathbf{B}_{\mathbf{A}})$. To make $\mathbf{B}_{\mathbf{A}}$ and $\mathbf{B}_{\mathbf{A}}^\perp$ integer bases we are implicitly choosing representatives in $\mathbb{Z}$ for the entries of $\mathbf{A}_1^{-1}\cdot\mathbf{A}_2$, e.g. the balanced representation. This choice is unimportant in theory, but the balanced representation minimises the size of the entries of these bases. We therefore have, when there exists a permutation of columns such that $\mathbf{A}_1$ is invertible,

that $\mathrm{vol}(\Lambda_q^\perp(\mathbf{A})) = q^n$, $\mathrm{vol}(\Lambda_q(\mathbf{A})) = q^{m-n}$. In general we have $\mathrm{vol}(\Lambda_q^\perp(\mathbf{A})) \leq q^n$ and $\mathrm{vol}(\Lambda_q(\mathbf{A})) \geq q^{m-n}$. In particular for q prime, where the notion of rank is well defined, if $\mathbf{A}$ has rank $0 \leq i \leq n$ over $\mathbb{Z}_q$ then $\mathrm{vol}(\Lambda_q^\perp(\mathbf{A})) = q^i$ and $\mathrm{vol}(\Lambda_q(\mathbf{A})) = q^{m-i}$.

In what follows we focus on $\mathbf{B}_\mathbf{A}^\perp$ that have maximal volume, though similar statements are true for $\mathbf{B}_\mathbf{A}$ with minimal volume. If we sample a basis of the form

$$\mathbf{B}_\mathbf{R}^\perp = \begin{pmatrix} q\mathbf{I}_n & \mathbf{R} \\ \mathbf{0} & \mathbf{I}_{m-n} \end{pmatrix},$$

by taking $\mathbf{R} \leftarrow X$, for X a random variable over $\mathbb{Z}_q^{n\times(m-n)}$, and form $\Lambda(\mathbf{B}_\mathbf{R}^\perp)$ then we obtain a sample from some random variable over the set

$$S = \{\Lambda_q^\perp(\mathbf{A}) \colon \mathrm{vol}(\Lambda_q^\perp(\mathbf{A})) = q^n\}.$$

One way to sample from S is to set $X = \mathsf{U}(\mathbb{Z}_q^{n\times(m-n)})$ and return $\Lambda(\mathbf{B}_\mathbf{R}^\perp)$, another is to sample $\mathbf{A} \leftarrow \mathsf{U}(\mathbb{Z}_q^{n\times m})$, reject it if $\Lambda_q^\perp(\mathbf{A})$ does not have maximal volume, and otherwise return $\Lambda_q^\perp(\mathbf{A})$.

Finally we note that in the case of $\mathbf{B}_\mathbf{A}$ we often prefer, e.g. in Section 2.4.3, to consider the basis which has had its columns and rows permuted such that

$$\mathbf{B}'_\mathbf{A} = \begin{pmatrix} q\mathbf{I}_{m-n} & \left(\mathbf{A}_1^{-1} \cdot \mathbf{A}_2\right)^t \\ \mathbf{0} & \mathbf{I}_n \end{pmatrix}.$$

Permuting the columns of a basis does not alter the lattice generated, and permuting the rows permutes the coordinates of lattice vectors, as before. This means that solving a lattice problem over $\Lambda(\mathbf{B}'_\mathbf{A})$ generally immediately solves it over $\Lambda(\mathbf{B}_\mathbf{A})$. We add this small complication because the basis profile (see Definition 2.4.8) of $\mathbf{B}'_\mathbf{A}$ seems to be more useful in practice.

Random lattices

In Definition 2.2.18 we give a construction of lattice bases that generate random lattices for the definition of random given below. They are referred to as Goldstein–Mayer lattices, after [GM03] which introduced them. The accuracy of various heuristics can be formally stated for lattices generated in this way.

Let $\mathbf{\Lambda}_d$ denote the class of all full rank lattices $\Lambda \subset \mathbb{R}^d$. We consider the relation $\sim$ on lattices (as sets) such that $\Lambda \sim \Lambda'$ when there exists an $\alpha \in \mathbb{R}^\times$ such that $\Lambda = \alpha \cdot \Lambda'$, i.e. we forget about scale. Let $G = \mathrm{SL}_d(\mathbb{R})$ and $H = \mathrm{SL}_d(\mathbb{Z})$, then the quotient $\mathbf{\Lambda}_d/\sim$

can be represented as G/H, which we call Ω_d. Considering G as a surface in $\mathbb{R}^{d^2}$ then G is a (locally compact) topological group and there is a unique up to scale Haar measure μ_H such that $(G, \mathscr{B}(G))$ is a measure space with measure μ_H. Note that Ω_d has open sets defined by the quotient topology. The Haar measure μ_H projects to a finite measure on Ω_d, which we normalise and call μ so that $\mu(\Omega_d) = 1$. Let $F \subset 2^{\Omega_d}$ be the set of measurable sets under μ, then we have the probability space (Ω_d, F, μ).

Finally, let $\mathscr{L}_n \subset \mathbf{\Lambda}_d$ be the set of all full rank integer lattices $\Lambda \subset \mathbb{Z}^d$ with volume n. This set is finite.

Theorem 2.2.7 ([GM03, Thm. 2.1])**.** *Let $A \in F$ such that its boundary has $\mu(\partial A) = 0$ and let $X_A \colon \Omega_d \to \{0, 1\}$ denote the indicator random variable for A, so $X_A(\Lambda) = 1$ if and only if $\Lambda \in A$, then*

$$\lim_{n\to\infty} \left(\frac{1}{|\mathscr{L}_n|} \cdot \sum_{\Lambda \in \mathscr{L}_n} X_A(\Lambda) \right) = \mu(A).$$

A point Λ is in the boundary ∂A if for all open sets O containing Λ we have $\mu(A \cap O) > 0$ and $\mu((\Omega_d \setminus A) \cap O) > 0$. The above theorem should be interpreted as allowing us to estimate the probability measure of A by taking a particular weighted average over the finite set of full rank integer lattices with a given volume. Compare this with a more visual example – consider the unit square $S = [0, 1] \times [0, 1] \subset \mathbb{R}^2$ and the two dimensional Lebesgue measure L^2. Since $L^2(S) = 1$ we have that $(S, \mathscr{B}(S), L^2)$ is a probability space. Imagine we have some process that selects a point p of S. If this process has the property that, for any measurable set $A \in \mathscr{B}(S)$, the probability that $p \in A$ equals $L^2(A)$, then we might call this process 'uniform sampling'. In our lattice case we have a sequence of processes $(M_n)_n$, realised by sampling $\Lambda \leftarrow \mathsf{U}(\mathscr{L}_n)$, such that for any measurable A with $\mu(\partial A) = 0$ the probability that M_n outputs some $\Lambda \in A$ tends to $\mu(A)$ as $n \to \infty$. Hence to sample a 'random' lattice, we should pick a large n and sample $\Lambda \leftarrow \mathsf{U}(\mathscr{L}_n)$.

If n is prime and each $x_i \in [n-1]_0$ then $\mathscr{L}_n$ consists of lattices generated by the following $(n^d - 1)/(n - 1)$ bases

$$\begin{pmatrix} n & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}, \begin{pmatrix} 1 & & & \\ x_1 & n & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}, \ldots, \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ x_1 & \cdots & x_{d-1} & n \end{pmatrix}.$$

Of these the rightmost basis type represents n^{d-1} lattices, and a $(n^d - n^{d-1})/(n^d - 1)$ fraction of them all, which tends to 1 as $n \to \infty$.

Definition 2.2.18 (Random lattice)**.** For a large prime p we call a random lattice one with a basis sampled uniformly from the set

$$\left\{ \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ x_1 & \cdots & x_{d-1} & p \end{pmatrix} : x_1, \ldots, x_{d-1} \leftarrow \mathsf{U}([p-1]_0) \text{ are i.i.d.} \right\}.$$

2.2.3 Orthogonal projections and Gram–Schmidt bases

Orthogonality, and how far from it a given lattice basis is, is an important idea in lattice based cryptography. One of the main tools we use is the Gram–Schmidt orthogonalisaton process. First we give definitions of orthogonality and orthogonal projections.

Definition 2.2.19 (Orthogonality)**.** Vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ are said to be orthogonal if $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.

Definition 2.2.20 (Orthogonal Projection)**.** For vector $\mathbf{x}$ and some non empty set of vectors $S \subseteq \mathbb{R}^d$, the orthogonal projection of $\mathbf{x}$ against $\operatorname{span}_{\mathbb{R}}(S)$ is the unique vector $\mathbf{y} \in \mathbb{R}^d$ such that $\mathbf{y} = \mathbf{x} - \mathbf{v}$ for some $\mathbf{v} \in \operatorname{span}_{\mathbb{R}}(S)$ and $\langle \mathbf{y}, \mathbf{w} \rangle = 0$ for all $\mathbf{w} \in \operatorname{span}_{\mathbb{R}}(S)$.

We write $\pi_S^{\perp} \colon \mathbb{R}^d \to \mathbb{R}^d$ for the orthogonal projection against the set $\operatorname{span}_{\mathbb{R}}(S)$. If $T \subset \mathbb{R}^d$ is a set, we define $\pi_S^{\perp}(T) = \{\pi_S^{\perp}(t) \colon t \in T\}$. In cases where we wish to project against a singleton $\{\mathbf{w}\}$, or against a basis $\mathbf{B}$, we abuse notation and write $\pi_{\mathbf{w}}^{\perp}$ and $\pi_{\mathbf{B}}^{\perp}$ for these projections, rather than $\pi_{\{\mathbf{w}\}}^{\perp}$ and $\pi_{\{\mathbf{b}_i\}_i}^{\perp}$. We assert the convention that projecting against the empty set results in the identity; $\pi_{\emptyset}^{\perp} = \mathrm{id}_{\mathbb{R}^d}$. Given two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ one can form the orthogonal projection of $\mathbf{y}$ with respect to $\mathbf{x}$ by subtraction

$$\pi_{\mathbf{x}}^{\perp}(\mathbf{y}) = \mathbf{y} - \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\langle \mathbf{x}, \mathbf{x} \rangle} \cdot \mathbf{x}.$$

One then has

$$\left\langle \mathbf{x}, \pi_{\mathbf{x}}^{\perp}(\mathbf{y}) \right\rangle = \langle \mathbf{x}, \mathbf{y} \rangle - \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\langle \mathbf{x}, \mathbf{x} \rangle} \cdot \langle \mathbf{x}, \mathbf{x} \rangle = 0,$$

and hence $\mathbf{x}$ and $\mathbf{y}$ are orthogonal.

A basis of some $\text{span}_{\mathbb{R}}(S) \subset \mathbb{R}^d$ is any linearly independent set of vectors $B \subset \mathbb{R}^d$ such that $\text{span}_{\mathbb{R}}(B) = \text{span}_{\mathbb{R}}(S)$. To project $\mathbf{x}$ against $\text{span}_{\mathbb{R}}(S)$ one can use an orthogonal basis (in which all basis vectors are pairwise orthogonal) for this span. Indeed, suppose $\mathbf{B}^* = (\mathbf{b}_1^* \cdots \mathbf{b}_n^*) \in \mathbb{R}^{d \times n}$ is such a basis, then for $i \neq j$ we have

$$\left\langle \mathbf{b}_i^*, \left(\pi_{\mathbf{b}_j^*}^{\perp} \circ \pi_{\mathbf{b}_i^*}^{\perp}\right)(\mathbf{x}) \right\rangle = 0,$$

since $\left\langle \mathbf{b}_i^*, \mathbf{b}_j^* \right\rangle = 0$, and so one may just perform each orthogonal projection $\pi_{\mathbf{b}_i^*}^{\perp}$ in turn. That is,

$$\left(\pi_{\mathbf{b}_n^*}^{\perp} \circ \cdots \circ \pi_{\mathbf{b}_1^*}^{\perp}\right)(\mathbf{x}) = \pi_S^{\perp}(\mathbf{x}).$$

We may assume we have a basis $\mathbf{B}$ for $\text{span}_{\mathbb{R}}(S)$ which is not necessarily orthogonal. This is where the Gram–Schmidt orthogonalisation process comes in, it produces an orthogonal basis $\mathbf{B}^* = (\mathbf{b}_1^* \cdots \mathbf{b}_n^*)$ such that $\text{span}_{\mathbb{R}}(\mathbf{B}) = \text{span}_{\mathbb{R}}(\mathbf{B}^*)$. The basis $\mathbf{B}^*$ is called the Gram–Schmidt basis of $\mathbf{B}$.

Definition 2.2.21 (Gram–Schmidt Orthogonalisation)**.** Given basis $\mathbf{B} \in \mathbb{R}^{d \times n}$ form $\mathbf{b}_i^*$ as

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j<i} \mu_{i,j} \cdot \mathbf{b}_j^*, \text{ with Gram–Schmidt coefficients } \mu_{i,j} = \frac{\left\langle \mathbf{b}_i, \mathbf{b}_j^* \right\rangle}{\left\langle \mathbf{b}_j^*, \mathbf{b}_j^* \right\rangle}.$$

Note that since computing $\mathbf{b}_i^*$ requires all $\mathbf{b}_j^*$ with $j < i$, this process is sensitive to the order of basis vectors in $\mathbf{B}$.

In the case of a lattice Λ with basis $\mathbf{B}$ only the first basis vector of the Gram–Schmidt basis $\mathbf{B}^*$ will necessarily be a lattice vector, due to the empty sum in the definition above. The main use of the Gram–Schmidt basis in our context is its use in lattice algorithms (see Section 2.3 and Section 2.4) and that it is formed of orthogonal vectors; the distance of a lattice basis from its associated Gram–Schmidt basis can tell us something about its *quality*, a notion to be defined later.

The relationship between $\mathbf{B}$ and $\mathbf{B}^*$ can be described in several ways,

$$\mathbf{B} = \begin{pmatrix} \mathbf{b}_1^* & \cdots & \mathbf{b}_n^* \end{pmatrix} \cdot \begin{pmatrix} 1 & \mu_{2,1} & \cdots & \mu_{n,1} \\ 0 & 1 & \cdots & \mu_{n,2} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix} = \mathbf{B}^* \cdot \mathbf{M} \tag{2.3}$$

$$= \begin{pmatrix} \mathbf{b}_1^\circ & \cdots & \mathbf{b}_n^\circ \end{pmatrix} \cdot \begin{pmatrix} \|\mathbf{b}_1^*\| & 0 & \cdots & 0 \\ 0 & \|\mathbf{b}_2^*\| & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & \|\mathbf{b}_n^*\| \end{pmatrix} \cdot \begin{pmatrix} 1 & \mu_{2,1} & \cdots & \mu_{n,1} \\ 0 & 1 & \cdots & \mu_{n,2} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix} \tag{2.4}$$

$$= \begin{pmatrix} \mathbf{b}_1^\circ & \cdots & \mathbf{b}_n^\circ \end{pmatrix} \cdot \begin{pmatrix} \langle \mathbf{b}_1, \mathbf{b}_1^\circ \rangle & \langle \mathbf{b}_2, \mathbf{b}_1^\circ \rangle & \cdots & \langle \mathbf{b}_n, \mathbf{b}_1^\circ \rangle \\ 0 & \langle \mathbf{b}_2, \mathbf{b}_2^\circ \rangle & \cdots & \langle \mathbf{b}_n, \mathbf{b}_2^\circ \rangle \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \langle \mathbf{b}_n, \mathbf{b}_n^\circ \rangle \end{pmatrix} = \mathbf{B}^\circ \cdot \mathbf{R}, \tag{2.5}$$

with $\mathbf{b}_i^\circ = \mathbf{b}_i^* / \|\mathbf{b}_i^*\|$ being orthonormal vectors that form an orthonormal basis denoted by $\mathbf{B}^\circ$. For the diagonal entries of (2.5) we use

$$\begin{aligned} \langle \mathbf{b}_i, \mathbf{b}_i^\circ \rangle &= \sum_{j<i} \left\langle \mu_{i,j} \mathbf{b}_j^*, \mathbf{b}_i^\circ \right\rangle + \langle \mathbf{b}_i^*, \mathbf{b}_i^\circ \rangle = \|\mathbf{b}_i^*\|, \text{ and for } j < i \\ \left\langle \mathbf{b}_i, \mathbf{b}_j^\circ \right\rangle &= \frac{\left\langle \mathbf{b}_i, \mathbf{b}_j^* \right\rangle}{\left\|\mathbf{b}_j^*\right\|} \\ &= \frac{1}{\left\|\mathbf{b}_j^*\right\|} \cdot \left(\left\langle \mathbf{b}_i^*, \mathbf{b}_j^* \right\rangle + \left\langle \mu_{i,j} \mathbf{b}_j^*, \mathbf{b}_j^* \right\rangle + \sum_{k<i, k \neq j} \left\langle \mu_{i,k} \mathbf{b}_k^*, \mathbf{b}_j^* \right\rangle \right) \\ &= \mu_{i,j} \left\|\mathbf{b}_j^*\right\|. \end{aligned}$$

It will often be useful to project vectors against the span of some number of the first basis vectors of a basis $\mathbf{B}$.

Definition 2.2.22 (Projection against $\mathbf{B}$)**.** For basis $\mathbf{B}$ we define $\pi_{\mathbf{B},\ell}$, $1 \leq \ell \leq n+1$, as the orthogonal projection $\pi_S^\perp$ for $S = \{\mathbf{b}_1, \ldots, \mathbf{b}_{\ell-1}\}$. If $\mathbf{B}$ is clear from context we write π_ℓ.

Note that we are neglecting the $\perp$ in this particular orthogonal projection notation, and that $\pi_{\mathbf{B},1} = \pi_1 = \mathrm{id}_{\mathbb{R}^d}$ for all $\mathbf{B}$. If $\mathbf{B}$ is full rank then the projection π_{n+1} is always the function $\mathbf{x} \mapsto \mathbf{0}$, but this is not the case if $n < d$. Using (2.5) above and given a

vector $\mathbf{w} \in \mathrm{span}_{\mathbb{R}}(B)$, i.e. $\mathbf{w} = \mathbf{B} \cdot \mathbf{v}$ for some $\mathbf{v} \in \mathbb{R}^n$, we have

$$\mathbf{w} = \mathbf{B}^\circ \cdot \mathbf{R} \cdot \mathbf{v} = \sum_{j=1}^{n} \left(v_j \sum_{i=j}^{n} \left\langle \mathbf{b}_i, \mathbf{b}_j^\circ \right\rangle \right) \cdot \mathbf{b}_j^\circ,$$

and so

$$\pi_\ell(\mathbf{w}) = \pi_\ell \left(\sum_{j=1}^{n} \left(v_j \sum_{i=j}^{n} \left\langle \mathbf{b}_i, \mathbf{b}_j^\circ \right\rangle \right) \cdot \mathbf{b}_j^\circ \right) = \sum_{j=\ell}^{n} \left(v_j \sum_{i=j}^{n} \left\langle \mathbf{b}_i, \mathbf{b}_j^\circ \right\rangle \right) \cdot \mathbf{b}_j^\circ \tag{2.6}$$

$$= \begin{pmatrix} \mathbf{b}_\ell^\circ & \cdots & \mathbf{b}_n^\circ \end{pmatrix} \cdot \begin{pmatrix} \langle \mathbf{b}_\ell, \mathbf{b}_\ell^\circ \rangle & \langle \mathbf{b}_{\ell+1}, \mathbf{b}_\ell^\circ \rangle & \cdots & \langle \mathbf{b}_n, \mathbf{b}_\ell^\circ \rangle \\ 0 & \left\langle \mathbf{b}_{\ell+1}, \mathbf{b}_{\ell+1}^\circ \right\rangle & \cdots & \left\langle \mathbf{b}_n, \mathbf{b}_{\ell+1}^\circ \right\rangle \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \langle \mathbf{b}_n, \mathbf{b}_n^\circ \rangle \end{pmatrix} \cdot \begin{pmatrix} v_\ell \\ \vdots \\ v_n \end{pmatrix}. \tag{2.7}$$

Therefore one simply drops the first $\ell - 1$ columns of $\mathbf{B}^\circ$, takes the appropriate bottomright minor of $\mathbf{R}$ and drops the topmost $\ell - 1$ entries of $\mathbf{v}$. One can do the same for general $\mathbf{w} \in \mathbb{R}^d$ by first splitting $\mathbf{w}$ as $\mathbf{w} = \mathbf{w_B} + \mathbf{w}^\perp$, its component in $\mathrm{span}_{\mathbb{R}}(B)$ and its component orthogonal to it, respectively, and then returning $\pi_\ell(\mathbf{w_B} + \mathbf{w}^\perp) = \pi_\ell(\mathbf{w_B}) + \mathbf{w}^\perp$.

The Gram–Schmidt vectors also allow us to compute or bound some of the lattice invariants from Section 2.2.1. For example, using (2.3) we have

$$\begin{aligned} \det(\mathbf{B}^t\mathbf{B}) &= \det((\mathbf{B}^* \cdot \mathbf{M})^t(\mathbf{B}^* \cdot \mathbf{M})) \\ &= \det(\mathbf{M}^t) \cdot \det((\mathbf{B}^*)^t) \cdot \det(\mathbf{B}^*) \cdot \det(\mathbf{M}) = \det((\mathbf{B}^*)^t\mathbf{B}^*), \end{aligned}$$

so we have

$$\mathrm{vol}(\Lambda) = \sqrt{\det(\mathbf{B}^t\mathbf{B})} = \sqrt{\det((\mathbf{B}^*)^t\mathbf{B}^*)} = \prod_{i=1}^{n} \|\mathbf{b}_i^*\|. \tag{2.8}$$

We also get lower bounds on the first lattice minimum from the lengths of the Gram–Schmidt vectors

$$\lambda_1(\Lambda) \geq \min \left\{ \left\| \mathbf{b}_j^* \right\| : j \geq 1 \right\}. \tag{2.9}$$

Indeed, let $\mathbf{w} = \mathbf{B} \cdot \mathbf{v} = \mathbf{B}^* \cdot \mathbf{M} \cdot \mathbf{v}$ such that $\|\mathbf{w}\| = \lambda_1(\Lambda)$. Also let k be the maximal index such that $v_k \neq 0$, which must exist since $\mathbf{w} \neq \mathbf{0}$. Then $|v_k| \geq 1$ is an integer and

$$\|\mathbf{w}\| = \left\| v_k \mathbf{b}_k^* + \sum_{l=1}^{k-1} \alpha_l \mathbf{b}_l^* \right\| \geq v_k \|\mathbf{b}_k^*\| \geq \min\left\{ \left\|\mathbf{b}_j^*\right\| : j \geq 1 \right\},$$

for some $\alpha_l \in \mathbb{R}$. We now have, by also appealing to Corollary 2.2.3, that

$$\min\left\{ \left\|\mathbf{b}_j^*\right\| : j \geq 1 \right\} \leq \lambda_1(\Lambda) \leq 2 \cdot v_n^{-1/n} \cdot \mathrm{vol}(\Lambda)^{1/n}.$$

We also introduce projected sublattices, which are objects important during lattice reduction procedures, see Section 2.4.

Definition 2.2.23 (Projected sublattices)**.** Let $\Lambda = \Lambda(\mathbf{B}) \subset \mathbb{R}^d$ be a rank n lattice and $1 \leq \ell < r \leq n+1$. Define $\mathbf{B}_{[\ell:\, r]} = (\pi_\ell(\mathbf{b}_\ell) \cdots \pi_\ell(\mathbf{b}_{r-1}))$ as the projected subbasis, and $\Lambda_{[\ell:\, r]} = \Lambda(\mathbf{B}_{[\ell:\, r]})$ as the projected sublattice. If $r = n+1$ then we denote $\mathbf{B}_{[\ell:r]}$ by $\mathbf{B}_{[\ell]}$ and $\Lambda_{[\ell:r]}$ by $\Lambda_{[\ell]}$.

Note that this definition is dependent on the basis used to define Λ, and also the order of its basis vectors. Recall Definition 2.2.7, then this intuitively makes sense, since in practice the projection π_ℓ can be realised by forming a Gram–Schmidt basis for $\mathbf{B}_{\ell-1}$. In fact, $\Lambda_{[\ell:\, r]}$ is exactly $\{\pi_\ell(\mathbf{w}) \colon \mathbf{w} \in \Lambda(\mathbf{B}_{r-1})\}$, or equivalently the projection of $\Lambda(\mathbf{B}_{r-1})$ against the real span of $\Lambda(\mathbf{B}_{\ell-1})$. We can be certain $\Lambda_{[\ell:\, r]}$ is a lattice because $\Lambda(\mathbf{B}_{\ell-1})$ is a primitive sublattice of $\Lambda(\mathbf{B}_{r-1})$.

Lemma 2.2.8. *Let Λ' be a rank $\ell - 1$ primitive sublattice of the rank n lattice Λ, and $\Lambda' = \Lambda(\mathbf{B}')$ for $\mathbf{B}' \in \mathbb{R}^{d \times (\ell-1)}$. The set $\pi_{\mathbf{B}'}^{\perp}(\Lambda)$ is a rank $n - \ell + 1$ lattice.*

Proof. Call $\mathbf{B}' = (\mathbf{b}_1 \cdots \mathbf{b}_{\ell-1})$. As Λ' is a primitive sublattice there exists a basis of Λ given by $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_{\ell-1}\ \mathbf{b}_\ell \cdots \mathbf{b}_n)$. The projection operator $\pi_{\mathbf{B}'}^{\perp}$ is then equivalent to $\pi_{\mathbf{B},\ell}$ which we shorten to π_ℓ. It is enough to show that $\pi_\ell(\mathbf{b}_\ell), \ldots, \pi_\ell(\mathbf{b}_n)$ are $n - \ell + 1$ linearly independent vectors. We have

$$\begin{aligned} \sum_{i=\ell}^{n} \alpha_i \cdot \pi_\ell(\mathbf{b}_i) = 0 &\Leftrightarrow \pi_\ell\left(\sum_{i=\ell}^{n} \alpha_i \cdot \mathbf{b}_i \right) = 0 \\ &\Leftrightarrow \sum_{i=\ell}^{n} \alpha_i \cdot \mathbf{b}_i \in \mathrm{span}_{\mathbb{R}}\left(\{\mathbf{b}_1, \ldots, \mathbf{b}_{\ell-1}\}\right). \end{aligned}$$

Since $\mathrm{span}_{\mathbb{R}}\left(\{\mathbf{b}_1, \ldots, \mathbf{b}_{\ell-1}\}\right)$ and $\mathrm{span}_{\mathbb{R}}\left(\{\mathbf{b}_\ell, \ldots, \mathbf{b}_n\}\right)$ intersect only at $\mathbf{0}$ we have

$$\sum_{i=\ell}^{n} \alpha_i \cdot \mathbf{b}_i = \mathbf{0}.$$

Since $\mathbf{b}_\ell, \ldots, \mathbf{b}_n$ are linearly independent as vectors of $\mathbf{B}$ we have $\alpha_i = 0$ for all i. □

To calculate $\mathrm{vol}\left(\Lambda_{[\ell:\, r]}\right)$ we note that $\mathbf{B}^*_{[\ell:\, r]} = (\mathbf{b}^*_\ell \cdots \mathbf{b}^*_{r-1})$ and so

$$\mathrm{vol}\left(\Lambda_{[\ell:\, r]}\right) = \prod_{i=\ell}^{r-1} \|\mathbf{b}^*_i\|.$$

Finally we note that the Gram–Schmidt basis can be used to give a fundamental domain for a lattice.

Lemma 2.2.9. *Let $\mathbf{B} \in \mathbb{R}^{d\times n}$ be a basis for the lattice $\Lambda = \Lambda(\mathbf{B})$, then $\mathcal{P}(\mathbf{B}^*)$ is a fundamental domain for Λ.*

Proof. We show a bijection between $\mathcal{P}(\mathbf{B})$ and $\mathcal{P}(\mathbf{B}^*)$ that preserves cosets of Λ. Let $\mathbf{w} = \mathbf{B}\cdot\mathbf{v} \in \mathcal{P}(\mathbf{B})$ with $\mathbf{v} \in [0,1)^n$. Recall the relation of (2.3). We want to subtract integer multiples of the basis vectors, some $\mathbf{x} = (x_1 \cdots x_n)^t \in \mathbb{Z}^n$, such that $\mathbf{w} - \mathbf{B}\cdot\mathbf{x} = \mathbf{B}\cdot(\mathbf{v}-\mathbf{x}) = \mathbf{B}^*\cdot\mathbf{M}\cdot(\mathbf{v}-\mathbf{x})$ has $\mathbf{M}\cdot(\mathbf{v}-\mathbf{x}) \in [0,1)^n$. Let $\bar{\mathbf{v}} = \mathbf{M}\cdot\mathbf{v}$, then

$$\bar{v}_i = v_i + \sum_{j=i+1}^{n} \mu_{j,i} v_j.$$

Note that $\bar{v}_n = v_n \in [0,1)$ already by assumption, and that replacing v_i with $v_i - x_i$ does not alter $\bar{v}_j$ for $j > i$. This suggests an iterative approach; always let $x_n = 0$ since $\bar{v}_n \in [0,1)$, then let $x_{n-1} = \lfloor \bar{v}_{n-1} \rfloor$ and update $v_{n-1} \leftarrow v_{n-1} - x_{n-1}$, then recalculate $\bar{v}_j$ for $j \in \{1, \ldots, n-2\}$. Continue until updating $v_1 \leftarrow v_1 - x_1$, for $x_1 = \lfloor \bar{v}_1 \rfloor$. We ultimately arrive at a point in $\mathcal{P}(\mathbf{B}^*)$ since if we consider the step $v_i \leftarrow v_i - x_i$, with x_i constructed as above, then the i^{th} entry of $\mathbf{M}\cdot(\mathbf{v} - x_i\mathbf{e}_i)$ is

$$v_i + \sum_{j=i+1}^{n} \mu_{j,i} v_j - \left\lfloor v_i + \sum_{j=i+1}^{n} \mu_{j,i} v_j \right\rfloor = \bar{v}_i - \lfloor \bar{v}_i \rfloor \in [0,1).$$

Further steps do not alter this entry. This procedure gives us a map $\phi\colon \mathcal{P}(\mathbf{B}) \to \mathcal{P}(\mathbf{B}^*)$ that preserves lattice cosets, as only lattice vectors are subtracted. If $\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{P}(\mathbf{B})$ and $\phi(\mathbf{y}_1) = \phi(\mathbf{y}_2) = \mathbf{z} \in \mathcal{P}(\mathbf{B}^*)$ then $\mathbf{y}_1 \equiv \mathbf{z} \bmod \Lambda$ and $\mathbf{y}_2 \equiv \mathbf{z} \bmod \Lambda$, so the two elements of $\mathcal{P}(\mathbf{B})$ differ by a lattice vector; $\mathbf{y}_1 \equiv \mathbf{y}_2 \bmod \Lambda$. In a fundamental domain this can only happen if $\mathbf{y}_1 = \mathbf{y}_2$, hence ϕ is injective. To prove surjectivity we show that a vector not in the image of ϕ cannot be in $\mathcal{P}(\mathbf{B}^*)$. This is equivalent to the statement all vectors of $\mathcal{P}(\mathbf{B}^*)$ are in $\mathsf{Im}\,\phi$, therefore ϕ is surjective. We know ϕ is an injective map and is therefore bijective to $\mathsf{Im}\,\phi$, and that it also preserves lattice cosets. Therefore, since $\mathcal{P}(\mathbf{B})$ is a fundamental domain, every lattice coset has a representative

in $\mathsf{Im}\,\phi$. For any $\mathbf{y}_2 \notin \mathsf{Im}\,\phi$ we must therefore be able to pick a $\mathbf{y}_1 \in \mathsf{Im}\,\phi$ such that $\mathbf{y}_1 \equiv \mathbf{y}_2 \bmod \Lambda$. Let $\mathbf{z} = \mathbf{B} \cdot \mathbf{v}$ be the element of $\mathcal{P}(\mathbf{B})$ such that $\phi(\mathbf{z}) = \mathbf{y}_1$. We therefore know $\mathbf{y}_1 = \mathbf{B} \cdot (\mathbf{v} - \mathbf{x})$ for some $\mathbf{x} \in \mathbb{Z}^n$ by the above procedure, and that $\mathbf{y}_2 = \mathbf{B} \cdot (\mathbf{v} - \mathbf{x}')$ for some $\mathbf{x}' \in \mathbb{Z}^n$ distinct from $\mathbf{x}$. Let k be the maximal index for which $x_k \neq x'_k$, then the k^{th} entries of $\mathbf{M} \cdot (\mathbf{v} - \mathbf{x})$ and $\mathbf{M} \cdot (\mathbf{v} - \mathbf{x}')$ are

$$(v_k - x_k) + \sum_{j=k+1}^{n} \mu_{j,k}(v_j - x_j) \quad \text{and} \quad (v_k - x'_k) + \sum_{j=k+1}^{n} \mu_{j,k}(v_j - x'_j)$$

respectively. Since k was chosen maximally the above two summations over j do not differ, that is $x_j = x'_j$ for $j > k$. The left expression is in the range $[0, 1)$ by construction, and the right expression differs by $x_k - x'_k \in \mathbb{Z} \setminus \{0\}$. Therefore the k^{th} entry of $\mathbf{M} \cdot (\mathbf{v} - \mathbf{x}')$ is not in $[0, 1)$ and $\mathbf{y}_2 \notin \mathcal{P}(\mathbf{B}^*)$. □

2.2.4 The Gaussian heuristic

Here we introduce a useful heuristic that estimates the number of lattice points in a given measurable body, alongside some comparison with provable statements, and conditions upon which it relies to be accurate. The heuristic is called the Gaussian heuristic, and the intuition is that if each vector $\mathbf{w} \in \Lambda$ occupies $\mathrm{vol}(\Lambda)$ volume, say in the centre of some $\mathcal{P}_{1/2}(\mathbf{B})$, then the number of lattice vectors in some measurable set C can be approximated by the number of $\mathcal{P}_{1/2}(\mathbf{B})$ that can fit within. As we want a heuristic that is agnostic both to the exact geometry of C and the choice of basis $\mathbf{B}$, this latter choice because the number of lattice vectors in C is agnostic to the choice of $\mathbf{B}$, we appeal solely to the volumes of these objects. We are assuming full rank lattices by Lemma 2.2.1. In particular, if $\Lambda \subset \mathbb{R}^d$ has rank n and $C \subset \mathbb{R}^d$, we apply P to obtain a full rank $\Lambda' \subset \mathbb{R}^n$ and consider $C' = P(C)$.

Definition 2.2.24 (Gaussian heuristic on a set C)**.** Given a measurable set $C \in \mathbb{R}^d$ the Gaussian heuristic estimates the number of lattice vectors in this set as

$$\mathrm{gh}(C) = \frac{\mathrm{vol}(C)}{\mathrm{vol}(\Lambda)}.$$

First we note that this heuristic can be arbitrarily violated by convex sets. Given $\Lambda = \mathbb{Z}^2$ the set $C = (0, 1) \times (0, i) \subset \mathbb{R}^2$ for $i > 0$ is convex and has volume i, and yet never contains a vector of Λ. However, this example is purposefully unhelpful. One can imagine symmetric, convex sets centred on the origin, like open balls. These are used to give an estimate for $\lambda_1(\Lambda)$ by solving for the r that gives $V_d(r) = \mathrm{vol}(\Lambda)$, as in

this instance $\mathrm{gh}(B_d(r)) = 1$.

$$\mathrm{vol}(B_d(r)) = \mathrm{vol}(\Lambda) \iff \pi^{d/2} \cdot \Gamma(1 + d/2)^{-1} \cdot r^d = \mathrm{vol}(\Lambda) \tag{2.10}$$

$$\iff r = \frac{\Gamma(1 + d/2)^{1/d}}{\sqrt{\pi}} \cdot \mathrm{vol}(\Lambda)^{1/d} \tag{2.11}$$

$$\iff r = v_d^{-1/d} \cdot \mathrm{vol}(\Lambda)^{1/d} \tag{2.12}$$

Definition 2.2.25 (Gaussian heuristic for Λ)**.** Given a lattice Λ the Gaussian heuristic estimates the first minimum $\lambda_1(\Lambda)$ as

$$\mathrm{gh}(\Lambda) = v_d^{-1/d} \cdot \mathrm{vol}(\Lambda)^{1/d} \approx \sqrt{d/2\pi e} \cdot (\pi d)^{1/2d} \cdot \mathrm{vol}(\Lambda)^{1/d} \approx \sqrt{d/2\pi e} \cdot \mathrm{vol}(\Lambda)^{1/d}.$$

We reuse the $\mathrm{gh}(\cdot)$ notation, but Λ has measure 0 and therefore will not be used as C in the $\mathrm{gh}(C)$ case. The first approximate equality comes the from the classical approximation to $\Gamma(z)$ due to Stirling. If we let $r = \mathrm{gh}(\Lambda)$ and take $C = B_d(\alpha \cdot r)$, then $\mathrm{gh}(C) = \alpha^d \cdot \mathrm{gh}(B_d(r)) = \alpha^d$, so as the radius increases beyond $\mathrm{gh}(\Lambda)$ we expect exponentially many lattice vectors in the dimension.

As a first comparison, note that Corollary 2.2.3 gives $\lambda_1(\Lambda) \leq 2 \cdot v_d^{-1/d} \cdot \mathrm{vol}(\Lambda)^{1/d}$, which is only twice as large as the Gaussian heuristic estimate. This style of estimate can be significantly improved upon using Theorem 2.2.6.

Lemma 2.2.10. *For a full rank $\Lambda \subset \mathbb{R}^d$ we have*

$$\lambda_1(\Lambda) \leq \sqrt{2} \cdot (1 + d/2)^{1/d} \cdot v_d^{-1/d} \cdot \mathrm{vol}(\Lambda)^{1/d} = \sqrt{2} \cdot (1 + d/2)^{1/d} \cdot \mathrm{gh}(\Lambda).$$

The factor $\sqrt{2} \cdot (1 + d/2)^{1/d}$ improves upon 2 for $d \geq 3$.

Proof. By using the identity $\Gamma(n + 1/2) = \frac{(2n)!}{4^n n!}\sqrt{\pi}$ for $n \in \mathbb{N}$ we find

$$\left(\frac{\Gamma(2 + d/2)}{\Gamma(1 + d/2)}\right)^{1/d} = (1 + d/2)^{1/d}.$$

By definition $\lambda_1(\Lambda) \leq \sqrt{\gamma_d} \cdot \mathrm{vol}(\Lambda)^{1/d}$, and from Theorem 2.2.6 $\sqrt{\gamma_d} \cdot \mathrm{vol}(\Lambda)^{1/d} \leq \sqrt{2} \cdot \Gamma(2 + d/2)^{1/d} \cdot \mathrm{vol}(\Lambda)^{1/d}/\sqrt{\pi}$. Together we have

$$\begin{aligned}\lambda_1(\Lambda) &\leq \sqrt{2} \cdot \left(\frac{\Gamma(2 + d/2)}{\Gamma(1 + d/2)}\right)^{1/d} \cdot \Gamma(1 + d/2)^{1/d} \cdot \mathrm{vol}(\Lambda)^{1/d}/\sqrt{\pi} \\ &= \sqrt{2} \cdot (1 + d/2)^{1/d} \cdot \mathrm{gh}(\Lambda).\end{aligned}$$

□

As a second comparison, for a random lattice of Definition 2.2.18 it is proven [Che13, Thm. 2.1.12] that as $d \to \infty$

$$\mathbb{E}\left[\lambda_1(\Lambda)\right] = (1 - \gamma/d) \cdot (2 \cdot \mathrm{vol}(\Lambda)/v_d)^{1/d},$$

which differs from the Gaussian heuristic estimate by a factor of $(1 - \gamma/d) \cdot 2^{1/d}$. This factor is essentially 1 even for small d. Here $\gamma \approx 0.577$ is the Euler–Mascheroni constant. We also recall the experimental evidence of [Che13, Fig. 3.3].

For the modular lattices introduced in Section 2.2.2, if q is prime and a full rank $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is sampled uniformly then the first minima of $\Lambda_q^{\perp}(\mathbf{A})$ and $\Lambda_q(\mathbf{A})$ are no more than a constant factor smaller than the upper bound given by Corollary 2.2.3 with overwhelming probability in m [Mic11, Sec. 3.1]. This is equivalent to saying that they differ by no more than a constant factor from the estimate the Gaussian heuristic gives.

2.2.5 Computational problems on lattices

Given a description of a lattice, one can ask many questions. What are the lengths of the minima, or more concretely, can lattice vectors of, or close to, these lengths, be found? How about if given a point in the ambient space $\mathbb{R}^d$, can the closest, or an unusually close, lattice vector be found? Computational problems relevant to this thesis are given below, in all cases we will assume we have an arbitrary basis for the lattice, and that the lattice is full rank. We note that, while not necessary for this work, when making formal complexity theory statements, any approximation factors are functions of d.

Definition 2.2.26 (Shortest vector problem, SVP)**.** Given a lattice $\Lambda \subset \mathbb{R}^d$ return $\mathbf{w} \in \Lambda$ with $\|\mathbf{w}\| = \lambda_1(\Lambda)$.

Definition 2.2.27 (Approximate shortest vector problem, α-SVP)**.** Given a lattice $\Lambda \subset \mathbb{R}^d$ and a factor $\alpha \in [1, \infty)$ return $\mathbf{w} \in \Lambda \setminus \{\mathbf{0}\}$ with $\|\mathbf{w}\| \leq \alpha \cdot \lambda_1(\Lambda)$.

Definition 2.2.28 (Hermite shortest vector problem, α-HSVP)**.** Given a lattice $\Lambda \subset \mathbb{R}^d$ and a factor $\alpha \in [1, \infty)$ return $\mathbf{w} \in \Lambda \setminus \{\mathbf{0}\}$ with $\|\mathbf{w}\| \leq \alpha \cdot \mathrm{vol}(\Lambda)^{1/d}$.

Definition 2.2.29 (Unique shortest vector problem, α-uSVP)**.** Given a lattice $\Lambda \subset \mathbb{R}^d$ and a factor $\alpha \in [1, \infty)$ such that $\lambda_2(\Lambda) > \alpha \cdot \lambda_1(\Lambda)$, return $\mathbf{w} \in \Lambda$ with $\|\mathbf{w}\| = \lambda_1(\Lambda)$.

Clearly setting $\alpha = 1$ makes α-SVP and SVP equivalent problems. Often SVP is referred to as exact SVP. We have $\alpha \cdot \lambda_1(\Lambda) \leq \alpha \cdot \sqrt{\gamma_d} \cdot \mathrm{vol}(\Lambda)^{1/d}$ by definition, so solving α-SVP solves $(\alpha \cdot \sqrt{\gamma_d})$-HSVP. More concretely, from Corollary 2.2.3 and the discussion below, one also has the relationship

$$\alpha \cdot \lambda_1(\Lambda) \leq 2\alpha \cdot v_d^{-1/d} \cdot \mathrm{vol}(\Lambda)^{1/d} < \alpha\sqrt{d} \cdot \mathrm{vol}(\Lambda)^{1/d}$$

so that solving α-SVP solves $\alpha\sqrt{d}$-HSVP. More interestingly [Lov86, p. 25] an oracle solving α-HSVP solves α^2-SVP, if called linearly many times.

In a sense α-HSVP is the most practical problem, in that the validity of a solution can be checked for any lattice using only a basis. We know from Lemma 2.2.10 that a solution to α-HSVP must exist for $\alpha(d) = \sqrt{2} \cdot (1 + d/2)^{1/d} \cdot v_d^{-1/d}$, but for random lattices as $d \to \infty$ we expect $\alpha(d) = (1 + \varepsilon) \cdot v_d^{-1/d}$ to be sufficient, as this asks for a lattice vector $\mathbf{w}$ such that $\|\mathbf{w}\| \leq (1+\varepsilon) \cdot \mathrm{gh}(\Lambda)$. The Darmstadt SVP Challenges [GS10] ask for an α-HSVP solution on a random lattice with $\alpha(d) = (1 + 0.05) \cdot v_d^{-1/d}$.

The uSVP problem is slightly different from the others which also have some factor α. It always asks for an exact solution, but instead as α grows the first minimum becomes more and more short compared to the second minimum. This gap can be exploited, indeed if it is exponential then there exist polynomial time algorithms that will find an exact SVP solution, see Section 2.4.1 on LLL. More generally solving α-SVP automatically solves α-uSVP, and so an oracle solving α-HSVP called linearly many times solves α^2-uSVP.

We know that α-SVP is NP hard under randomised reductions for slightly sublinear factors $\alpha(d) = d^{1/\log\log d}$ [HR07] and that it can be solved in polynomial time for the slightly subexponential factor $\alpha(d) = 2^{n \log\log d/\log d}$ by realising the SVP oracle of [Sch87] with a lattice sieve (see Section 2.5). Cryptography is built on approximation factors between these two extremes, in the polynomial approximation world. For an introduction to complexity theory results in this realm see [Reg10].

The computational problems introduced thus far regard vectors in the lattice. If instead we are given a vector in the ambient space, subject perhaps to various conditions, we can ask for lattice vectors close to it. These are the closest vector type problems.

Definition 2.2.30 (Closest vector problem, CVP)**.** Given a lattice $\Lambda \subset \mathbb{R}^d$ and a target $\mathbf{t} \in \mathbb{R}^d$, return $\mathbf{w} \in \Lambda$ such that $\|\mathbf{w} - \mathbf{t}\| = \min\{\|\mathbf{v} - \mathbf{t}\| : \mathbf{v} \in \Lambda\}$.

Definition 2.2.31 (Approximate closest vector problem, α-CVP)**.** Given a lattice $\Lambda \subset \mathbb{R}^d$, a factor $\alpha \in [1, \infty)$, and a target $\mathbf{t} \in \mathbb{R}^d$, return $\mathbf{w} \in \Lambda$ such that $\|\mathbf{w} - \mathbf{t}\| \leq \alpha \cdot \min\{\|\mathbf{v} - \mathbf{t}\| : \mathbf{v} \in \Lambda\}$.

Definition 2.2.32 (Bounded distance decoding, γ-BDD)**.** Given a lattice $\Lambda \subset \mathbb{R}^d$, a factor $\gamma \in (0, \infty)$, and a target $\mathbf{t}$ such that $\min\{\|\mathbf{v} - \mathbf{t}\| : \mathbf{v} \in \Lambda\} < \gamma \cdot \lambda_1(\Lambda)$, return $\mathbf{w} \in \Lambda$ such that $\|\mathbf{w} - \mathbf{t}\| = \min\{\|\mathbf{v} - \mathbf{t}\| : \mathbf{v} \in \Lambda\}$.

Again approximate CVP with $\alpha = 1$ is CVP, which is often called exact CVP. Note that γ-BDD is unusual in that it is the only problem that gets *harder* as γ grows – one must always return a closest lattice vector to $\mathbf{t}$, but the promised closeness of $\mathbf{t}$ to the lattice decreases as γ increases. If $\gamma \leq 1/2$ then there is a unique vector $\mathbf{w}$ in the ball of radius $\gamma \cdot \lambda_1(\Lambda)$ around $\mathbf{t}$, and this is the γ-BDD solution. Once $\gamma \cdot \lambda_1(\Lambda)$ is above the covering radius of the lattice, that is, the maximum distance a point of $\mathbb{R}^d$ can be from a lattice point, γ-BDD is equivalent to CVP.

We note that α-CVP is not easier than α-SVP [GMSS99] and so certainly the hardness results for α-SVP are applicable to α-CVP. However, in the CVP case the reductions are deterministic. There is also a close relationship between the uSVP and BDD problems [LM09, BSW16].

Finally we introduce the learning with errors, or LWE, problem [Reg05, Reg09b], and two further related lattice problems. The first, GapSVP, is the decision variant of approximate SVP, and the second, SIVP, asks for a set of linearly independent lattice vectors with lengths below a certain bound.

Definition 2.2.33 (Gap shortest vector problem, GapSVP_α)**.** Given a lattice $\Lambda \subset \mathbb{R}^d$, some $r \in \mathbb{R}^+$, and an approximation factor α output `YES` if $\lambda_1(\Lambda) \leq r$ and `NO` if $\lambda_1(\Lambda) > \alpha(d) \cdot r$. If $\lambda_1(\Lambda) \in (r, \alpha(d) \cdot r]$ then any output is correct.

Definition 2.2.34 (Shortest independent vectors problem, SIVP_α)**.** Given a lattice $\Lambda \subset \mathbb{R}^d$ and some approximation factor α output d linearly independent vectors $\mathbf{w}_1, \ldots, \mathbf{w}_d \in \Lambda$ such that $\|\mathbf{w}_i\| \leq \alpha(d) \cdot \lambda_d(\Lambda)$ for all i.

Definition 2.2.35 (Learning with errors, LWE (search))**.** Let $n \geq 1$ be an integer representing the secret dimension, and $q = q(n) \geq 2$ an integer denoting the modulus. Let $\mathbf{s} \in \mathbb{Z}_q^n$ be the secret. Let χ be a discrete random variable that gives pmf $f_\chi \colon \mathbb{Z} \to [0, 1]$. Let $A_{s,\chi}$ describe the distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ formed by sampling $\mathbf{a} \leftarrow \mathsf{U}(\mathbb{Z}_q^n), e \leftarrow \chi$ and then returning the pair $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e \bmod q)$. Given the ability to sample from $A_{s,\chi}$, return $\mathbf{s}$.

Intuitively this problem says 'find a secret vector when given many noisy inner products of it with other known vectors'. If χ taken mod q is either the uniform distribution or the zero error distribution, then the problem is either information theoretically hard or trivial, respectively. It is for errors between these extremes where the problem is interesting. There is also a decision variant of this problem thats asks one to distinguish samples from $A_{\mathbf{s},\chi}$ and $\mathsf{U}(\mathbb{Z}_q^n \times \mathbb{Z}_q)$. These problems are polynomially equivalent for prime q [Reg09b, Sec. 4], and also when q is the product of distinct small primes [Pei09, Sec. 3.2] or a prime power [ACPS09, Lem. 1], subject to some extra restrictions on χ. Further reductions between the search and decision variant that allow the width of the error to grow slightly in the decision case are found in [MP12, BLP$^+$13]. We note that though in the definition $\mathbf{s}$ is arbitrary, for prime power q one may instead sample $\mathbf{s} \leftarrow \chi^n$ without altering the hardness of the problem [ACPS09, MR09].

The relationship between LWE and lattices may not be immediately obvious, but stems from worst case to average case hardness results, and how one may reinterpret the LWE problem as a lattice problem. The LWE problem enjoys worst to average case quantum reductions from a number of lattice problems conjectured to be post quantum [Reg05, Reg09b], i.e. resistant to quantum polynomial time attacks. In particular, although we do not provide details here, a machine solving a properly parametrised average case instance of LWE implies a quantum machine that can solve worst case instances of the SIVP and GapSVP problems to small polynomial approximation factors. A classical reduction was first given in [Pei09], i.e. one in which a machine solving average case instances of LWE implies a *classical* machine that can solve an approximate worst case lattice problem. The downsides of this classical reduction are that the correct parametrisations of the LWE problem require an exponential modulus q, see the final row of [Pei09, Fig. 1], and it is based only on the GapSVP problem. However, this was partially remedied in [BLP$^+$13] where the same style of classical reduction was proven for polynomial q.

We reinterpret LWE as a lattice problem by numbering m samples from $A_{\mathbf{s},\chi}$ as $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \bmod q)_{i \leq m}$ and letting $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be the matrix with $\mathbf{a}_i$ as its i^{th} column. By considering the lattice $\Lambda_q(\mathbf{A})$ we may view solving LWE as an instance of the γ-BDD problem. Indeed, by considering $\mathbf{A}$ and $\mathbf{s}$ over $\mathbb{Z}$ via the balanced representatives we have that $\mathbf{A}^t \cdot \mathbf{s} + q \cdot \mathbf{x}$ is a point of this lattice for any $\mathbf{x} \in \mathbb{Z}^m$, and we may define the target $\mathbf{t}$ as $\mathbf{A}^t \cdot \mathbf{s} + \mathbf{e}$ where $\mathbf{e}$ is the concatenation of $e_1, \ldots, e_m$. Both the lattice $\Lambda_q(\mathbf{A})$ and the target $\mathbf{t}$ can be constructed from the LWE samples from $A_{\mathbf{s},\chi}$. If too few samples are chosen (m is too small), or the errors are too large, then $\mathbf{A}^t \cdot \mathbf{s}$ may not be the closest lattice point to $\mathbf{t}$. However, it is typically the case in cryptography

that χ is such that by taking a sufficiently large m not only is $\mathbf{A}^t \cdot \mathbf{s}$ a solution to the γ-BDD problem given target $\mathbf{t}$ as constructed above, but also that $\gamma \leq 1/2$.

The LWE problem has been incredibly versatile when building cryptographic primitives, a non exhaustive list of constructions include public key encryption [Reg09b, LP11, NAB+20], oblivious transfer [PVW08], identity based encryption [GPV08], hierarchical identity based encryption [CHKP12], fully homomorphic encryption [BV11, GSW13], attributed based encryption [GVW13], and digital signatures [BG14a]. There are also variants of LWE which use an increase in algebraic structure to create more efficient (both in terms of time and memory) constructions. In ring learning with errors (RLWE) [SSTX09, LPR10] the unstructured matrices and vectors above are replaced with ring elements. Module learning with errors (MLWE) [LS15] is a further generalisation that interpolates between LWE and RLWE. Here the unstructured matrices and vectors above are replaced with elements from a ring module of a given rank. The definitions, efficiency, and security of these more general variants of LWE will not be discussed further in this thesis.

How LWE, short vector type problems, and close vector type problems are solved will be dealt with throughout the next two chapters of these preliminaries.

2.3 Size reduction and nearest plane

In this section we will look at an algorithmic way to control the sizes of the $\mu_{i,j}$ that determine the relationship between some basis $\mathbf{B}$ and its Gram–Schmidt basis $\mathbf{B}^*$. This algorithm is called size reduction. As input it receives a basis vector $\mathbf{b}_i$, an index $j < i$, and subtracts from $\mathbf{b}_i$ specific multiples of $\mathbf{b}_1, \ldots, \mathbf{b}_j$ in a particular order. These subtractions do not alter the lattice generated by the basis, and ensure that $\mu_{i,1}, \ldots, \mu_{i,j}$ each have absolute value no more than one half. Running this process on all basis vectors of some $\mathbf{B}$, also in a particular order, is an important component of lattice reduction algorithms.

Size reduction can also be applied to arbitrary vectors $\mathbf{t} \in \mathbb{R}^d$, where it becomes an approximate CVP solver; this procedure was analysed by Babai [Bab86] on bases preprocessed, or 'reduced', by an algorithm called LLL [LLL82], and is often referred to as the Nearest Plane algorithm.

2.3.1 Size reduction on a basis

Let $\mathbf{B} \in \mathbb{R}^{d \times n}$ be the basis of a lattice, and consider (2.3). Algorithm 1 takes a single basis vector of $\mathbf{B}$, say $\mathbf{b}_i = \mathbf{B} \cdot \mathbf{e}_i = \mathbf{B}^* \cdot \mathbf{M} \cdot \mathbf{e}_i$, which looks like

$$\mathbf{b}_i = \begin{pmatrix} & & \\ \mathbf{b}_1^* & \cdots & \mathbf{b}_n^* \\ & & \end{pmatrix} \cdot \begin{pmatrix} \mu_{i,1} \\ \vdots \\ \mu_{i,i-1} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

and iteratively ensures $\mu_{i,j}, \ldots, \mu_{i,1}$ are small, for some $j < i$. We consider, in order, the Gram–Schmidt coefficients $\mu_{i,k}$ for k decrementing from j to 1. As shown below going 'bottom up' like this requires the $\mu_{i,\ell}$ with $\ell < k$ to be recomputed after ensuring $\mu_{i,k}$ is small, but guarantees that previous work is not undone. That is, once we have e.g. $|\mu_{i,j}| \leq 1/2$, it is not altered by subsequent steps of the algorithm. Going 'top down' would not require us to recompute the lower Gram–Schmidt coefficients, but would not guarantee that previous work remains unaltered.

Definition 2.3.1 (Size reduction of $\mathbf{b}_i$)**.** Given a basis vector $\mathbf{b}_i$ of some basis $\mathbf{B}$ and an index $j < i$, we call $\mathbf{b}_i$ size reduced (to index j) if $|\mu_{i,k}| \leq 1/2$ for all $k \in \{1, \ldots, j\}$. Typically we consider $j = i - 1$.

Algorithm 1 Size reduction for $\mathbf{b}_i$ of a basis $\mathbf{B}$.

Require: Basis vector $\mathbf{b}_i$, basis $\mathbf{B}$ and implied $\mu_{i,j}$, and index $j < i$

procedure SizeReduce$_{\mathbf{B},j}(\mathbf{b}_i)$
  for $k \leftarrow j, \ldots, 1$ **do**
    $\mathbf{b}_i \leftarrow \mathbf{b}_i - \lceil \mu_{i,k} \rfloor \mathbf{b}_k$
    for $\ell \leftarrow 1, \ldots, k$ **do**
      Recompute $\mu_{i,\ell} \leftarrow \mu_{i,\ell} - \lceil \mu_{i,k} \rfloor \mu_{k,\ell}$
  return $\mathbf{b}_i$

We show that Algorithm 1 both achieves the desired notion, that $|\mu_{i,k}| \leq 1/2$ for all $1 \leq k \leq j < i$, and that it does not alter the lattice generated by $\mathbf{B}$ when this new $\mathbf{b}_i$ replaces the original.

Lemma 2.3.1. *Let $\mathbf{b}_i' = \mathbf{b}_i - \lceil \mu_{i,k} \rfloor \mathbf{b}_k$ for some $k < i$, and let $\mu_{i,k}'$ be the implied new Gram–Schmidt coefficient. Then $\left|\mu_{i,k}'\right| \leq 1/2$.*

Proof. Decomposing $\mathbf{b}_k$ as $\mathbf{b}_k = \mathbf{b}_k^* + \sum_{\ell<k} \mu_{k,\ell}\mathbf{b}_\ell^*$ for the final equality, we see that

$$\mu'_{i,k} = \frac{\langle \mathbf{b}_i - \lceil \mu_{i,k} \rfloor \mathbf{b}_k, \mathbf{b}_k^* \rangle}{\langle \mathbf{b}_k^*, \mathbf{b}_k^* \rangle} = \mu_{i,k} - \lceil \mu_{i,k} \rfloor \frac{\langle \mathbf{b}_k, \mathbf{b}_k^* \rangle}{\langle \mathbf{b}_k^*, \mathbf{b}_k^* \rangle} = \mu_{i,k} - \lceil \mu_{i,k} \rfloor,$$

and hence $\left|\mu'_{i,k}\right| \leq 1/2$. □

We note that by fixing a rounding convention, in our case $\lceil n + 1/2 \rfloor = n + 1$ for all $n \in \mathbb{Z}$, we have that the new Gram–Schmidt coefficient is in $[-1/2, 1/2)$.

Lemma 2.3.2. *Let $1 \leq k < i \leq n$, $\mathbf{b}'_i \leftarrow \mathbf{b}_i - \lceil \mu_{i,k} \rfloor \mathbf{b}_k$, and let $\mu'_{i,\ell}$ be the implied new Gram–Schmidt coefficients. This allows $\mu'_{i,\ell} \neq \mu_{i,\ell}$ for $1 \leq \ell \leq k$, but fixes $\mu'_{i,\ell} = \mu_{i,\ell}$ for $k < \ell < i$.*

Proof. Again, decompose $\mathbf{b}_k$ as $\mathbf{b}_k = \mathbf{b}_k^* + \sum_{\ell<k} \mu_{k,\ell}\mathbf{b}_\ell^*$ and note that

$$\langle \mathbf{b}_k, \mathbf{b}_\ell^* \rangle = \begin{cases} \mu_{k,\ell} \left\|\mathbf{b}_\ell^*\right\|^2 & \text{if } \ell < k, \\ \left\|\mathbf{b}_\ell^*\right\|^2 & \text{if } \ell = k, \\ 0 & \text{if } \ell > k. \end{cases}$$

We therefore have the following cases

$$\mu'_{i,\ell} = \frac{\langle \mathbf{b}_i - \lceil \mu_{i,k} \rfloor \mathbf{b}_k, \mathbf{b}_\ell^* \rangle}{\langle \mathbf{b}_\ell^*, \mathbf{b}_\ell^* \rangle} = \mu_{i,\ell} - \lceil \mu_{i,k} \rfloor \cdot \begin{cases} \mu_{k,\ell} & \text{if } \ell < k, \\ 1 & \text{if } \ell = k, \\ 0 & \text{if } \ell > k. \end{cases}$$

This also shows how one should recompute the $\mu_{i,\ell}$ for $\ell < k$. The $\ell = k$ case collapses into the statement of Lemma 2.3.1. □

Lemma 2.3.3. *Let $\mathbf{b}'_i \leftarrow \textsc{SizeReduce}_{\mathbf{B},j}(\mathbf{b}_i)$ for any $j < i$, and let $\mathbf{B}'$ be defined as $\mathbf{B}$ with $\mathbf{b}_i$ replaced by $\mathbf{b}'_i$. Then $\Lambda(\mathbf{B}) = \Lambda(\mathbf{B}')$.*

Proof. Performing $\mathbf{b}'_i \leftarrow \mathbf{b}_i - \lceil \mu_{i,k} \rfloor \mathbf{b}_k$ for $k \leq j$ is equivalent to transforming $\mathbf{B}$ by the matrix $\mathbf{U}_k$, where $\mathbf{U}_k = \mathbf{I}_n + \mathbf{M}_k$ and $\mathbf{M}_k \in \mathbb{Z}^{n \times n}$ is all zeros except $\mathbf{M}[k, i] = -\lceil \mu_{i,k} \rfloor$. The matrix $\mathbf{U}_k$ is unimodular, and so is any product of such matrices. We have $\mathbf{B}' = \mathbf{B} \cdot \mathbf{U}_j \cdots \mathbf{U}_1$, hence the statement follows. □

Definition 2.3.2 (Size reduction for basis $\mathbf{B}$)**.** We say a basis $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n)$ is size reduced if $|\mu_{i,j}| \leq 1/2$ for all $i \in \{2, \ldots, n\}$ and $j \in \{1, \ldots, i-1\}$.

Algorithm 2 Size reduction for a basis $\mathbf{B}$.

Require: A basis $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n)$
1: **procedure** SizeReduce($\mathbf{B}$)
2: **for** $i \leftarrow 2, \ldots, n$ **do**
3: Update $\mathbf{b}_i \leftarrow \text{SizeReduce}_{\mathbf{B},i-1}(\mathbf{b}_i)$
return B

Algorithm 2 achieves this notion by applying Algorithm 1, in order, to $\mathbf{b}_1, \ldots, \mathbf{b}_n$, with $j = i - 1$ for $\mathbf{b}_i$.

We note that in Algorithm 2 we reuse notation somewhat, but here SizeReduce takes as input a basis and has no subscripts, so the distinction is clear. From repeated application of the lemmata above it is clear that Algorithm 2 does not alter the lattice, and each individual $\mathbf{b}_i$ is size reduced, at least immediately after we call $\text{SizeReduce}_{\mathbf{B},i-1}(\mathbf{b}_i)$. It remains to show that as i increases this size reduction is maintained; this follows immediately since $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$ are fixed during $\text{SizeReduce}_{\mathbf{B},i-1}(\mathbf{b}_i)$, and therefore so are their Gram–Schmidt coefficients. Note that this would not be the case if we were to decrement i. Finally note that $\mathbf{b}_1$ is always trivially size reduced.

In conclusion, after performing Algorithm 2 on some basis $\mathbf{B}$, we have $\mathbf{B} = \mathbf{B}^* \cdot \mathbf{M}$, where all off diagonal entries in $\mathbf{M}$ are either 0 below the diagonal (as before), or no greater than one half in absolute value above the diagonal.

2.3.2 Size reduction in general (nearest plane)

How should we interpret size reduction? We are removing integer multiples of basis vectors from a point in space, therefore if we let $\Lambda = \Lambda(\mathbf{B})$ the output remains in the same coset of Λ as the input. Furthermore, we are doing this in such a way that the coefficients representing this point in space in the basis $\mathbf{B}^*$ are small. We can also see size reduction very concretely, for some $k \leq j < i$ and $\mathbf{B} = \mathbf{B}^* \cdot \mathbf{M}$, if $\mathbf{b}'_i \leftarrow \mathbf{b}_i - \lceil \mu_{i,k} \rfloor \mathbf{b}_k$

as

$$\mathbf{B} = \mathbf{B}^* \cdot \begin{pmatrix} 1 & \mu_{2,1} & \cdots & \cdots & \cdots & \cdots & \cdots & \mu_{i,1} & \cdots & \mu_{n,1} \\ 0 & 1 & \mu_{3,2} & \cdots & \cdots & \cdots & \cdots & \mu_{i,2} & \cdots & \mu_{n,2} \\ \vdots & \ddots & \ddots & & & & & \vdots & & \vdots \\ \vdots & & \ddots & \ddots & & & & \mu_{i,k} & & \vdots \\ \vdots & & & \ddots & \ddots & & & \mu_{i,k+1} & & \vdots \\ \vdots & & & & \ddots & \ddots & & \vdots & & \vdots \\ \vdots & & & & & \ddots & \ddots & \mu_{i,i-1} & & \vdots \\ \vdots & & & & & & \ddots & 1 & & \vdots \\ \vdots & & & & & & & \ddots & \ddots & \mu_{n,n-1} \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}$$

becoming, by letting $\mathbf{B}' = (\mathbf{b}_1 \cdots \mathbf{b}'_i \cdots \mathbf{b}_n)$,

$$\mathbf{B}' = \mathbf{B}^* \cdot \begin{pmatrix} 1 & \mu_{2,1} & \cdots & \cdots & \cdots & \cdots & \cdots & \mu'_{i,1} & \cdots & \mu_{n,1} \\ 0 & 1 & \mu_{3,2} & \cdots & \cdots & \cdots & \cdots & \mu'_{i,2} & \cdots & \mu_{n,2} \\ \vdots & \ddots & \ddots & & & & & \vdots & & \vdots \\ \vdots & & \ddots & \ddots & & & & \mu'_{i,k} & & \vdots \\ \vdots & & & \ddots & \ddots & & & \mu_{i,k+1} & & \vdots \\ \vdots & & & & \ddots & \ddots & & \vdots & & \vdots \\ \vdots & & & & & \ddots & \ddots & \mu_{i,i-1} & & \vdots \\ \vdots & & & & & & \ddots & 1 & & \vdots \\ \vdots & & & & & & & \ddots & \ddots & \mu_{n,n-1} \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix}.$$

From here we are most interested in the output of size reduction operations, so to remove the need for many primes in the notation, we may label inputs with a superscript 'pre'. This also implicitly applies to Gram–Schmidt vectors and coefficients. Recall the notation from Definition 2.2.11 and Definition 2.2.7. If we perform $\mathbf{b}_i = \textsc{SizeReduce}_{\mathbf{B},i-1}(\mathbf{b}_i^{\text{pre}})$, we know from Definition 2.3.2 and $\mathbf{B} = \mathbf{B}^* \cdot \mathbf{M}$ that

$$\mathbf{b}_i = \mathbf{b}_i^* + \sum_{j<i} \mu_{i,j}\mathbf{b}_j^* \in \mathbf{b}_i^* + \mathcal{P}_{1/2}(\mathbf{B}_{i-1}^*).$$

Indeed, we always round ties upwards, therefore each $\mu_{i,j} \in [-1/2, 1/2)$ and so $\mathbf{b}_i$ falls in the i dimensional orthotope $\mathcal{P}_{1/2}(\mathbf{B}_{i-1}^*)$ centred around $\mathbf{b}_i^*$.

As proven in Lemma 2.2.9 the n dimensional orthotope $\mathcal{P}(\mathbf{B}^*)$ is a fundamental domain for $\Lambda(\mathbf{B})$, and therefore $\mathbf{b}_i^* + \mathcal{P}_{1/2}(\mathbf{B}^*)$, as a translation of it, is also. We are dealing with a subset of this fundamental domain, $\mathbf{b}_i^* + \mathcal{P}_{1/2}(\mathbf{B}_{i-1}^*)$, and so any lattice coset intersects it in either 0 or 1 places. We therefore know that $\mathbf{b}_i$ is the unique element in the intersection of the lattice coset $\mathbf{b}_i + \Lambda(\mathbf{B}) = \Lambda(\mathbf{B})$ and this subset. Note that were we to perform $\mathbf{b}_i \leftarrow \textsc{SizeReduce}_{\mathbf{B},i}(\mathbf{b}_i^{\text{pre}})$ we would always return $\mathbf{b}_i = \mathbf{0}$. Indeed we would first calculate $\mathbf{b}_i \leftarrow \mathbf{b}_i^{\text{pre}} - \lceil 1 \rfloor \mathbf{b}_i^{\text{pre}}$ and subsequently set all Gram–Schmidt coefficients to 0. This can be interpreted as

$$\mathbf{b}_i \in \mathcal{P}_{1/2}(\mathbf{B}_i^*),$$

the only lattice vector of which is $\mathbf{0}$.

It does beg the question, what if we tried size reducing a non basis vector, or indeed, a non lattice vector? If we instead consider $\mathbf{t} = \nu_1 \mathbf{b}_1^* + \cdots + \nu_n \mathbf{b}_n^*$ with each $\nu_i \in \mathbb{R}$, the syntax of Algorithm 1 does not quite allow for '$\textsc{SizeReduce}_{\mathbf{B},j}(\mathbf{t})$'. By almost identical arguments to Lemma 2.3.1 and Lemma 2.3.2 we arrive at Algorithm 3. We note that throughout this thesis $\mathbf{b}_i$ is reserved solely for lattice basis vectors, so the syntax $\textsc{SizeReduce}_{\mathbf{B},j}(\mathbf{t})$ is unambiguous. This procedure no longer alters

Algorithm 3 Size reduction for $\mathbf{t}$ by a basis $\mathbf{B}$.

Require: Vector $\mathbf{t}$ and implied ν_i, basis $\mathbf{B}$ and implied $\mu_{i,j}$, and index $1 \leq j \leq n$

```
procedure SizeReduce_{B,j}(t)
    t = t_B + t^⊥
    for k ← j, ..., 1 do
        t_B ← t_B − ⌈ν_k⌋ b_k
        for ℓ ← 1, ..., k do
            Recompute ν_ℓ ← ν_ℓ − ⌈ν_k⌋ μ_{k,ℓ}
    return t = t_B + t^⊥
```

the basis at all, instead it only uses it to size reduce a vector. If $\mathbf{B}$ is not full rank then we split an arbitrary $\mathbf{t} \in \mathbb{R}^d$ as $\mathbf{t} = \mathbf{t}_\mathbf{B} + \mathbf{t}^\perp$, its components in and orthogonal to $\text{span}_\mathbb{R}(B)$, respectively. We then have $\mathbf{t}_\mathbf{B} = \nu_1 \mathbf{b}_1^* + \cdots + \nu_n \mathbf{b}_n^*$ and $\textsc{SizeReduce}_{\mathbf{B},j}(\mathbf{t}_\mathbf{B} + \mathbf{t}^\perp) = \textsc{SizeReduce}_{\mathbf{B},j}(\mathbf{t}_\mathbf{B}) + \mathbf{t}^\perp$ for all $\mathbf{t}, \mathbf{B}$ and j. The output of Algorithm 3 satisfies the following definition.

Definition 2.3.3 (Size reduction of $\mathbf{t}$)**.** Given a vector $\mathbf{t} = \mathbf{t}_\mathbf{B} + \mathbf{t}^\perp$ with $\mathbf{t}_\mathbf{B} = \nu_1 \mathbf{b}_1^* + \cdots + \nu_n \mathbf{b}_n^*$, some basis $\mathbf{B}$, and an index $j \leq n$, we call $\mathbf{t}$ size reduced (to index j) if $|\nu_i| \leq 1/2$ for all $i \in \{1, \ldots, j\}$. Typically we consider $j = n$.

If we have $\mathbf{t}^{\text{pre}} = \nu_1^{\text{pre}}\mathbf{b}_1^* + \cdots + \nu_n^{\text{pre}}\mathbf{b}_n^*$ and perform $\mathbf{t} \leftarrow \textsc{SizeReduce}_{\mathbf{B},j}(\mathbf{t}^{\text{pre}})$ then

$$\mathbf{t} = \nu_1\mathbf{b}_1^* + \cdots + \nu_j\mathbf{b}_j^* + \nu_{j+1}^{\text{pre}}\mathbf{b}_{j+1}^* + \cdots + \nu_n^{\text{pre}}\mathbf{b}_n^*,$$

since $\mathbf{t}^{\text{pre}}$ is only altered by elements in the span of $\mathbf{b}_1^*, \ldots, \mathbf{b}_j^*$, and

$$\mathbf{t} \in \nu_{j+1}^{\text{pre}}\mathbf{b}_{j+1}^* + \cdots + \nu_n^{\text{pre}}\mathbf{b}_n^* + \mathcal{P}_{1/2}(\mathbf{B}_j^*),$$

is the unique element of this set in the coset $\mathbf{t}^{\text{pre}} + \Lambda(\mathbf{B})$. Since $\mathbf{t}^{\text{pre}}$ and $\mathbf{t}$ lie in the same lattice coset their difference lies in the lattice; $\mathbf{t}^{\text{pre}} - \mathbf{t} \in \Lambda(\mathbf{B})$. Furthermore, we also know

$$\mathbf{t}^{\text{pre}} - \mathbf{t} = (\nu_1^{\text{pre}} - \nu_1) \cdot \mathbf{b}_1^* + \cdots + (\nu_j^{\text{pre}} - \nu_j) \cdot \mathbf{b}_j^*,$$

which is in the span of the first j basis vectors. Hence $\mathbf{t}^{\text{pre}} - \mathbf{t} \in \Lambda(\mathbf{B}_j)$. We have therefore found a lattice vector at distance $\|\mathbf{t}\|$ from $\mathbf{t}^{\text{pre}}$ in the sublattice generated by the first j basis vectors. If we let $j = n$ then we recover Babai's Nearest Plane algorithm. This finds a nearby lattice vector $\mathbf{w} \in \Lambda$ to target $\mathbf{t}$ using $\mathbf{B}$, see Algorithm 4. We can use this

Algorithm 4 Babai's nearest plane for $\mathbf{t}$.

Require: Vector $\mathbf{t}$ and basis $\mathbf{B}$

1: **procedure** $\textsc{NearestPlane}_{\mathbf{B}}(\mathbf{t})$ **return** $\mathbf{w} \leftarrow \mathbf{t} - \textsc{SizeReduce}_{\mathbf{B},n}(\mathbf{t})$

algorithm to solve CVP type problems. Note that since $\textsc{SizeReduce}_{\mathbf{B},n}(\mathbf{t}_{\mathbf{B}} + \mathbf{t}^{\perp}) = \textsc{SizeReduce}_{\mathbf{B},n}(\mathbf{t}_{\mathbf{B}}) + \mathbf{t}^{\perp}$, when $\mathbf{t} = \mathbf{t}_{\mathbf{B}} + \mathbf{t}^{\perp}$ the orthogonal components $\mathbf{t}^{\perp}$ cancel in Line 1 of Algorithm 4. Furthermore, since the distance from some point $\mathbf{x} \in \text{span}_{\mathbb{R}}(B)$ to $\mathbf{t}$ is given by

$$\|\mathbf{t} - \mathbf{x}\| = \left\|\mathbf{t}_{\mathbf{B}} - \mathbf{x} + \mathbf{t}^{\perp}\right\| = \left(\|\mathbf{t}_{\mathbf{B}} - \mathbf{x}\|^2 + \left\|\mathbf{t}^{\perp}\right\|^2\right)^{1/2}, \tag{2.13}$$

finding the closest lattice vector $\mathbf{w}$ to $\mathbf{t}_{\mathbf{B}}$ finds the closest lattice vector to $\mathbf{t}$.

We summarise the above in a lemma. It says that $\mathbf{w} \leftarrow \textsc{NearestPlane}_{\mathbf{B}}(\mathbf{t})$ is the unique lattice vector such that $\mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$ contains $\mathbf{t}_{\mathbf{B}}$. Due to (2.13) we only consider targets in $\text{span}_{\mathbb{R}}(B)$ in the lemma.

Lemma 2.3.4. *Let* $\mathbf{B} \in \mathbb{R}^{d\times n}$ *be a basis for lattice* Λ, $\mathbf{t}^{\text{pre}} \in \text{span}_{\mathbb{R}}(B)$ *and* $\mathbf{t} \leftarrow \textsc{SizeReduce}_{\mathbf{B},n}(\mathbf{t}^{\text{pre}})$, *and* $\mathbf{w} \in \Lambda$. *We have* $\mathbf{w} = \mathbf{t}^{\text{pre}} - \mathbf{t} \Leftrightarrow \mathbf{t}^{\text{pre}} \in \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$.

Proof. If $\mathbf{w} = \mathbf{t}^{\text{pre}} - \mathbf{t}$ and $\mathbf{t} \leftarrow \textsc{SizeReduce}_{\mathbf{B},n}(\mathbf{t}^{\text{pre}})$ then we know $\mathbf{t} \in \mathcal{P}_{1/2}(\mathbf{B}^*)$ and $\mathbf{t}^{\text{pre}} = \mathbf{w} + \mathbf{t}$, hence $\mathbf{t}^{\text{pre}} \in \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$. For the counter implication let $\mathbf{t}^{\text{pre}} \in$

$\mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$ and $\mathbf{t} = \mathbf{t}^{\text{pre}} - \mathbf{B} \cdot \mathbf{v}$ for some $\mathbf{v} \in \mathbb{Z}^d$. As $\mathcal{P}_{1/2}(\mathbf{B}^*)$ is a fundamental domain for Λ we may partition space as

$$\text{span}_{\mathbb{R}}(B) = \bigcup_{\mathbf{w} \in \Lambda} \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*),$$

so for any $\mathbf{w}_1, \mathbf{w}_2 \in \Lambda$ there exists a unique $\mathbf{w}' \in \Lambda$ such that $\mathbf{w}_1 + \mathcal{P}_{1/2}(\mathbf{B}^*) - \mathbf{w}' = \mathbf{w}_2 + \mathcal{P}_{1/2}(\mathbf{B}^*)$, in particular $\mathbf{w}' = \mathbf{w}_1 - \mathbf{w}_2$. Furthermore, for any other lattice vector $\mathbf{w}'' \neq \mathbf{w}'$ we have that $\mathbf{w}_1 + \mathcal{P}_{1/2}(\mathbf{B}^*) - \mathbf{w}''$ has an empty intersection with $\mathbf{w}_2 + \mathcal{P}_{1/2}(\mathbf{B}^*)$. Therefore $\mathbf{t}^{\text{pre}} \in \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$, $\mathbf{t} \in \mathcal{P}_{1/2}(\mathbf{B}^*)$ and $\mathbf{t} = \mathbf{t}^{\text{pre}} - \mathbf{B} \cdot \mathbf{v}$ together imply $\mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*) - \mathbf{B} \cdot \mathbf{v} = \mathcal{P}_{1/2}(\mathbf{B}^*)$ and therefore that $\mathbf{w} - \mathbf{B} \cdot \mathbf{v} = \mathbf{0}$. Since $\mathbf{t} = \mathbf{t}^{\text{pre}} - \mathbf{B} \cdot \mathbf{v}$ we conclude that $\mathbf{w} = \mathbf{t}^{\text{pre}} - \mathbf{t}$. □

Given a CVP instance on $\Lambda = \Lambda(\mathbf{B})$ with some target $\mathbf{t}$, the success condition for $\text{NearestPlane}_{\mathbf{B}}(\mathbf{t})$ to return a CVP solution $\mathbf{w}$ is therefore that $\mathbf{t}_{\mathbf{B}} \in \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$. Note that if $\mathbf{B}$ is an orthogonal basis, i.e. $\mathbf{B} = \mathbf{B}^*$, then $\mathcal{P}_{1/2}(\mathbf{B}) = \mathcal{P}_{1/2}(\mathbf{B}^*)$ is a so called Voronoi cell for Λ, see [MV10b] for an introduction to their uses in lattice cryptography. This means that $\mathbf{w}$ is a CVP solution for all $\mathbf{t}_{\mathbf{B}} \in \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$. Therefore, $\text{NearestPlane}_{\mathbf{B}}(\mathbf{t})$ always solves CVP exactly if $\mathbf{B}$ is orthogonal. Outside of this orthogonal case, we give below a sufficient condition and a necessary condition for $\mathbf{t}_{\mathbf{B}} \in \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$.

A sufficient condition

Since $\mathcal{P}_{1/2}(\mathbf{B}^*)$ is an n dimensional orthotope, a sufficient condition to ensure $\mathbf{t}_{\mathbf{B}} \in \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$ is

$$\|\mathbf{t}_{\mathbf{B}} - \mathbf{w}\| \leq \min\left\{\frac{1}{2}\|\mathbf{b}_i^*\| : i \in \{1, \ldots, n\}\right\}.$$

If we set $r_s = \min\left\{\frac{1}{2}\|\mathbf{b}_i^*\| : i \in \{1, \ldots, n\}\right\}$ then this is equivalent to $\mathbf{t}_{\mathbf{B}} \in B_d(r_s; \mathbf{w}) \cap \text{span}_{\mathbb{R}}(B)$. This condition requires $\mathbf{t}_{\mathbf{B}}$ to lie in the largest ball centred on $\mathbf{w}$ that is itself entirely contained in $\mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$, clearly ensuring $\mathbf{t}_{\mathbf{B}} \in \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$.

A necessary condition

If we define the radius

$$r_n = \left(\sum_{i=1}^{n} \frac{\|\mathbf{b}_i^*\|^2}{2}\right)^{1/2}$$

then any $\mathbf{t_B}$ such that $\|\mathbf{t_B} - \mathbf{w}\| > r_n$ cannot be such that $\mathbf{t_B} \in \mathbf{w} + \mathcal{P}_{1/2}(\mathbf{B}^*)$. This follows since r_n is the distance from $\mathbf{w}$ to the farthest corner of $\mathbf{w}+\mathcal{P}_{1/2}(\mathbf{B}^*)$. Therefore, it is necessary that $\mathbf{t_B} \in B_d(r_n; \mathbf{w}) \cap \mathrm{span}_{\mathbb{R}}(B)$ for Algorithm 4 to solve CVP.

For $r \in (r_s, r_n)$ the behaviour of $\textsc{NearestPlane}_{\mathbf{B}}(\mathbf{t})$ becomes dependent on the geometry of $\mathbf{B}$ and $\mathbf{t}$ beyond just the lengths $\|\mathbf{b}_1^*\|, \ldots, \|\mathbf{b}_{n^*}\|$ and $\|\mathbf{t_B} - \mathbf{w}\|$.

An average case analysis

Since the distance of the lattice vector $\mathbf{w}$ output by $\textsc{NearestPlane}_{\mathbf{B}}(\mathbf{t})$ from target $\mathbf{t}$ is equal to the length of the unique $\mathbf{t}' \in \mathcal{P}_{1/2}(\mathbf{B}^*)$ such that $\mathbf{t} \equiv \mathbf{t}' \bmod \Lambda$, one can perform an average case analysis by specifying some distribution over $\mathcal{P}_{1/2}(\mathbf{B}^*)$. In particular, if we let $\ell_i = \|\mathbf{b}_i^*\|$ and assume that targets $\mathbf{t}$ are chosen such that their representative element $\mathbf{t}' \in \mathcal{P}_{1/2}(\mathbf{B}^*)$ is uniform in that fundamental domain, then we can calculate the average squared distance $\|\mathbf{w} - \mathbf{t}\|^2$ as

$$\left(\prod_{i=1}^{n} \ell_i\right)^{-1} \cdot \int_{-\ell_1/2}^{\ell_1/2} \cdots \int_{-\ell_n/2}^{\ell_n/2} \left(x_1^2 + \cdots + x_n^2\right) \, \mathrm{d}x_1 \cdots \, \mathrm{d}x_n = \frac{1}{12} \cdot (\ell_1^2 + \cdots + \ell_n^2). \quad (2.14)$$

This quantity is exactly the average squared length of an element in $\mathcal{P}_{1/2}(\mathbf{B}^*)$.

We can now begin to think about what kind of properties a basis $\mathbf{B}$ might have such that it is 'useful' or 'high quality' for this particular task. In particular we would like to maximise $\min\{\|\mathbf{b}_i^*\| : i \in \{1, \ldots, n\}\}$ for the sufficient condition, and minimise $(\|\mathbf{b}_1^*\|^2 + \cdots + \|\mathbf{b}_n^*\|^2)/12$ for the average case analysis, both subject to $\|\mathbf{b}_1^*\| \cdots \|\mathbf{b}_n^*\| = \mathrm{vol}(\Lambda)$. In both cases we succeed when $\|\mathbf{b}_1^*\| = \cdots = \|\mathbf{b}_n^*\| = \mathrm{vol}(\Lambda)^{1/n}$, so that the lengths of the Gram–Schmidt vectors are somehow balanced. This gives us a simple measure of quality for bases of a given lattice.

2.3.3 Size reduction in 'dimensions for free'

One particular use case of size reduction that appears in this thesis comes from [Duc18a]. Recall the notation of Definition 2.2.23. For this use case we consider some basis $\mathbf{B} \in \mathbb{R}^{d \times n}$ of the lattice $\Lambda = \Lambda(\mathbf{B})$ along with some projected sublattice of it, $\Lambda_{[\ell]}$ described by basis $\mathbf{B}_{[\ell]}$. Given many short vectors in $\Lambda_{[\ell]}$ it is shown how to return a short vector in Λ. The key ideas of [Duc18a] are that an algorithm called a lattice sieve (to be introduced in Section 2.5) can provide the required short vectors in $\Lambda_{[\ell]}$ and that the procedure that uses them to return a short vector in Λ adds only a small overhead compared to the complexity of this lattice sieve. The cost of using a lattice

sieve to output short lattice vectors grows exponentially in the rank of the lattice. Here, instead of calling a lattice sieve on Λ of rank n we are calling it on $\Lambda_{[\ell]}$ of rank $n - \ell + 1$, hence we have achieved $\ell - 1$ 'dimensions for free'.[3]

We refer to [Duc18a] for an analysis of when the method described in more detail below is successful and derivations of how large we may take ℓ. For completeness we note that we may take the sublinear $\ell \in \Theta(n/\log n)$, and that while this does not lead to an exponential improvement, the experimental results given in [Duc18a, Sec. 5] display its practical importance.

For our purposes we assume we are in the following setting. Assume $\mathbf{s} \in \Lambda$ is such that $\|\mathbf{s}\| = \lambda_1(\Lambda)$ and that we have some list of vectors $L \subset \Lambda_{[\ell]}$ such that $\pi_\ell(\mathbf{s}) \in L$. We wish to recover $\mathbf{s} \in \Lambda$ using L and size reduction. To start let $\mathbf{w} = \mathbf{B} \cdot \mathbf{v}$ for $\mathbf{v} \in \mathbb{Z}^n$ and $\mathbf{B} = (\mathbf{B}'|\mathbf{B}'')$ with $\mathbf{B}' \in \mathbb{R}^{d \times (\ell-1)}, \mathbf{B}'' \in \mathbb{R}^{d \times (n-\ell+1)}$. If we similarly split $\mathbf{v}^t = (\mathbf{v}'^t|\mathbf{v}''^t)$ such that $\mathbf{w} = \mathbf{B}' \cdot \mathbf{v}' + \mathbf{B}'' \cdot \mathbf{v}''$ then

$$\pi_\ell(\mathbf{w}) = \pi_\ell(\mathbf{B}' \cdot \mathbf{v}') + \pi_\ell(\mathbf{B}'' \cdot \mathbf{v}'') = \mathbf{0} + \mathbf{B}_{[\ell]} \cdot \mathbf{v}''.$$

Therefore, given any $\mathbf{w}_{[\ell]} \in \Lambda_{[\ell]}$ described as $\mathbf{w}_{[\ell]} = \mathbf{B}_{[\ell]} \cdot \mathbf{v}_{[\ell]}$, we know that any $\mathbf{w} \in \Lambda$ such that $\pi_\ell(\mathbf{w}) = \mathbf{w}_{[\ell]}$ is of the form $\mathbf{w} = \mathbf{B}' \cdot \mathbf{v}' + \mathbf{B}'' \cdot \mathbf{v}_{[\ell]}$. Put another way, to find the shortest $\mathbf{w} \in \Lambda$ that projects to $\mathbf{w}_{[\ell]}$ we need to find the $\mathbf{v}' \in \mathbb{Z}^{\ell-1}$ that minimises the length of $\mathbf{B}' \cdot \mathbf{v}' + \mathbf{B}'' \cdot \mathbf{v}_{[\ell]}$.

We relabel to frame this as a CVP problem. Let $\Lambda' = \Lambda(\mathbf{B}')$ and let $\mathbf{t} = \mathbf{B}'' \cdot \mathbf{v}_{[\ell]}$. Finding the lattice vector $\mathbf{w}' = \mathbf{B}' \cdot \mathbf{v}' \in \Lambda'$ closest to $-\mathbf{t}$ and returning $\mathbf{w} = \mathbf{w}' + \mathbf{t}$ then gives us exactly the shortest $\mathbf{w} \in \Lambda$ such that $\pi_\ell(\mathbf{w}) = \mathbf{w}_{[\ell]}$. We therefore present Algorithm 5. One subtle point to note is that the target $\mathbf{t}$ we construct is not in $\mathrm{span}_{\mathbb{R}}(B')$ since it is given by a linear combination of vectors of $\mathbf{B}''$. However, this is implicitly dealt with when Algorithm 3 is called during $\textsc{NearestPlane}_{\mathbf{B}'}(-\mathbf{t})$, since it internally decomposes $-\mathbf{t}$ as $-\mathbf{t}_{\mathbf{B}'} + -\mathbf{t}^\perp$. Assuming that $\pi_\ell(\mathbf{s}) \in L$, whether Algorithm 5 succeeds in returning $\mathbf{s}$ is entirely down to the geometry of $\mathbf{B}$, or more specifically $\mathbf{B}'$. Indeed, if $\pi_\ell(\mathbf{s}) \in L$ then one of the candidates $\mathbf{c}'$ on Line 7 will be $\mathbf{s}$ provided that CVP is solved exactly on Line 6. The analysis of [Duc18a, Sec. 3] shows that under several standard heuristics choosing $\ell \in \Theta(n/\log n)$ implies both that $\pi_\ell(\mathbf{s}) \in L$ and that applying Algorithm 4 to the above CVP instance solves it exactly. Therefore the shortest vector is recovered. Improving the geometry of lattice bases is the topic of the next section of these preliminaries.

[3] In [Duc18a] full rank lattices Λ are considered, so we avoid the awkward sounding 'ranks for free'.

Algorithm 5 Size reduction for dimensions for free.

Require: Basis $\mathbf{B}$ for $\Lambda = \Lambda(\mathbf{B})$, index ℓ, and $L \subset \Lambda(\mathbf{B}_{[\ell]})$

procedure DimsForFree$_{\mathbf{B}}(L)$
  $\mathbf{c} = \mathbf{B} \cdot \mathbf{e}_1$
  Let $\mathbf{B} = (\mathbf{B}'|\mathbf{B}'')$ with $\mathbf{B}' \in \mathbb{R}^{d\times(\ell-1)}$ and $\mathbf{B}'' \in \mathbb{R}^{d\times(n-\ell+1)}$
  for $\mathbf{w}_{[\ell]} = \mathbf{B}_{[\ell]} \cdot \mathbf{v}_{[\ell]} \in L$ **do**
    $\mathbf{t} = \mathbf{B}'' \cdot \mathbf{v}_{[\ell]}$
    $\mathbf{w}' \leftarrow$ NearestPlane$_{\mathbf{B}'}(-\mathbf{t})$
    $\mathbf{c}' = \mathbf{w}' + \mathbf{t}$
    if $\|\mathbf{c}'\| < \|\mathbf{c}\|$ **then**
      $\mathbf{c} \leftarrow \mathbf{c}'$
  return $\mathbf{c}$

2.4 Lattice reduction algorithms

Lattice reduction algorithms take as input a lattice basis and output a different basis for the same lattice that is somehow a *better* basis and may be guaranteed to satisfy some properties. We have already seen one potential definition for what better may mean in this context. Indeed the sufficient condition for solving exact CVP and the average case analysis of NearestPlane both suggest that balanced Gram–Schmidt lengths are preferable, $\|\mathbf{b}_1^*\| = \cdots = \|\mathbf{b}_n^*\|$. The behaviour of the Gram–Schmidt lengths is something lattice reduction algorithms allow us some control over.

Being in the balanced Gram–Schmidt lengths case is related to the basis being close to orthogonal, though they are not the same. Indeed, if

$$\mathbf{B} = \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \mathbf{C} = \begin{pmatrix} 2 & 10 \\ 0 & 2 \end{pmatrix}$$

then $\mathbf{B}$ is an orthogonal basis for some lattice that does not have balanced Gram–Schmidt lengths, and $\mathbf{C}$ is a non orthogonal basis for some lattice that does. However, if we have some method to control the $\mu_{i,j}$, e.g. Algorithm 2, then we can show that having balanced Gram–Schmidt lengths gives us some guarantees about how far from orthogonal the lattice basis is.

Definition 2.4.1 (Orthogonality Defect)**.** Given a basis $\mathbf{B} \in \mathbb{R}^{d\times n}$ its orthogonality defect is

$$\mathrm{od}(\mathbf{B}) = \prod_{i=1}^{n} \|\mathbf{b}_i\| \cdot \mathrm{vol}(\Lambda)^{-1} = \prod_{i=1}^{n} \frac{\|\mathbf{b}_i\|}{\|\mathbf{b}_i^*\|}.$$

Since $\mathbf{b}_i = \mathbf{b}_i^* + \sum_{j<i} \mu_{i,j}\mathbf{b}_j^*$ and $\mathbf{B}^*$ is orthogonal we have $\|\mathbf{b}_i^*\| \leq \|\mathbf{b}_i\|$ for all i. Therefore $\mathrm{od}(\mathbf{B}) \geq 1$. It is clear that a basis $\mathbf{B}$ with $\mathbf{b}_i = \mathbf{b}_i^*$ for all i is orthogonal,

and that in this case $\|\mathbf{b}_i^*\| = \|\mathbf{b}_i\|$. Conversely, if $\|\mathbf{b}_i^*\| = \|\mathbf{b}_i\|$ then $\mu_{i,j} = 0$ for all j, since $\mathbf{B}^*$ is orthogonal, therefore $\mathbf{b}_i^* = \mathbf{b}_i$ and a basis with $\|\mathbf{b}_i^*\| = \|\mathbf{b}_i\|$ for all i is also orthogonal. The above tells us that $\mathrm{od}(\mathbf{B}) = 1 \Leftrightarrow \|\mathbf{b}_i^*\| = \|\mathbf{b}_i\| \Leftrightarrow \mathbf{b}_i^* = \mathbf{b}_i \Leftrightarrow \mathbf{B}$ is an orthogonal basis. We may then say that the closer in length each $\|\mathbf{b}_i^*\|$ is to $\|\mathbf{b}_i\|$, the closer to orthogonal $\mathbf{B}$ is. If we have $\|\mathbf{b}_1^*\| = \cdots = \|\mathbf{b}_n^*\| = C = \mathrm{vol}(\Lambda)^{1/n}$ then for a size reduced basis $\mathbf{B}$

$$\|\mathbf{b}_i\|^2 = \|\mathbf{b}_i^*\|^2 + \sum_{j<i} \mu_{i,j}^2 \cdot \left\|\mathbf{b}_j^*\right\|^2 \leq \|\mathbf{b}_i^*\|^2 + C^2 \cdot \left(\frac{i-1}{4}\right) = \|\mathbf{b}_i^*\|^2 \cdot \left(\frac{i+3}{4}\right).$$

We therefore have

$$\|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_i\|^2 \leq \|\mathbf{b}_i^*\|^2 \cdot \left(\frac{i+3}{4}\right),$$

so that for the worst index we have $\|\mathbf{b}_n\|^2 \leq (n+3) \cdot \|\mathbf{b}_n^*\|^2 / 4$.

Another way in which we may discuss the quality of the output basis is via the lengths of basis vectors, in particular the first. Finding short lattice vectors is clearly a necessary component of solving SVP type problems over lattices, but is also a key part of solving instances of the LWE problem via lattice reduction, see Section 2.4.3.

2.4.1 The LLL and BKZ algorithms

The two lattice reduction algorithms we will focus on in this thesis are the LLL and BKZ algorithms.

The LLL algorithm

The first lattice reduction algorithm we will introduce is the LLL algorithm [LLL82]. It has applications across number theory and computer science, for example in finding integer relations and minimal polynomials of algebraic numbers, and it has also been used to disprove a conjecture of Mertens [OtR85]. For our purposes it provides a certain notion of quality for an output lattice basis. We will first describe the properties of an LLL reduced basis for a lattice, and then give a terminating algorithm that outputs such a basis, proving existence the gentle way.

Definition 2.4.2 (LLL reduced basis)**.** A basis $\mathbf{B} \in \mathbb{R}^{d \times n}$ is LLL reduced with parameter $\delta \in (1/4, 1)$, or δ-LLL reduced, when

- $\mathbf{B}$ is size reduced by Definition 2.3.2, and
- for all $i \in \{2, \ldots, n\}$ we have $\|\mathbf{b}_i^*\|^2 \geq (\delta - \mu_{i,i-1}^2) \cdot \left\|\mathbf{b}_{i-1}^*\right\|^2$.

Notice that the second condition, called the Lovász condition, controls the maximum rate at which the Gram–Schmidt lengths can decrease. Indeed, as the first condition guarantees that $\mathbf{B}$ is size reduced, the term $\delta - \mu_{i,i-1}^2 \geq \delta - 1/4$ is bounded away from 0. Clearly the guarantee given by an LLL reduced basis gets stronger as δ gets closer to 1. We have from [Gal12, Thm. 17.2.12] that

Lemma 2.4.1. *If $\mathbf{B} \in \mathbb{R}^{d\times n}$ is a 3/4-LLL reduced basis and $\Lambda = \Lambda(\mathbf{B})$ then*

- $\|\mathbf{b}_1\| \leq 2^{(n-1)/2} \cdot \lambda_1(\Lambda)$*, and*
- $\|\mathbf{b}_1\| \leq 2^{(n-1)/4} \cdot \mathrm{vol}(\Lambda)^{1/n}$.

The choice of $\delta = 3/4$ is mostly historical, and indeed as $\delta \to 1$ the 2 in the above lemma becomes 4/3. Interestingly, an LLL reduced basis is not necessarily one such that $\mathbf{b}_1$ is the shortest basis vector. Indeed, for any $\delta \in (1/4, 1)$ if

$$\mathbf{B} = \begin{pmatrix} 1 & 1/2 \\ 0 & \sqrt{\delta - 1/4} \end{pmatrix}$$

then $\|\mathbf{b}_2\| < \|\mathbf{b}_1\|$ and the conditions of Definition 2.4.2 are satisfied. One cannot simply reorder the basis vectors as then the size reduction requirement will be violated.

Algorithm 6 The LLL algorithm.

Require: Basis $\mathbf{B} \in \mathbb{R}^{d\times n}$ for $\Lambda = \Lambda(\mathbf{B})$ and implied $\mathbf{B}^*$, parameter $\delta \in (1/4, 1)$

procedure LLL($\mathbf{B}, \delta$)
  $k = 2$
  while $k \leq n$ **do**
    Update $\mathbf{b}_k \leftarrow \textsc{SizeReduce}_{\mathbf{B},k-1}(\mathbf{b}_k)$ in $\mathbf{B}$
    Recompute $\mathbf{B}^*$
    if $\|\mathbf{b}_k^*\|^2 \geq (\delta - \mu_{k,k-1}^2) \left\|\mathbf{b}_{k-1}^*\right\|^2$ **then**
      $k \leftarrow k + 1$
    else
      Swap the order of $\mathbf{b}_k$ and $\mathbf{b}_{k-1}$ in $\mathbf{B}$
      Recompute $\mathbf{B}^*$
      $k = \max\{2, k - 1\}$
  return B

We introduce LLL basis reduction in Algorithm 6. We note that the one can be far more efficient than simply recalculating the entirety of $\mathbf{B}^*$ after SizeReduce and swapping, but it is not important for our purposes.

The output of Algorithm 6 is a δ-LLL reduced basis. Indeed, the Lovász condition with parameter δ is immediate; the while loop does not exit otherwise. Whenever a swap occurs the indices below $k-1$ are unaffected, so $\mathbf{b}_1, \ldots, \mathbf{b}_{k-2}$ remain size reduced, and the while loop restarts by size reducing $\mathbf{b}_{k-1}$. Recalling Lemma 2.3.2 for the properties of $\textsc{SizeReduce}_{\mathbf{B},k-2}(\mathbf{b}_{k-1})$, and noting in particular that the Gram–Schmidt coefficients for $\mathbf{b}_k, \ldots, \mathbf{b}_n$ are also unchanged, we see that any call to Line 4 does not affect whether other basis vectors are size reduced. Therefore, as Algorithm 6 returns only after $k > n$, the returned basis is size reduced in the sense of Definition 2.3.2, and therefore is δ-LLL reduced.

The complexity of Algorithm 6 is usually considered on input an integer basis, $\mathbf{B} \in \mathbb{Z}^{d \times n}$. For any real basis, an arbitrarily close approximation of its entries can be made by a rational basis, which can then by scaled by the lowest common multiple of the denominators of its entries. We therefore consider integer bases. If we let $X = \max\{\|\mathbf{b}_i\|^2 : i \in \{1, \ldots, n\}\}$ then the above description of the LLL algorithm[4] has complexity $O(n^5 d(\log X)^3)$ bit operations using naïve rational arithmetic [Gal12, Cor. 17.5.4]. However, significant theoretical and practical improvements have been made by considering floating point arithmetic and variants of Algorithm 6 that provably output δ-LLL reduced bases [Ste10]. In any case, Algorithm 6 requires a finite amount of bit operations, and therefore terminates. The case of $\delta = 1$ is interesting in that a proof for polynomial complexity is not known, and given this parameterisation LLL is exactly the BKZ algorithm with block size 2, see the BKZ paragraph below.

What tasks can one perform using LLL? Using Lemma 2.4.1 and below, we can clearly solve α-SVP and α-HSVP for any $\alpha \geq (4/3)^{(n-1)/2}$ and $\alpha \geq (4/3)^{(n-1)/4}$ respectively. Also, if we are considering a uSVP instance with an exponentially large gap, specifically α-uSVP with $\alpha \geq (4/3)^{(n-1)/2}$, then we must find $\lambda_1(\Lambda)$ by calling LLL, and therefore solve the α-uSVP instance. In the case of CVP from [Bab86, Thm. 3.1] we know that NearestPlane solves α-CVP on a 3/4-LLL reduced basis $\mathbf{B}$ for any $\alpha \geq 2^{n/2}$. By letting $\delta \to 1$, as soon as $\delta \geq 1/4 + 1/\sqrt{2}$ then this is improved to $\alpha \geq 1.6 \cdot 2^{n/4}$ [Gal12, Thm. 18.1.7]. There are also some practical improvements due to Coppersmith to be found in [GGH97]. Something we will discuss more in Section 2.4.2 is the difference between the above worst case guarantees, and the 'average case' performance of LLL.

[4] At least when displaying slightly more nuance than simply 'Recompute $\mathbf{B}^*$'.

The BKZ algorithm

The second lattice reduction algorithm we will consider is known as the Block Korkine–Zolotarev, or BKZ, algorithm. The LLL algorithm makes local changes to the basis by considering whether swapping two neighbouring basis vectors will improve (under the Lovász condition) the current state of the basis. One can imagine instead choosing to reorder three neighbouring basis vectors. Of course, reordering basis vectors is just a special case of a unimodular matrix given by a permutation of the columns of the identity matrix. One need not be limited to two neighbouring basis vectors, or to these special cases of unimodular matrices. In the LLL case the proof that the number of swaps is polynomial relies on considering only this restricted class of unimodular matrices on pairs of neighbours, as the effect such swaps have on a *potential* function of the basis can be explicitly understood.

In [Sch87] generalisations of other notions of reduction for lattice bases are considered that ultimately provide a smooth trade off between the LLL and BKZ algorithms.

Definition 2.4.3 (Minkowski reduction)**.** Let $\mathbf{B} \in \mathbb{R}^{d\times n}$ define $\Lambda = \Lambda(\mathbf{B})$. We say $\mathbf{B}$ is a Minkowski reduced basis of Λ if for all $i \in \{1, \ldots, n\}$ we have

$$\|\mathbf{b}_i\| = \min\{\|\mathbf{b}\| : \mathbf{b} \in \Lambda \text{ and } (\mathbf{b}_1 \cdots \mathbf{b}_{i-1}\, \mathbf{b}) \text{ can be extended to a lattice basis}\}.$$

This notion is considered in [Min91], and is strong as it requires $\mathbf{b}_1$ to be an SVP solution for Λ. Indeed, Lemma 2.4.2 and Corollary 2.4.3 below show that a basis does exist with the first vector achieving the first minimum. Note that one cannot demand that $\|\mathbf{b}_i\| = \lambda_i(\Lambda)$ for all i since

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

defines $\Lambda = \Lambda(\mathbf{B})$ where no basis can include vectors of length $\lambda_1(\Lambda) = \cdots = \lambda_5(\Lambda) = 2$. Ultimately the notion that was generalised is the following.[5]

[5] In [Sch87] this reduction notion is referred to as Korkine–Zolotarev reduction, sans Hermite. However, Hermite's use of the size reduction property led to the current naming convention.

Definition 2.4.4 (Hermite–Korkine–Zolotarev (HKZ) reduction)**.** Let $\mathbf{B} \in \mathbb{R}^{d \times n}$ define $\Lambda = \Lambda(\mathbf{B})$. We say $\mathbf{B}$ is a Hermite–Korkine–Zolotarev (HKZ) reduced basis of Λ if we have

- $\mathbf{B}$ is size reduced by Definition 2.3.2, and
- for all $i \in \{1, \ldots, n\}$, recalling Definition 2.2.23, $\|\mathbf{b}_i^*\| = \lambda_1(\Lambda_{[i]})$.

How might one, given as much computing power as necessary, go about finding an HKZ reduced basis? Indeed, given that there exist lattices for which one cannot have basis vectors matching the lengths of the minima, does a HKZ basis always exist for a lattice? In fact yes, using an elementary lemma found in e.g. [Kan83, Prop. 1].

Lemma 2.4.2. *Let $\mathbf{w} \in \Lambda$ such that there does not exist an $\alpha \in (0, 1)$ for which $\alpha \cdot \mathbf{w} \in \Lambda$. Then there exists a basis $\mathbf{B}$ for Λ containing $\mathbf{w}$.*

Corollary 2.4.3. *For any lattice Λ there exists a basis $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n)$ such that $\Lambda = \Lambda(\mathbf{B})$ and $\mathbf{b}_1 = \mathbf{w}$ for $\|\mathbf{w}\| = \lambda_1(\Lambda)$.*

Proof. Let $\mathbf{w} \in \Lambda$ be such that $\|\mathbf{w}\| = \lambda_1(\Lambda)$, then by definition there does not exist an $\alpha \in (0, 1)$ such that $\alpha \cdot \mathbf{w} \in \Lambda$. Hence a basis exists that contains the vector $\mathbf{w}$. The appropriate permutation $\mathbf{U} \in \mathrm{GL}_n(\mathbb{Z})$ of $\mathbf{I}_n$ ensures it is the first basis vector. □

Lemma 2.4.4. *For any Λ there exists a basis $\mathbf{B} \in \mathbb{R}^{d \times n}$ such that $\Lambda = \Lambda(\mathbf{B})$ and $\mathbf{B}$ is HKZ reduced.*

Proof. We prove the statement by induction on index $i \in \{1, \ldots, n\}$ such that we assume we have a size reduced basis $\mathbf{B}$ of Λ satisfying $\left\|\mathbf{b}_j^*\right\| = \lambda_1(\Lambda_{[j]})$ for all $j \in \{1, \ldots, i\}$. The $i = 1$ base case is immediate from Corollary 2.4.3 and Algorithm 2. We now construct a size reduced basis $\hat{\mathbf{B}}$ from $\mathbf{B}$ such that $\left\|\mathbf{b}_j^*\right\| = \lambda_1(\Lambda_{[j]})$ for all $j \in \{1, \ldots, i+1\}$. Our current basis for $\Lambda_{[i+1]}$ is $\mathbf{B}_{[i+1]}$, and by Corollary 2.4.3 there exists a basis $\mathbf{B}'_{[i+1]}$ of $\Lambda_{[i+1]}$ such that $\mathbf{B}'_{[i+1]} = (\mathbf{w} \cdots) = \mathbf{B}_{[i+1]} \cdot \mathbf{U}'$ for some $\mathbf{U}' \in \mathrm{GL}_{n-i}(\mathbb{Z})$ and has $\|\mathbf{w}\| = \lambda_1(\Lambda_{[i+1]})$. If we construct $\hat{\mathbf{U}}$ as

$$\hat{\mathbf{U}} = \begin{pmatrix} \mathbf{I}_i & \mathbf{0} \\ \mathbf{0} & \mathbf{U}' \end{pmatrix} \in \mathrm{GL}_n(\mathbb{Z}),$$

and let $\hat{\mathbf{B}} = \mathbf{B} \cdot \hat{\mathbf{U}} = (\hat{\mathbf{b}}_1 \cdots \hat{\mathbf{b}}_n)$, then $\hat{\mathbf{b}}_j = \mathbf{b}_j$ for $j \in \{1, \ldots, i\}$ and $\Lambda_{[1]}, \ldots, \Lambda_{[i+1]}$ are unchanged. We therefore have $\left\|\mathbf{b}_j^*\right\| = \lambda_1(\Lambda_{[j]})$ for $j \in \{1, \ldots, i\}$ by assumption and $\hat{\mathbf{b}}_{i+1}^* = \pi_{i+1}(\hat{\mathbf{b}}_{i+1}) = \mathbf{w}$ so that $\left\|\hat{\mathbf{b}}_{i+1}^*\right\| = \lambda_1(\Lambda_{[i+1]})$ as required. Applying Algorithm 2 to $\hat{\mathbf{B}}$ alters neither $\Lambda_{[i]}$ or $\mathbf{b}_i^*$ for any i, so we conclude. □

The proof of [Kan83, Prop. 1] is constructive; it returns a basis $\mathbf{B}$ with $\mathbf{w}$ as its shortest vector. Therefore, the ability to find shortest vectors in lattices of rank n and below is sufficient to find an HKZ reduced basis. In particular one must solve SVP on $\Lambda_{[1]}, \ldots, \Lambda_{[n]}$ exactly once each. For the purposes of this thesis, HKZ reduction is too powerful a notion to always achieve. For an example of its strength constrast the guarantees it gives vs. δ-LLL reduction.

Theorem 2.4.5 ([SE94, Thm. 1])**.** *If $\delta \in (1/4, 1)$ and $\mathbf{B} \in \mathbb{R}^{d \times n}$ is a δ-LLL reduced basis for $\Lambda = \Lambda(\mathbf{B})$, let $\alpha = (\delta - 1/4)^{-1}$, then for all i we have*

$$\alpha^{1-i} \leq \|\mathbf{b}_i\|^2 \cdot \lambda_i(\Lambda)^{-2} \leq \alpha^{n-1}.$$

Theorem 2.4.6 ([LLS90, Thm. 2.1])**.** *If $\mathbf{B} \in \mathbb{R}^{d \times n}$ is an HKZ reduced basis for $\Lambda = \Lambda(\mathbf{B})$, then for all i we have*

$$\frac{4}{i+3} \leq \|\mathbf{b}_i\|^2 \cdot \lambda_i(\Lambda)^{-2} \leq \frac{i+3}{4}.$$

The generalisation of HKZ given in [Sch87] is to 'block' Korkine–Zolotarev reduction. For this notion one considers blocks, or projected sublattices of a certain size, rather than necessarily always continuing them to the end of the basis. This allows one a parameter, throughout this thesis β, to parametrise the strength of the reduction.

Definition 2.4.5. Let $\mathbf{B} \in \mathbb{R}^{d \times n}$ define $\Lambda = \Lambda(\mathbf{B})$ and let $\beta \in \{2, \ldots, n\}$. We say $\mathbf{B}$ is a β Block Korkine–Zolotarev (BKZ-β) reduced basis of Λ if we have

- $\mathbf{B}$ is size reduced by Definition 2.3.2, and
- for all $i \in \{1, \ldots, n\}$, recalling Definition 2.2.23, $\|\mathbf{b}_i^*\| \leq \lambda_1(\Lambda_{[i:\min\{i+\beta, n+1\}]})$.

This definition says that each block, i.e. each $\Lambda_{[i,\min\{i+\beta:n+1\}]}$, is HKZ reduced, even if Λ is not. We have given the definition of [SE94, Sec. 5] except we allow $\beta = n$ and require the trivially satisfied $i = n$ case, so that it may coincide with the original HKZ definition. An HKZ reduced basis is a BKZ-β reduced basis for any β since $\|\mathbf{b}_i^*\| = \lambda_1(\Lambda_{[i]}) \leq \lambda_1(\Lambda_{[i:\min\{i+\beta,n+1\}]})$ and hence BKZ-β reduced bases exist by Lemma 2.4.4. Note that although one is now only required to solve SVP on a rank β or smaller lattice, at least to satisfy the second condition of Definition 2.4.5 for a given i, it is not enough to do this once per block, as altering overlapping blocks may violate previously satisfied conditions.

At the other end of the spectrum from BKZ-n reduction being equivalent to HKZ reduction we have BKZ-2 reduction. This is equivalent to the $\delta = 1$ case of LLL

reduction by [SE94, Thm. 5]. However, as previously mentioned, no proof of polynomial complexity is known for 1-LLL reduction. In [Sch87, Thm. 3.2], by considering BKZ reduction only on disjoint blocks in the lattice basis, an algorithm that improves upon the guarantees of LLL reduction is shown to run in polynomial time provided we take the slightly sublogarithmic $\beta \in O(\log n / \log\log n)$. If the SVP oracle is replaced with a lattice sieve (see Section 2.5) then we may take $\beta \in O(\log n)$.

The algorithm [SE91, SE94] we are interested in instead foregoes a polynomial time running proof (for any β) and instead achieves BKZ-β reduction and performs well in practice. We introduce the BKZ algorithm as Algorithm 7. It makes use of LLL as a subroutine, and an oracle O_{SVP} that solves SVP on an input lattice. There is a

Algorithm 7 The BKZ algorithm.

Require: Basis $\mathbf{B} \in \mathbb{R}^{d \times n}$, block size $\beta \in \{2, \ldots, n\}$, parameter $\delta \in (1/2, 1)$

1: **procedure** BKZ($\mathbf{B}, \beta, \delta$)
2: $\quad\mathbf{B} \leftarrow \mathrm{LLL}(\mathbf{B}, \delta)$
3: $\quad$`clean` $=$ `True`
4: $\quad i = 1$
5: $\quad$**while** $i < n$ **do**
6: $\quad\quad j \leftarrow \min\{i + \beta, n + 1\}$
7: $\quad\quad \mathbf{w}_{[i:j]} = v_i\pi_i(\mathbf{b}_i) + \cdots + v_{j-1}\pi_i(\mathbf{b}_{j-1}) \leftarrow O_{\mathrm{SVP}}(\Lambda_{[i:j]})$
8: $\quad\quad$**if** $\|\mathbf{w}_{[i:j]}\| < \|\mathbf{b}_i^*\|$ **then**
9: $\quad\quad\quad$`clean` $=$ `False`
10: $\quad\quad\quad \mathbf{w} = v_i\mathbf{b}_i + \cdots + v_{j-1}\mathbf{b}_{j-1}$
11: $\quad\quad\quad \mathbf{B}_g = (\mathbf{b}_1 \cdots \mathbf{w}\, \mathbf{b}_i \cdots \mathbf{b}_n)$
12: $\quad\quad\quad \mathbf{B} \leftarrow \mathrm{LLL}(\mathbf{B}_g, \delta)$ and remove the $\mathbf{0}$ column
13: $\quad\quad i \leftarrow i + 1$
14: $\quad\quad$**if** `clean` $=$ `False` and $i = n$ **then**
15: $\quad\quad\quad i = 1$
$\quad$**return** $\mathbf{B}$

slight restriction on the range of δ not present before which is an artefact of [SE94] using a floating point version of LLL. In any case the implementations we consider in this work [dt21a, dt21b] set $\delta = 0.99$, and if δ is omitted we assume this value. We also implicitly assume that the changes to $\mathbf{B}^*$ are known from the calls to LLL. Finally, we note that the size reduction requirement of LLL reduced bases means that running LLL on a non full rank lattice will produce a full rank output basis and some $\mathbf{0}$ columns. This is used on Line 12 to transform the generating set $\mathbf{B}_g$ for the lattice $\Lambda = \Lambda(\mathbf{B})$ back into a basis, and is discussed in more detail in [SE94, Sec. 4].

We call the variable i taking values $1, \ldots, n-1$ a *tour* of BKZ. We can see that Algorithm 7 continues solving SVP in the necessary blocks until some tour completes

such that no insertions, the name given to operations on Line 11, are made. The basis returned will therefore be BKZ-β reduced. We note that often the $\mathbf{w}_{[i:j]}$ of Line 7 is said to be inserted, which implicitly describes the insertion of Line 11.

In its original form Algorithm 7 is not known to terminate in a polynomial number of tours in n. By examining a slight variant of Algorithm 7 called BKZ', in [HPS11, Thm. 1] it is proven that BKZ' can be terminated after a polynomial number of tours in n while still ensuring a short first basis vector. This notion of early termination, often called 'early abort', was one of many practical improvements given in [CN11] which proposes a variant of Algorithm 7. The variant of [CN11], along with its subsequent improvements, is implemented in e.g. [dt21a, dt21b]. A common value for the number of tours used in practice before aborting is 16, given in [Alb17, Sec. 2.5] following an analysis of [Che13, Fig. 4.6]. More recently [LN20] gave a slightly smaller polynomial for the number of required tours, along with slightly better bounds on the lengths of the returned basis vectors. The following specialised version of the theorem statement can be achieved by applying the proper preprocessing to a basis and applying the methods of [HPS11].

Theorem 2.4.7 ([LN20, Thm. 2] specialised)**.** *For γ_β the Hermite constant in dimension β and $\mathbf{B} \in \mathbb{R}^{d\times n}$ a basis that has had*

$$\Theta\left(\frac{n^2}{\beta^2}\cdot \log n\right)$$

tours of BKZ-β reduction applied to it, we have that

$$\|\mathbf{b}_1\| \leq \gamma_\beta^{f(n,\beta)} \cdot \mathrm{vol}(\Lambda)^{1/n}, \text{ where } f(n,\beta) = \frac{n-1}{2(\beta-1)} + \frac{\beta(\beta-2)}{2n(\beta-1)}.$$

Given that an α-HSVP oracle called linearly many times can solve α^2-SVP [Lov86, p. 25], we have the following lemma.

Lemma 2.4.8. *Let γ_β denote the Hermite factor in dimension β and C_β the complexity of solving SVP on a rank β lattice, then for a lattice Λ described by input basis $\mathbf{B}'$, one can output another basis $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n)$ in complexity* $\mathrm{poly}(n) \cdot C_\beta$ *such that*

- $\|\mathbf{b}_1\| \leq \gamma_\beta^{2\cdot f(n,\beta)} \cdot \lambda_1(\Lambda)$*, and*
- $\|\mathbf{b}_1\| \leq \gamma_\beta^{f(n,\beta)} \cdot \mathrm{vol}(\Lambda)^{1/n}$*,*

for $f(n,\beta) = \dfrac{n-1}{2(\beta-1)} + \dfrac{\beta(\beta-2)}{2n(\beta-1)}$.

Although the closed form is complex, if we use Minkowski's $\gamma_n \leq n/4 + 1$ for $n \geq 1$ [Ngu10], we have $\gamma_\beta < \beta$ for $\beta \geq 2$ and can show that $\beta^{f(n,\beta)}$ is decreasing until $\beta \approx n$. We may also compare this to Lemma 2.4.1 and below by setting $\beta = 2$ and letting $\delta \to 1$. Indeed, recalling $\gamma_2 = 2/\sqrt{3}$ the bounds are identical in this case, which we expect since BKZ-2 reduction is 1-LLL reduction. Note that Lemma 2.4.8 does not give a polynomial time running proof for *full* BKZ-2 reduction, it merely says that one can terminate after a polynomial number of tours (and so potentially not be fully BKZ-2 reduced) and achieve the same upper bound on $\|\mathbf{b}_1\|$.

The variant of BKZ we will be most concerned with in this thesis is called Progressive BKZ. This variant was introduced in [AWHT16] and allows the block size β to alter during its execution. It is often used as a form of preprocessing, where a certain number of tours are run for growing block sizes, e.g. one tour of BKZ-β for $\beta \in \{2, \ldots, \beta'\}$ for some upper bound β'. If one requires a $\mathbf{b}_1$ shorter than a known bound, such as when solving α-HSVP, one can also not set an upper bound and run tours of BKZ with larger and larger block sizes, until a solution is returned or some termination condition reached.

2.4.2 Lattice reduction heuristics

We have seen two lemmas, Lemma 2.4.1 and Lemma 2.4.8, that describe the *guarantees* we are given by LLL after termination and BKZ after a polynomial number of tours. To show the usefulness of these worst case results it is enough to exhibit a basis of some lattice that, after reduction, has $\mathbf{b}_1$ of the maximum length allowed by them. In cryptography average case results are often more useful. For example, imagine a finite class of rank n lattices. If $n > 1$ then they each have infinitely many bases, but imagine a canonical basis $\mathbf{B}_c$ for each exists and can be efficiently calculated, and that we apply lattice reduction to this canonical basis. Assume for exactly one lattice in our finite class we have $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n) \leftarrow \mathrm{LLL}(\mathbf{B}_c, 3/4)$ where $\|\mathbf{b}_1\| = 2^{(n-1)/2} \cdot \lambda_1(\Lambda(\mathbf{B}))$, but for all other lattices $\mathbf{B} \leftarrow \mathrm{LLL}(\mathbf{B}_c, 3/4)$ is such that $\mathbf{b}_1$ is an SVP solution. We would hope no cryptosystem is built from the hardness of solving SVP for a uniform lattice in this class solely because of the worst case bounds. This is what we aim to capture by studying the average case performance of lattice reduction algorithms. For some sampling distribution over some class of lattices (and possibly also over their bases, though this is trickier), what is the expected performance of a given lattice reduction algorithm in terms of e.g. the length of the first basis vector?

We note that since the introduction of the LLL algorithm it has been experimentally observed that on many classes of lattices the first basis of vector of the output basis is

much smaller than predicted by e.g. Lemma 2.4.1 [NS06], and similarly for BKZ [GN08b]. To discuss their findings, we introduce the following quantity of a basis.

Definition 2.4.6 (Root Hermite Factor)**.** Given a basis $\mathbf{B} \in \mathbb{R}^{d\times n}$ for lattice $\Lambda = \Lambda(\mathbf{B})$ the Hermite factor of this lattice *basis* is

$$h(\mathbf{B}) = \frac{\|\mathbf{b}_1\|}{\mathrm{vol}(\Lambda)^{1/n}},$$

so that if $\mathbf{b}_1$ is an SVP solution, $h(\mathbf{B}) = h(\Lambda)$. We define the *root* Hermite factor of the basis as

$$\delta(\mathbf{B}) = \left(\frac{\|\mathbf{b}_1\|}{\mathrm{vol}(\Lambda)^{1/n}}\right)^{1/(n-1)} = h(\mathbf{B})^{1/(n-1)}.$$

If the root Hermite factor of some basis is $\delta(\mathbf{B})$ we have $\|\mathbf{b}_1\| = \delta(\mathbf{B})^{n-1} \cdot \mathrm{vol}(\Lambda)^{1/n}$. We note that in some works the n^{th} root is considered, but we follow [Vir21, Sec. 1.4.1.2] and arguments therein for why the $n-1^{\text{th}}$ root is more natural.[6] From Lemma 2.4.1 and below we know in the worst case that a δ-LLL reduced basis $\mathbf{B}$ will have $h(\mathbf{B}) \leq (4/3)^{(n-1)/4}$ as $\delta \to 1$, and therefore root Hermite factor $\delta(\mathbf{B}) \leq (4/3)^{1/4} \approx 1.07$. In [NS06] it is instead shown that for various classes of lattices the root Hermite factor for such LLL reduction is $\delta(\mathbf{B}) \approx 1.02$. We therefore expect $\|\mathbf{b}_1\| \approx 1.02^{(n-1)} \cdot \mathrm{vol}(\Lambda)^{1/n}$ after e.g. 0.999-LLL reduction. While the two constants 1.02 and 1.07 are similar, since we take their $(n-1)^{\text{th}}$ power, the difference is important in practice. Similarly [GN08b] show an experimental root Hermite factor of 1.013 for BKZ-20 where one would expect 1.034. When we say 'for BKZ-20' we implicitly mean 'for a basis of a lattice from a particular class reduced by BKZ-20'. Following [NS06, GN08b] we note that, although solving α-HSVP provides a solution to α^2-SVP, if the lattice in question follows the Gaussian heuristic then we expect $\lambda_1(\Lambda) = \sqrt{n/2\pi e} \cdot \mathrm{vol}(\Lambda)^{1/n}$ and therefore

$$\|\mathbf{b}_1\| = \delta(\mathbf{B})^{n-1} \cdot \mathrm{vol}(\Lambda)^{1/n} = \sqrt{2\pi e/n} \cdot \delta(\mathbf{B})^{n-1} \cdot \lambda_1(\Lambda).$$

That is, we solve α-HSVP where $\alpha = \delta(\mathbf{B})^{n-1}$ and also α'-SVP, where α and α' are equal up to sublinear factors. More generally, whenever $\mathrm{vol}(\Lambda)^{1/n} \leq \lambda_1(\Lambda)$ we have $\alpha' \leq \alpha$.

In [Che13] the asymptotic expression

$$\lim_{n\to\infty} \delta(\mathbf{B}) = \left(\frac{\beta}{2\pi e} \cdot (\pi\beta)^{1/\beta}\right)^{1/2(\beta-1)} \tag{2.15}$$

[6] In Chapter 3 this leads to a slight rederivation of some formulas compared to the original.

is given for BKZ-β reduction on random lattice bases $\mathbf{B}$ of rank n in the sense of [GM03]. This formula is frequently used in the following way, first let $\beta \geq 50$ and $n \geq C \cdot \beta$ for some constant C bigger than say 2. Then for an 'ordinary' lattice basis $\mathbf{B}$ that we encounter in cryptography, say a basis of a modular lattice of the form introduced in (2.2), if we perform $\mathbf{B} \leftarrow \mathrm{BKZ}(\mathbf{B}, \beta)$ then we assume $\delta(\mathbf{B})$ is given by the righthand side of (2.15).

Definition 2.4.7. For $\beta \geq 50$, let δ_β denote the estimate given by (2.15) for the root Hermite factor after BKZ-β reduction.

Note that for a lattice following the Gaussian heuristic we can therefore expect to solve α-SVP and α-HSVP for $\alpha \approx \delta_\beta^{n-1}$. The $\beta \geq 50$ requirement is both a sanity check, as δ_β is increasing for $\beta \leq 36$, against intuition (weaker lattice reduction giving better approximations – I wish) and also to try and ensure the projected sublattices that BKZ operates on have a distribution similar to random lattices. It has been experimentally observed that this randomness criterion is not the case for block sizes below 30 [GN08b], see e.g. [CN11, Fig. 2].

Another way to measure the quality of a basis after reduction is by its Gram-Schmidt norms, and how quickly they decrease. Here there exists a heuristic called the Geometric Series Assumption [Sch03], we define the basis profile of some basis $\mathbf{B}$ and then introduce this heuristic.

Definition 2.4.8 (Basis profile)**.** For a basis $\mathbf{B} \in \mathbb{R}^{d \times n}$ of lattice $\Lambda = \Lambda(\mathbf{B})$ we define its basis profile as the list of norms of the Gram–Schmidt vectors of $\mathbf{B}^*$, $(\|\mathbf{b}_i^*\|)_{i=1}^n$.

Definition 2.4.9 (Geometric Series Assumption (GSA))**.** The Geometric series assumption states that after lattice reduction the profile of $\mathbf{B}$ is such that

$$\|\mathbf{b}_i^*\| = \gamma^{i-1} \cdot \|\mathbf{b}_1\|$$

for some constant $\gamma \in (0, 1)$.

There is the implicit notion that after 'stronger' lattice reduction, for example using larger β, the constant factor γ gets closer to 1. The GSA is known to be accurate [CN11, YD17] when $\beta \geq 50$, β is sufficiently smaller than n, and we are not considering the so called head or tail of the basis. The head is some number of the first indices, $i \in \{1, \ldots, h\}$, and the tail some number of the final indices, $i \in \{t, \ldots, n\}$.

We note that given Definition 2.4.6 we have the following two lemmata.

Lemma 2.4.9 ([Vir21, Lem. 7], slightly adapted)**.** *If $\mathbf{B} \in \mathbb{R}^{d \times n}$ is a basis that follows the GSA, we have the relationship $\delta(\mathbf{B}) = 1/\gamma^{1/2}$.*

Lemma 2.4.10 ([Vir21, Lem. 9])**.** *If* $\mathbf{B} \in \mathbb{R}^{d\times n}$ *is given by* $\mathbf{B} \leftarrow \mathrm{BKZ}(\mathbf{B}, \beta)$*, and* $\mathbf{B}$ *follows the GSA and the Gaussian heuristic, we have the relationship*

$$\gamma(\beta) = \left((\pi\beta)^{1/\beta} \cdot \frac{\beta}{2\pi e}\right)^{1/(1-\beta)}.$$

It is therefore shown that by assuming the GSA and the Gaussian heuristic one can rederive the righthand side of (2.15).

2.4.3 Solving LWE using lattice reduction

As introduced in Definition 2.2.35 and below, the LWE problem is closely related to the lattice $\Lambda_q(\mathbf{A})$ when the columns of $\mathbf{A}$ are formed from the first part of some LWE samples. In this section we will discuss how the lattice reduction algorithms of Section 2.4.1, in particular BKZ, may be used in combination with the lattice reduction heuristics of Section 2.4.2 to mount the primal attack [Kan87, BG14b] against LWE instances. There are other lattice reduction based attacks against LWE instances, such as the decoding attack [LP11], and the dual attack [MR09, Alb17]. The first of these alternatives generalises [Bab86], and the second works in the (scaled) dual lattice of $\Lambda_q(\mathbf{A})$, namely $\Lambda_q^\perp(\mathbf{A})$. There are also purely algebraic [AG11, ACF+15] and combinatorial [BKW03, GJMS17] style attacks.

The idea of the primal attack is to embed a concatenation of the errors of some LWE samples as a short vector into a lattice, and then find said short vector using lattice reduction. It can be thought of as converting a BDD instance into a uSVP instance. Concretely we number m LWE samples as $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s}\rangle + e_i \bmod q)_{i\le m}$ for fixed $\mathbf{s} \in \mathbb{Z}_q^n$ and i.i.d. $\mathbf{a}_i \leftarrow \mathsf{U}(\mathbb{Z}_q^n)$ and $e_i \leftarrow \chi$. We form $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ such that the i^{th} column of $\mathbf{A}$ is $\mathbf{a}_i$ and assume we can permute the columns of $\mathbf{A}$ such that $\mathbf{A} = (\mathbf{A}_1|\mathbf{A}_2)$ has $\mathbf{A}_1 \in \mathrm{GL}_n(\mathbb{Z}_q)$. After permuting the indices so that we have an invertible $\mathbf{A}_1$ we let $\mathbf{b} = (\langle \mathbf{a}_1, \mathbf{s}\rangle + e_1 \bmod q, \ldots, \langle \mathbf{a}_m, \mathbf{s}\rangle + e_m \bmod q)^t \in \mathbb{Z}_q^m$ and $\mathbf{e} = (e_1, \ldots, e_m)^t \in \mathbb{Z}^m$. We recall from (2.2) that $\Lambda_q(\mathbf{A})$ then has volume q^{m-n} and an $m \times m$ integer basis $\mathbf{B}_\mathbf{A}$. Implicitly considering $\mathbf{A}$ and $\mathbf{s}$ over the integers, it also has $\mathbf{A}^t \cdot \mathbf{s} + q \cdot \mathbf{x}$ for any $\mathbf{x} \in \mathbb{Z}^m$ as lattice vectors. Let $\mathbf{b}' \in \mathbb{Z}^m$ be any vector in the class of $\mathbf{b} \in \mathbb{Z}_q^m$ and set the target $\mathbf{t} = \mathbf{b}'$ for a BDD instance over $\Lambda_q(\mathbf{A})$. If $\|\mathbf{e}\| \le \lambda_1(\Lambda_q(\mathbf{A}))/2$, then we have a γ-BDD instance with $\gamma \le 1/2$. Indeed, for any choice of $\mathbf{b}'$ there exists an $\mathbf{x} \in \mathbb{Z}^m$ such that we have $\mathbf{b}' - \mathbf{A}^t \cdot \mathbf{s} + q \cdot \mathbf{x} = \mathbf{e}$ over the integers. In practice we choose $\mathbf{b}'$ with balanced representatives. Solving this BBD instance returns $\mathbf{A}^t \cdot \mathbf{s} + q \cdot \mathbf{x}$ from which $\mathbf{s} \in \mathbb{Z}_q^n$ can be recovered, since $\mathbf{A}^t$ has full column rank over $\mathbb{Z}_q$.

The Kannan embedding[7] primal attack [Kan87] creates a basis from $\mathbf{B_A}$ and $\mathbf{b}'$ in which $\mathbf{e}$ is expected to be a uSVP solution, which is then found. Finding $\mathbf{e}$ allows one to solve $\mathbf{b} - \mathbf{e} = \mathbf{A}^t \cdot \mathbf{s}$ over $\mathbb{Z}_q$ for $\mathbf{s}$ and therefore also solve the LWE instance. The primal basis $\mathbf{B}_p \in \mathbb{Z}^{(m+1)\times(m+1)}$ is full rank and constructed as

$$\mathbf{B}_p = \begin{pmatrix} \mathbf{B_A} & \mathbf{b}' \\ \mathbf{0}^t & t \end{pmatrix}. \tag{2.16}$$

The parameter $t \in \mathbb{N} \setminus \{0\}$ is called the embedding parameter, and we have $\mathrm{vol}(\mathbf{B}_p) = q^{m-n} \cdot t$. So long as we have an invertible $\mathbf{A}_1$ we may use any number of samples $m \geq n$ by removing columns of $\mathbf{A}_2$ before forming $\mathbf{B_A}$ and removing entries of $\mathbf{b}'$ from $\mathbf{B}_p$ as appropriate. While using fewer samples reduces the dimension of $\mathbf{B}_p$ to a minimum of $n+1$, it also decreases the volume, so we cannot simply take $m = n$ in the primal attack. The vector we are looking for is

$$\begin{pmatrix} \mathbf{e} \\ t \end{pmatrix} = \begin{pmatrix} -\mathbf{A}^t \cdot \mathbf{s} + q \cdot \mathbf{x} \\ 0 \end{pmatrix} + \begin{pmatrix} \mathbf{b}' \\ t \end{pmatrix} = \mathbf{B}_p \cdot \begin{pmatrix} \mathbf{v} \\ 1 \end{pmatrix}$$

for some $\mathbf{v} \in \mathbb{Z}^m$. The expected length of $\left\|(\mathbf{e}|t)^t\right\|^2$ is $t^2 + \mathbb{E}\left[\sum_{i=1}^m \chi^2\right] = t^2 + m\mathbb{E}\left[\chi^2\right]$. Given a known χ we can calculate this quantity.

Until 2016 one would now calculate (possibly an approximation of) the length of the shortest non zero vector that is linearly independent to $(\mathbf{e}|t)^t$ and hopefully have an α-uSVP instance over $\Lambda(\mathbf{B}_p)$ with a large α, see e.g. [GN08b].

However, a new success condition was introduced in [ADPS16] and experimentally verified in [AGVW17, BMW19, DDGR20, PV21]. It assumes that $\mathbf{B}_p$ behaves under lattice reduction like a $\mathbf{B_A}$ for uniform $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, i.e. that the final column introduced in $\mathbf{B}_p$ has no effect in this regard. It can then determine, using the heuristics of Section 2.4.2, the basis profile of $\mathbf{B}_p$ after e.g. BKZ-β reduction. For the rest of this section we let $\mathbf{e} = (\mathbf{e}|t)^t \in \mathbb{Z}^{m+1}$, $\mathbf{B} = \mathbf{B}_p$, and $d = m+1$.

If in a tour of BKZ-β we consider the final full block $\Lambda_{[d-\beta+1]}$ then, prior to any information about $\mathbf{e}$ or its projections being 'visible' to the BKZ algorithm, we have $\mathbf{w}_{[d-\beta+1]} \leftarrow O_{\mathrm{SVP}}(\Lambda_{[d-\beta+1]})$ being inserted in this final full block. After this insertion we have that $\mathbf{w}_{[d-\beta+1]} = \mathbf{b}^*_{d-\beta+1}$ is an SVP solution in this block. If subsequently, as more tours of BKZ-β are run, the length of $\mathbf{b}^*_{d-\beta+1}$ increases – recall that stronger lattice reduction means the basis profile lengths decrease less quickly – then it may become the case that $\pi_{d-\beta+1}(\mathbf{e})$ becomes an SVP solution in the final full block. If

[7] In this thesis we deal with uniform secrets, and therefore do not use other embeddings [BG14b].

so it will be inserted into the basis in the next BKZ tour. After this insertion it is argued [AGVW17] that Algorithm 1 on $\mathbf{b}_{d-\beta+1}$ is sufficient to recover $\mathbf{e}$, so we consider ensuring $\|\pi_{d-\beta+1}(\mathbf{e})\| < \left\|\mathbf{b}^*_{d-\beta+1}\right\|$ a win condition.

We therefore require two ingredients, estimates for $\|\pi_i(\mathbf{e})\|$ and for $\|\mathbf{b}^*_i\|$ after BKZ-β reduction. The latter comes from the root Hermite factor and the GSA. For the projections of $\mathbf{e}$ one first writes $\mathbf{e}$ in terms of the orthonormal basis $\mathbf{B}^\circ$ using (2.5), and can therefore represent its square length as follows

$$\mathbf{e} = \mathbf{B}\cdot\mathbf{v} = \mathbf{B}^\circ\cdot\mathbf{R}\cdot\mathbf{v} = \mathbf{B}^\circ\cdot\mathbf{v}^\circ \Rightarrow \|\mathbf{e}\|^2 = \sum_{i=1}^{d}(v_i^\circ)^2.$$

If we consider $\pi_i(\mathbf{e})$ then we immediately see $\|\pi_i(\mathbf{e})\|^2 = \sum_{j=i}^{d}(v_j^\circ)^2$. Note that since we fix t we always know that $v_d = 1$ and therefore that $v_d^\circ = \langle \mathbf{b}_d, \mathbf{b}_d^\circ\rangle$. It is therefore enough to know a distribution for the $(v_i^\circ)^2$ with $i \in [m]$. The assumption used in [ADPS16] is given in the following definition.

Definition 2.4.10 (Projection Heuristic)**.** If χ is a centred random variable over $\mathbb{Z}$ with variance σ^2, $\mathbf{x} \leftarrow \chi^m$, and $\mathbf{B}$ is a lattice basis e.g. one encountered during Algorithm 7, then writing $\mathbf{x} = \mathbf{B}^\circ\cdot\mathbf{v}^\circ$ we assume each $(v_i^\circ)^2$ is i.i.d. sampled as $(v_i^\circ)^2 \leftarrow \mathsf{N}(0,\sigma^2)$. In particular the expected value of $\|\pi_i(\mathbf{x})\|^2$ is $(m-i+1)\cdot\sigma^2$.

For centred χ we have

$$\mathbb{V}[\chi] = \mathbb{E}\left[\chi^2\right] - 0 \Rightarrow t^2 + m\cdot\mathbb{V}[\chi] = t^2 + m\cdot\mathbb{E}\left[\chi^2\right],$$

so that the expected value of $\|\mathbf{e}\|^2$ is always given by $t^2 + m\cdot\sigma^2$. The assumption above says furthermore that the expected value of $\|\pi_i(\mathbf{e})\|^2$ is given by $t^2 + (m-i+1)\cdot\sigma^2$. For χ representing the discrete Gaussian distribution over $\mathbb{Z}$ and $t = 1$ the accuracy of this assumption after LLL reduction is experimentally shown in [AGVW17, Fig. 2].

To concretise, t is usually set as 1 in practice and it is assumed that $(d-i+1)\cdot\sigma^2$ is a good approximation to $1^2 + (m-i+1)\cdot\sigma^2$. This is often the case since for successful $i = d-\beta+1$ the ratio $(m-i+1)/(d-i+1)$ will be close to 1, and practical schemes often take small constant σ so that $1 \approx \sigma^2$. The expected value of $\|\pi_i(\mathbf{e})\|$ is approximated as the square root of the above approximation, $\sqrt{d-i+1}\cdot\sigma$. This introduces a small error that disappears as $d-i+1 \to \infty$ [PV21, App. B]. We therefore approximate $\|\pi_{d-\beta+1}(\mathbf{e})\|$ as $\sqrt{\beta}\cdot\sigma$.

To calculate $\left\|\mathbf{b}^*_{d-\beta+1}\right\|$ after BKZ-β reduction, using the GSA and δ_β, we have

$$\left\|\mathbf{b}^*_{d-\beta+1}\right\| = \|\mathbf{b}_1\|\cdot\gamma^{d-\beta} = \delta_\beta^{d-1}\cdot\mathrm{vol}(\Lambda)^{1/d}\cdot\left(\delta_\beta^{-2}\right)^{d-\beta} = \delta_\beta^{2\beta-d-1}\cdot\mathrm{vol}(\Lambda)^{1/d}.$$

We therefore conclude with the condition,

$$\begin{aligned}\sqrt{\beta} \cdot \sigma &< \delta_\beta^{2\beta-d-1} \cdot \mathrm{vol}(\Lambda)^{1/d} \\ &= \delta_\beta^{2\beta-m-2} \cdot q^{(m-n)/(m+1)}.\end{aligned} \tag{2.17}$$

We can control the number of samples $m \geq n$, and the block size β, so given a cost model for BKZ-β reduction on a dimension $m+1$ lattice we can attempt to minimise the cost of the primal attack while satisfying (2.17).

2.5 Lattice sieves

Lattice sieves (often in this thesis just 'sieves' or 'a sieve') are algorithms that form large lists of lattice vectors and then combine elements of these lists to obtain new and shorter lattice vectors. Lattice sieves fall into two broad categories, provable and heuristic. In this thesis we focus on the heuristic category and their analysis, and assume full rank lattices. Lattice sieves are often used to solve SVP type problems, for example within lattice reduction algorithms like BKZ, where they realise the O_{SVP} oracle (or an approximate variant). As well as outputting (approximate) SVP solutions, more generally they terminate with a list containing many short lattice vectors which is important in applications such as Section 2.3.3.

2.5.1 Basic ideas

A sieve combines lattice vectors by making use of the group operation, i.e. addition or subtraction. Since $\alpha \cdot \mathbf{u}$ for $\alpha \in \mathbb{Z}$ and $\mathbf{u} \in \Lambda$ can only give a shorter lattice vector than $\mathbf{u}$ in the trivial $\alpha = 0$ case, we focus on combining $k \geq 2$ lattice vectors at once. For now we focus on $k = 2$. The fundamental question is whether for two vectors $\mathbf{u}, \mathbf{v} \in \Lambda$ we have that $\min\{\|\mathbf{u}+\mathbf{v}\|, \|\mathbf{u}-\mathbf{v}\|\} < \max\{\|\mathbf{u}\|, \|\mathbf{v}\|\}$. If so we can replace e.g. the longer of $\mathbf{u}$ and $\mathbf{v}$ with the shorter of $\mathbf{u}+\mathbf{v}$ and $\mathbf{u}-\mathbf{v}$.

Definition 2.5.1. Given $\mathbf{u}, \mathbf{v} \in \Lambda$ we say $(\mathbf{u}, \mathbf{v})$ is a reduced pair if

$$\min\{\|\mathbf{u}+\mathbf{v}\|, \|\mathbf{u}-\mathbf{v}\|\} \geq \max\{\|\mathbf{u}\|, \|\mathbf{v}\|\},$$

else they are a reducible pair.

We will write $\|\mathbf{u} \pm \mathbf{v}\|$ to denote $\min\{\|\mathbf{u}+\mathbf{v}\|, \|\mathbf{u}-\mathbf{v}\|\}$. Replacing the longest element in a reducible pair with the shorter of the sum and difference of the pair

is called reducing the pair. We note that in practice one uses inner products and calculates e.g. $\|\mathbf{u}\|^2 = \langle \mathbf{u}, \mathbf{u} \rangle$ to perform these styles of comparison.

If one continues to reduce pairs, then the lengths of the vectors in the list held by the lattice sieve continue to decrease. Some natural questions arise. How does one populate the list to begin with? How large does the list have to be to ensure there are reducible pairs within it? When can we stop reducing pairs and expect to have found unusually short vectors? Here the definition of 'unusually' is somewhat application specific, but one should certainly think of vectors shorter than those that can be found in polynomial time and often we aim to solve SVP almost exactly.

To answer the first question we require an efficient sampling routine, the output of which satisfies some notion of 'well distributed' over the lattice. For example a particularly degenerate sampling procedure that only outputs $\mathbf{u} \in \mathrm{span}_{\mathbb{Z}}(\{\mathbf{b}_1, \mathbf{b}_2\})$ in some $\Lambda = \Lambda(\mathbf{B})$ with $\mathbf{B} \in \mathbb{R}^{n \times n}$ and $n > 2$ is clearly of little use. A notion of 'well distributed' can be achieved by using a randomised variant of Babai's Nearest Plane algorithm due to Klein [Kle00] which has complexity polynomial in n and the size of its input. This outputs a lattice vector sampled from a distribution close to a spherical Gaussian of a sufficiently large variance restricted to the lattice, a detailed analysis is given in [GPV08, Thm. 4.1]. Sampling a list $L \subset \Lambda$ of lattice vectors in this manner has complexity $|L|^{1+o(1)}$.

To answer how large the list must be to ensure that reducible pairs exist, we must first determine the conditions that determine whether a pair is reducible. This is displayed in Figure 2.1. For any $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, provided they are linearly independent, we can consider the plane that they span. If they are not linearly independent then if $\mathbf{u}$ and $\mathbf{v}$ are elements of some lattice it is efficient to find the shortest non trivial lattice vector one can form by adding or subtracting multiples of them from each other. We may rotate this plane so that $\mathbf{u}$ lies on the first axis, as in Figure 2.1. If $\mathbf{v}$ lies in the interior of the shaded region then $\|\mathbf{u} \pm \mathbf{v}\| < \max\{\|\mathbf{u}\|, \|\mathbf{v}\|\}$. In particular, if $\mathbf{v}$ lies in one of the two circles then $\|\mathbf{u} \pm \mathbf{v}\| < \|\mathbf{u}\|$ and if $\mathbf{v}$ lies outside of the two vertical lines then $\|\mathbf{u} \pm \mathbf{v}\| < \|\mathbf{v}\|$. We note that if $\theta(\mathbf{u}, \mathbf{v}) < \pi/3$ then $(\mathbf{u}, \mathbf{v})$ is a reducible pair. If further $\|\mathbf{u}\| = \|\mathbf{v}\|$ then $\theta(\mathbf{u}, \mathbf{v}) < \pi/3$ is an equivalent condition to $(\mathbf{u}, \mathbf{v})$ being a reducible pair. For some $\mathbf{u} \in \mathbb{R}^2$ let $R(\mathbf{u}) = \{\mathbf{x} \in \mathbb{R}^2 \colon (\mathbf{u}, \mathbf{x}) \text{ is a reducible pair}\}$. While it is not quite true that $R(\mathbf{u}) \subset R(\alpha \cdot \mathbf{u})$ for $\alpha \in (0, 1)$ because the circular regions of $R(\mathbf{u})$ are larger, in practice shorter lattice vectors in the list of a sieve will cause far more reductions. Indeed, one can calculate that regardless of the circular regions, narrowing the vertical band of unshaded space as in the $\alpha \cdot \mathbf{u}$ case increases the fraction of Figure 2.1 that is shaded, and also captures more $\mathbf{v}$ that are close to

orthogonal with $\mathbf{u}$. As we shall see in Section 2.5.2, most vectors we encounter in lattice sieves will be almost orthogonal to one another.

From the above discussion we have that $\theta(\mathbf{u}, \mathbf{v}) < \pi/3$ is always a sufficient condition for $(\mathbf{u}, \mathbf{v})$ to be a reducible pair. Therefore the amount of lattice vectors required in our list is upper bounded by the maximum number of points in $\mathbb{R}^n$ one can choose such that the angle between any pair of points is at least $\pi/3$. This value is called the kissing number.

Definition 2.5.2 (Kissing Number, τ_n)**.** The kissing number τ_n of $\mathbb{R}^n$ is

$$\tau_n = \max\{|L| : L \subset \mathbb{R}^n, \theta(\mathbf{x}, \mathbf{y}) \geq \pi/3 \text{ for all } \mathbf{x}, \mathbf{y} \in L\}.$$

This quantity is known exactly for only a few small dimensions [CS99, Mus08] but has asymptotic upper and lower bounds.

Theorem 2.5.1 (Bounds on τ_n)**.** *We have, using the lower bound of [JJP18, Thm. 2] and the upper bound of [KL78], that*

$$(1 + o(1)) \sqrt{\frac{3\pi}{8}} \log\left(\frac{3}{2\sqrt{2}}\right) \cdot n^{3/2} \left(\sqrt{\frac{4}{3}}\right)^n \leq \tau_n \leq 2^{0.401n + o(n)}$$

which can be written as

$$\left(\sqrt{\frac{4}{3}}\right)^{n+o(n)} \leq \tau_n \leq 2^{0.401n+o(n)}.$$

We note here that $2^{0.2075} \approx (4/3)^{1/2}$.

We therefore know something about the list size we require to ensure it contains reducible pairs. As we will see, in practice $|L| \in 2^{0.2075n+o(n)}$, i.e. the lower end of our asymptotic bounds, seems sufficient. Indeed, finding lattices for which significantly more vectors are required could imply progress in the research of sphere packings.

The final question, when we can stop reducing pairs, is both sieve and application specific. But in a model where sieving occurs in iterations, which is the case in the first heuristic lattice sieve [NV08], if in each iteration the maximum length of vectors in the sieve decreases by some geometric factor $\gamma < 1$ then we at least know that the total number of iterations is polynomial. Indeed, let $\Lambda = \Lambda(\mathbf{B})$ with $\mathbf{B} \in \mathbb{R}^{d \times n}$ and recall that $\|\mathbf{A}\|$ represents the length of the longest column vector in $\mathbf{A}$. Then one can use [Kle00] to sample lattice vectors such that their lengths will be shorter than

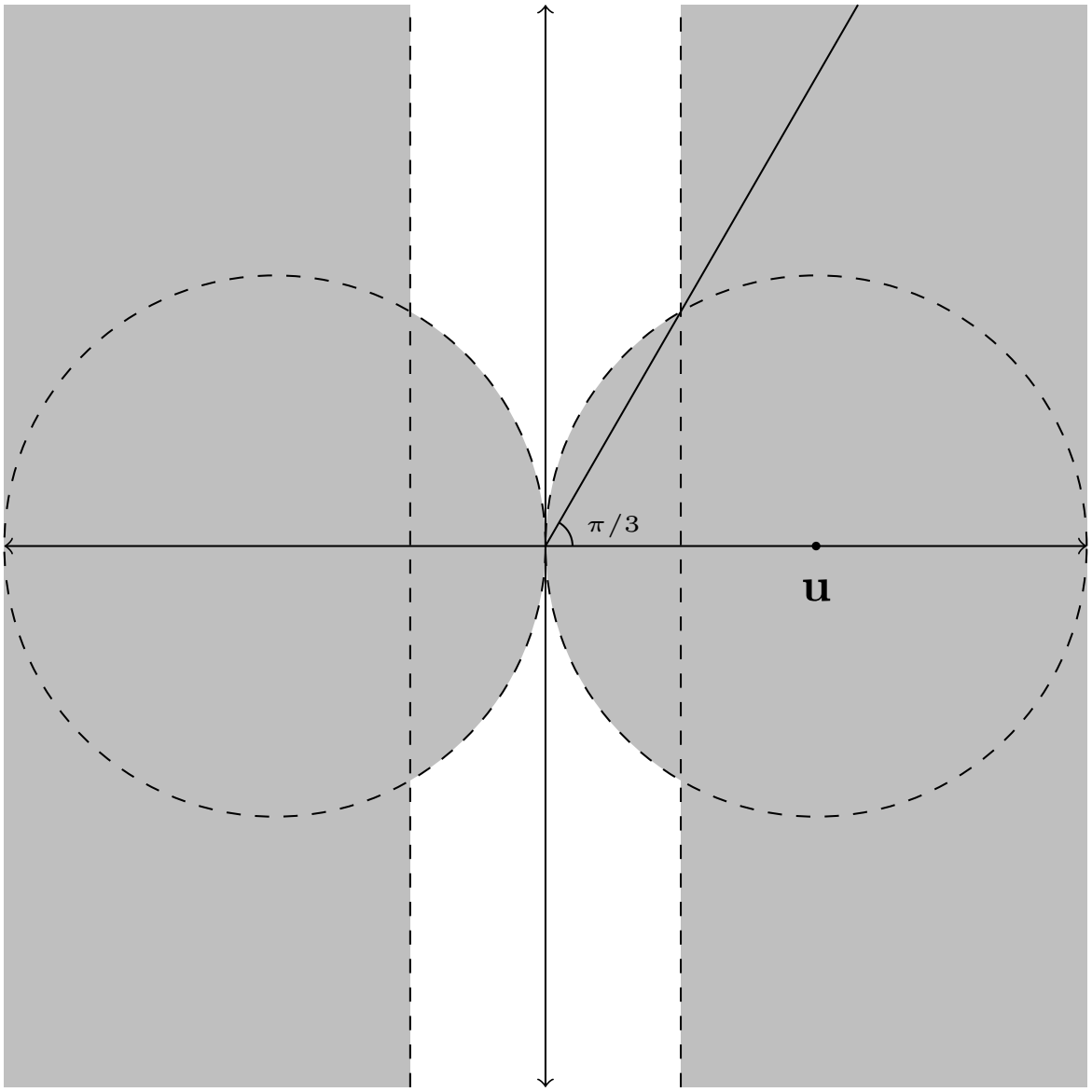


Figure 2.1 Visual depiction of $R(\mathbf{u}) \subset \mathbb{R}^2$, the subset of the plane such that $\mathbf{v} \in R(\mathbf{u})$ implies $(\mathbf{u}, \mathbf{v})$ is a reducible pair. The vertical dashed lines have x coordinates $\mathbf{u}/2$ and $-\mathbf{u}/2$, and the two circles have radius $\|\mathbf{u}\|$ and centres $\mathbf{u}$ and $-\mathbf{u}$.

$\|\mathbf{B}^*\| \cdot \text{poly}(n)$ with overwhelming probability in n by [GPV08, Lem 2.5, Thm. 4.1]. Using LLL as preprocessing we can ensure $\|\mathbf{B}^*\| \in 2^{O(n)} \cdot \lambda_1(\Lambda)$ as the following lemma shows.

Lemma 2.5.2. *Let $\mathbf{B} \in \mathbb{R}^{d \times n}$ be a (3/4)-LLL reduced basis defining $\Lambda = \Lambda(\mathbf{B})$. If there exists an $i \in \{2, \ldots, n\}$ such that $\|\mathbf{b}_i^*\| > 2^{n-i} \cdot \|\mathbf{b}_1\|$ then any $\mathbf{w} \in \Lambda$ such that $\|\mathbf{w}\| = \lambda_1(\Lambda)$ does not require $\mathbf{b}_i, \ldots, \mathbf{b}_n$. That is $\mathbf{w} \in \Lambda(\mathbf{B}')$ for $\mathbf{B}' = (\mathbf{b}_1 \cdots \mathbf{b}_{i-1})$.*

Proof. Assume such an index i exists. For a (3/4)-LLL reduced basis and any index $i \in [n]$ we have $\left\|\mathbf{b}_{i+1}^*\right\| \geq \|\mathbf{b}_i^*\|/2$, therefore $\|\mathbf{b}_i^*\| > 2^{n-i} \cdot \|\mathbf{b}_1\| \Rightarrow \|\mathbf{b}_i^*\|, \ldots, \|\mathbf{b}_n^*\| > \|\mathbf{b}_1\|$. If $\mathbf{w}$ is such that $\|\mathbf{w}\| = \lambda_1(\Lambda)$ then $\|\mathbf{w}\|^2 \leq \|\mathbf{b}_1\|^2 < \|\mathbf{b}_i^*\|^2, \ldots, \|\mathbf{b}_n^*\|^2$. Let

$$\mathbf{w} = \sum_{j=1}^{n} v_j \mathbf{b}_j = \sum_{j=1}^{n} x_j \mathbf{b}_j^*$$

with each $v_j \in \mathbb{Z}$ and $x_j \in \mathbb{R}$. We have $x_j = v_j + v_{j+1}\mu_{j+1,j} + \cdots + v_n \mu_{n,j}$ for all $j \in [n]$, and in particular $x_n = v_n \in \mathbb{Z}$.

Given the orthogonality of the Gram–Schmidt basis we have

$$\|\mathbf{w}\|^2 = \sum_{j=1}^{n} x_j^2 \left\|\mathbf{b}_j^*\right\|^2$$

In particular $x_n^2 \|\mathbf{b}_n^*\|^2 \leq \|\mathbf{w}\|^2$ and therefore, since $x_n = v_n$ is an integer and $\|\mathbf{b}_n^*\| > \|\mathbf{w}\|$, $x_n = v_n = 0$. We therefore have $x_{n-1} = v_{n-1} + v_n \mu_{n,n-1} = v_{n-1}$, and by the same argument, $x_{n-1} = v_{n-1} = 0$. This process continues until we find $x_i = v_i = 0$, and therefore have that $\mathbf{w} = v_1 \mathbf{b}_1 + \cdots + v_{i-1} \mathbf{b}_{i-1}$ as required. □

If any $\mathbf{b}_i^*$ is 'too much' longer than $\mathbf{b}_1$ we may remove $\mathbf{b}_i, \ldots, \mathbf{b}_n$ from the basis and still find a shortest vector as an integer combination of the remaining basis vectors. The maximal allowable length of a Gram–Schmidt vector that does not cause this removal occurs at $i = 2$, and

$$\|\mathbf{b}_2^*\| \leq 2^{n-2} \cdot \|\mathbf{b}_1\| \leq 2^{n-2} \cdot 2^{(n-1)/2} \cdot \lambda_1(\Lambda) \in 2^{O(n)} \cdot \lambda_1(\Lambda),$$

with the final inequality from Lemma 2.4.1. Therefore, polynomially many iterations of the sieve are sufficient, i.e. $\gamma^{p(n)} \cdot 2^{O(n)} \approx 1$ for some polynomial $p(n)$.

Heuristics

Here we introduce four heuristic ideas that are found within the analysis of heuristic lattice sieves. The first deals with the lengths and distribution of the lattice vectors

we are considering, the second with the angles between reducible pairs, and the third with the average case size of $L \subset \Lambda$ required to find reducing pairs. Finally, the fourth heuristic regards the contents of the list upon termination of a lattice sieve. The first heuristic is somehow fundamental, as the others follow from applying mathematical results while assuming it holds.

One of the main theoretical difficulties when analysing provable lattice sieves is determining and controlling the distribution that the lattice vectors in the list L follow. In the original lattice sieve [AKS01] this is accounted for by adding small Gaussian perturbations to make the sieve 'forget' the discrete structure of the lattice, and in subsequent works by ensuring the lattice vectors fall into particular equivalence classes [ASD18, ALSD21]. In heuristic lattice sieving, a line of works that began with [NV08, Sec. 4], instead the vectors considered by the sieve are assumed to be uniformly distributed in some thin annulus.

Definition 2.5.3 (Uniform heuristic)**.** Let $L \subset \Lambda \subset \mathbb{R}^n$ be a list of lattice vectors encountered after some *iteration* of a heuristic lattice sieve. We assume, after the appropriate scaling, that the elements of L are i.i.d. uniformly distributed in

$$\mathsf{ann}_\gamma = \{\mathbf{x} \in \mathbb{R}^n : \gamma \leq \|\mathbf{x}\| \leq 1\},$$

for some fixed $\gamma < 1$.

Like a heuristic version of the perturbation method of [AKS01], the above assumes that we can forget the discrete structure of the lattice, at least until we have found many short vectors, see below. We typically consider this heuristic as $\gamma \to 1$, as this will lead to the smallest required size for L, as well as allow us to argue about uniformly distributed points on S^{n-1}. What an *iteration* means is sieve specific, but it can be thought of for example as performing one double loop over L to look for reducible pairs, and reducing any found in this particular search.

In Section 2.5.2 we discuss geometric figures on the sphere which, in conjunction with Definition 2.5.3 of the uniform heuristic as $\gamma \to 1$, will give us the following heuristic. The intuition is that i.i.d. $\mathbf{u}, \mathbf{v} \leftarrow \mathsf{U}(S^{n-1})$ are such that $\theta(\mathbf{u}, \mathbf{v})$ is concentrated around $\pi/2$. If we restrict to $(\mathbf{u}, \mathbf{v})$ such that $\theta(\mathbf{u}, \mathbf{v}) < \pi/3$ then $\theta(\mathbf{u}, \mathbf{v})$ will be concentrated just below $\pi/3$.

Definition 2.5.4 ($\pi/3$ Heuristic)**.** A reducible pair $(\mathbf{u}, \mathbf{v})$ discovered during a lattice sieve will be such that $\theta(\mathbf{u}, \mathbf{v}) \in (\pi/3 - \varepsilon, \pi/3)$ for some small $\varepsilon > 0$.

In Theorem 2.5.1 we give asymptotic bounds on the list size required to find reducing pairs. Let $C \subset L$ be a set of uniformly distributed vectors in ann_γ as in Definition 2.5.3,

then it is shown [NV08, Lem. 4.1] that for many further uniformly distributed points $\mathbf{v} \in \mathsf{ann}_\gamma$ to be able to form reducible pairs $(\mathbf{u}, \mathbf{v})$ for some $\mathbf{u} \in C$ with overwhelming probability, C need only take the lower bound given in Theorem 2.5.1.

Lemma 2.5.3 (Average case size of C, [NV08, Lem. 4.1])**.** *Let $n \in \mathbb{Z}^+$, $\gamma \in (2/3, 1)$ and $c_\gamma = 1/(\gamma \cdot \sqrt{1 - \gamma^2/4})$. Also let $N_\gamma \in c_\gamma^{n+o(n)}$ and $N \in (N_\gamma, 2^n)$. Finally let $L \subset \mathsf{ann}_\gamma$ be a list of size N of uniformly distributed vectors. Then for any $C \subset L$ of size at least N_γ, for all $\mathbf{v} \in L$ there exists a $\mathbf{u} \in C$ such that $\|\mathbf{u} \pm \mathbf{v}\| \leq \gamma$ with overwhelming probability in n.*

Note that $c_\gamma \to \sqrt{4/3}$ as $\gamma \to 1$, so N_γ becomes the lower bound of Theorem 2.5.1.

When $L \subset \mathsf{ann}_\gamma$ the condition $\|\mathbf{u} \pm \mathbf{v}\| \leq \gamma$ is equivalent to the maximum allowable length of vectors in L reducing by a factor of γ. As $\gamma \to 1$ the condition $\|\mathbf{u} \pm \mathbf{v}\| \leq \gamma$ becomes almost identical to the reduction criterion of Definition 2.5.1; $\|\mathbf{u} \pm \mathbf{v}\| < \max\{\|\mathbf{u}\|, \|\mathbf{v}\|\}$.

This heuristic tells us that for a list of the correct size, almost all elements will be part of a non trivial reducible pair. It is given in terms of the set C of *centres* because [NV08] form one element, say $\mathbf{u}$, of each reducible pair from such a set, and then remove C from the list L after each iteration of their sieve. The sieves we consider in this work do not delete vectors in this way, and instead perform nested loops over the list L to find reducible pairs. Such an approach certainly finds the reducible pairs $(\mathbf{u}, \mathbf{v})$ such that $\mathbf{u}$ is from a subset $C \subset L$. We therefore arrive at the following heuristic.

Definition 2.5.5 (Average case size of L)**.** Let $\Lambda \subset \mathbb{R}^n$, $L \subset \Lambda$ be uniformly distributed in ann_γ, and $\gamma \to 1$. To ensure a $1 - o(1)$ fraction of elements $\mathbf{u} \in L$ are part of a reducible pair $(\mathbf{u}, \mathbf{v})$ for $\mathbf{v} \in L \setminus \{\mathbf{u}\}$ with overwhelming probability, we may take $|L| \in (4/3 + \varepsilon)^{n/2+o(n)}$ for any $\varepsilon > 0$.

The final heuristic concerns the properties of the entire list L after a sieve has terminated. The beginning list size of [NV08, Alg. 4] is $\mathrm{poly}(n) \cdot c_\gamma^{n+o(n)}$ because in each iteration approximately $c_\gamma^{n+o(n)}$ elements of L are deleted and the maximum allowable length of the remaining elements of L is decreased by a factor of γ. The sieve is iterated until the list is empty, and the shortest vector in the previous list is returned.

As the vectors remaining in the list become particularly short, the list size can decrease either (as before) by centres being removed, or through collisions. A collision is when a pair reduces to $\mathbf{0}$ and may occur when a repeated element of L becomes a centre. By [NV08, Lem. 4.4] and the experimental evidence of [NV08, Fig. 1], we expect collisions to become numerous only after we have many good approximations

to the shortest vector in our list, specifically lattice vectors of length approximately $4/3 \cdot \lambda_1(\Lambda)$. This is effectively a birthday paradox argument as $\gamma \to 1$ – we expect about $(4/3)^n$ lattice vectors below this length, mostly having length approximately $4/3 \cdot \lambda_1(\Lambda)$, from which we have a uniform sample of size $(4/3)^{n/2}$.

By the Gaussian heuristic we expect a ball of radius roughly $\sqrt{4/3} \cdot \mathrm{gh}(\Lambda)$ to contain $(4/3)^{n/2}$ vectors. For sieves that maintain a database of size $(4/3)^{n/2}$ throughout their operation, e.g. those that follow the blueprint of [MV10c] and do not discard centres, and instead terminate when they see many collisions or make little progress, we have the following heuristic.

Definition 2.5.6 (Composition of terminal L)**.** Let $r = \sqrt{4/3} \cdot \mathrm{gh}(\Lambda)$ for some $\Lambda \subset \mathbb{R}^n$, and $L \subset \Lambda$. After a lattice sieve has terminated, the final list L is expected to be of size approximately $(4/3)^{n/2}$ and such that

$$\frac{|L \cap B_d(r)|}{(4/3)^{n/2}} = c \in (0, 1).$$

This heuristic may serve a dual purpose as a termination condition, one can *define* this notion of saturation for the final list as the termination condition, as suggested in [Duc18a]. That is, one can continue sampling and sieving until the list contains a specified fraction c of the number of lattice vectors expected by the Gaussian heuristic to be below a certain length. Having a large number of short lattice vectors in the terminal list of a sieve is crucial for the dimensions for free techniques of Algorithm 5.

2.5.2 Geometric figures on the sphere

Our analyses throughout this thesis will require several facts about the size of some geometric figures on the sphere. We measure the volume of subsets of S^{n-1} using the standard $(n-1)$ dimensional spherical probability measure μ^{n-1}. This can be defined by considering the n dimensional Lebesgue measure L^n, some $X \subset S^{n-1}$, and defining $\mathsf{cone}(X) = \{\mathbf{y} \colon \mathbf{y} = t \cdot \mathbf{x},\, t \in [0,1],\, \mathbf{x} \in X\} \subset \mathbb{R}^n$. We say X is measurable under μ^{n-1} if and only if $\mathsf{cone}(X)$ is measurable under L^n, and in such a case set

$$\mu^{n-1}(X) = v_n^{-1} \cdot L^n(\mathsf{cone}(X)).$$

We have $\mu^{n-1}(S^{n-1}) = 1$, and for measurable $X \subset S^{n-1}$ intuitively $\mu^{n-1}(X)$ is the fraction of the surface of S^{n-1} that X covers. The spherical cap of angle θ about

$\mathbf{u} \in S^{n-1}$ is $C^{n-1}(\mathbf{u}, \theta) = \{\mathbf{v} \in S^{n-1} : \theta(\mathbf{u}, \mathbf{v}) \leq \theta\}$. The measure of a spherical cap is

$$C_n(\mathbf{u}, \theta) = \mu^{n-1}(C^{n-1}(\mathbf{u}, \theta)) = \frac{1}{\sqrt{\pi}} \frac{\Gamma(\frac{n}{2})}{\Gamma(\frac{n-1}{2})} \int_0^\theta \sin^{n-2}(t)\ \mathrm{d}t.$$

We will often interpret $C_n(\mathbf{u}, \theta)$ as the probability that $\mathbf{v}$ drawn uniformly from S^{n-1} satisfies $\theta(\mathbf{u}, \mathbf{v}) \leq \theta$. We denote the density of the event $\theta(\mathbf{u}, \mathbf{v}) = \theta$ by

$$A_n(\mathbf{u}, \theta) = \frac{\partial}{\partial \theta} C_n(\mathbf{u}, \theta) = \frac{1}{\sqrt{\pi}} \frac{\Gamma(\frac{n}{2})}{\Gamma(\frac{n-1}{2})} \sin^{n-2}(\theta). \tag{2.18}$$

Note that $C_n(\mathbf{u}, \theta)$ does not depend on $\mathbf{u}$, so we may write $C_n(\theta)$ and $A_n(\theta)$ without ambiguity. The wedge formed by the intersection of two caps is $W^{n-1}(\mathbf{u}, \theta_\mathbf{u}, \mathbf{v}, \theta_\mathbf{v}) = C^{n-1}(\mathbf{u}, \theta_\mathbf{u}) \cap C^{n-1}(\mathbf{v}, \theta_\mathbf{v})$. The measure of a wedge only depends on $\theta = \theta(\mathbf{u}, \mathbf{v})$, $\theta_\mathbf{u}$, and $\theta_\mathbf{v}$, so we denote it

$$W_n(\theta, \theta_\mathbf{u}, \theta_\mathbf{v}) = \mu^{n-1}(W^{n-1}(\mathbf{u}, \theta_\mathbf{u}, \mathbf{v}, \theta_\mathbf{v})).$$

We will often interpret $W_n(\theta, \theta_\mathbf{u}, \theta_\mathbf{v})$ as the probability that $\mathbf{w}$ drawn uniformly from S^{n-1} satisfies $\theta(\mathbf{u}, \mathbf{w}) \leq \theta_\mathbf{u}$ and $\theta(\mathbf{v}, \mathbf{w}) \leq \theta_\mathbf{v}$. Note $\theta \geq \theta_\mathbf{u} + \theta_\mathbf{v} \Rightarrow W_n(\theta, \theta_\mathbf{u}, \theta_\mathbf{v}) = 0$.

We see that the density (2.18) becomes a point function at $\pi/2$ as $n \to \infty$. Since the density is symmetric around $\pi/2$ on the interval $[0, \pi]$ we need only understand the interval $[0, \pi/2]$. By [BDGL16, Lem. 2.1] we have $C_n(\theta) \in 1/2 \cdot (\sin\theta)^{n+o(n)}$ for $\theta \in [0, \pi/2]$. We note that the spherical caps $\mathcal{C}_n(\alpha)$ of [BDGL16] with $\alpha \in [0, 1]$ are equivalent to $C_n(\theta)$ above for $\cos\theta = \alpha$ and $\theta \in [0, \pi/2]$. The factor of a half is present because we know $C_n(\pi/2) = 1/2$. The asymptotic relation $C_n(\theta) \in 1/2 \cdot (\sin\theta)^{n+o(n)}$ gives us the $\pi/3$ heuristic of Definition 2.5.4. Indeed, given a $\mathbf{u} \in S^{n-1}$ and any $\theta < \pi/2$ imagine we sample many i.i.d. $\mathbf{v}_i \leftarrow \mathsf{U}(S^{n-1})$. For any $\varepsilon > 0$, any $\mathbf{v}_i \in C_n(\mathbf{u}, \theta)$ will be such that $\theta(\mathbf{u}, \mathbf{v}_i) \in (\theta - \varepsilon, \theta]$ with overwhelming probability in n. An integral representation of $W_n(\theta, \theta_\mathbf{u}, \theta_\mathbf{v})$ is given in [AGPS19, App. A].

2.5.3 Various sieves

In this section a brief introduction to the sieves considered in this thesis is given, though more details can be found in the relevant papers. All the sieves introduced below make some use of the heuristics introduced in Section 2.5.1. While only 2-sieves are introduced above, in Chapter 3 we will also make use of a 3-sieve, and in Chapter 4 consider k-sieves for any constant k. For a more thorough introduction see [Laa16], which provides a comprehensive analysis of many of the sieves described below.

Nguyen–Vidick

The sieve of [NV08, Sec. 4] is the first heuristic lattice sieve. It introduced many of the heuristics above and provided the first experimental evidence for their accuracy. In its original formulation a sieving iteration strictly ensured the maximum allowable length of vectors in the list decreased by a geometric factor $\gamma < 1$. It achieved this via Lines 3–13 of [NV08, Alg. 5]; a set of centres is initialised as empty, any lattice vector that is already short enough immediately passes to the next list, any that is not is checked against the current contents of the set of centres to see if a satisfying reducible pair can be found, and if not is added to the set of centres. This set of centres is discarded at the end of the sieving iteration, and a new sieving iteration begins. The sieve concludes when the list output from a sieving iteration is empty, and the shortest vector from the previous list is returned. By the analysis of [NV08, Lem. 4.1], under the heuristic of Definition 2.5.3 they expect a space complexity of $(4/3+\varepsilon)^{n/2+o(n)}$ for any $\varepsilon > 0$. Since each of the polynomial number of iterations performs a number of operations quadratic in the list size, the time complexity is $(4/3+\varepsilon)^{n+o(n)}$.

The phrase 'a Nguyen–Vidick (or NV) style sieve' has come to mean something more general than the original description given above, and instead means a heuristic sieve that first samples a list of lattice vectors, of a size assumed to be sufficient, and then performs some form of search for reducible pairs (or k tuples in the case of k-sieves) without concerning itself with the distribution of the returned lattice vectors. That is, it is the initial sampling, the length reduction factor γ, and the uniform distribution heuristic of Definition 2.5.3, and not the use of centres, that describe an 'NV style sieve'. We note that when considering $\mathbf{u}, \mathbf{v} \in S^{n-1}$ requiring that $\theta(\mathbf{u}, \mathbf{v}) \leq \theta < \pi/3$ is equivalent to requiring $\|\mathbf{u} \pm \mathbf{v}\| \leq \gamma < 1$ for some $\gamma = \gamma(\theta)$.

We make precise the versions of an NV sieve used in this thesis in Section 3.6.1 and as Algorithm 10.

Gauss

The Gauss sieve [MV10c] introduced several practical improvements to heuristic sieves, in particular the idea that the required list size need not be determined a priori, that the list itself need not be sampled in advance, and that no lattice vectors need be deleted from the list. To achieve this, two data structures are used, a list and a stack, which are both initially empty. Lattice vectors are sampled as required and pairs are considered in a particular manner, specifically GaussReduce of [MV10c, Alg. 2], such that any pair of vectors in the list is always 'Gauss reduced', hence the

name. A pair being Gauss reduced is another expression for a pair being reduced, as in Definition 2.5.1. Since $\theta(\mathbf{u}, \mathbf{v}) \geq \pi/3$ is a necessary condition for a pair $(\mathbf{u}, \mathbf{v})$ to be reduced, this upper bounds the size of the list in the Gauss sieve on some full rank lattice $\Lambda \subset \mathbb{R}^n$ by τ_n. We have therefore gone from an average case argument on the list size under a heuristic [NV08, Lem. 4.1] to knowing that the list size is upper bounded by τ_n, a quantity for which the best known upper and lower bounds differ by an exponential factor, recall Theorem 2.5.1. However, in both cases we arrive at the same expected list size, since lattices that would give a kissing number above the lower bound of Theorem 2.5.1 seem hard to find.

Now that there is no natural termination condition, as opposed to [NV08] where the output list is eventually empty, the question of when to terminate the Gauss sieve needs to be addressed. The original work [MV10c] counts collisions, where reducing pairs return $\mathbf{0}$, and termiates after a certain number are observed. For example, terminating after 500 collisions is suggested in prose, and the implementation [Vou11] terminates after a linear function of the maximum list size of collisions are observed. In Chapter 3 we will replace this termination condition with one that ensures the saturation heuristic of Definition 2.5.6 is achieved by simply continuing to sample and reduce lattice vectors until satisfaction.

A 'Gauss style sieve' is one that does not sample all lattice vectors in advance, but rather one that samples them as required and attempts to maintain a list where each pair is reduced. Since there are no known non heuristic termination conditions for the Gauss sieve, a strict time complexity cannot be given. However in practice it appears more efficient than NV style sieves, while maintaining the same expected space complexity.

We make precise the version of a Gauss sieve used in this thesis in Section 3.6.1.

Locality Sensitive Sieves

Most pairs of vectors considered in the above sieving procedures will be almost orthogonal to each other, and therefore already reduced. In particular, for a list of size N we check whether approximately N^2 pairs are reduced and hope to find N that are not so the procedure can be repeated. These checks use inner products to calculate lengths, and since sampling N vectors costs $N^{1+o(1)}$, these inner products represent the dominant term when calculating time complexity.

The idea of locality sensitive sieves is to perform some preprocessing on the list. This preprocessing allows us to consider pairs of vectors that have a higher probability to be reducible than pairs sampled uniformly from the list. Locality sensitive procedures

come in two flavours. In locality sensitive *hashing* the list is preprocessed and its elements are stored in hash tables. The hashing procedure is designed so that vectors that share a hash value in one or more of the tables are likely to form reducible pairs. In locality sensitive *filtering*, rather than all vectors being given hash values, they either survive a series of filters, or they do not. These filters are designed such that vectors that survive the same filters are likely to form reducible pairs. Provided the process of creating the hash tables or of filtering is not too costly, and that vectors that share a hash value or survive the same filters are sufficiently likely to form reducible pairs, then by foregoing testing all possible pairs the sieve procedure can become more efficient.

For lattice sieving this line of work was initiated by [Laa15] and followed by a series of works [Laa17, LdW15, BL16, BGJ15, BDGL16] which provide many time memory tradeoffs. These works take the angle between a pair of vectors as their notion of locality due to its close relationship to a pair being reducible, recall Section 2.5.1.

The two works we study in this thesis are the Becker–Gama–Joux sieve [BGJ15], which decreases the time complexity without increasing the space complexity, and the Becker–Ducas–Gama–Laarhoven sieve [BDGL16], which is the asymptotically fastest known heuristic sieve. Both of the above sieves use of locality sensitive filtering techniques. In particular these filters are defined using spherical caps, recall Section 2.5.2.

The original Becker–Gama–Joux sieve on a lattice $\Lambda \subset \mathbb{R}^n$ recursively applies K filters from a set $\mathcal{F}(m, \delta)$,[8] with $m \in \mathbb{Z}^+$ and $\delta \in (0, 1)$, to a list $L \subset \Lambda$. A filter f is defined by $\mathbf{x}_1, \ldots, \mathbf{x}_m \leftarrow \mathsf{U}(S^{n-1})$ and

$$f(\mathbf{u}) = \begin{cases} 0 & \text{if } |\{\mathbf{x}_i \colon \langle \mathbf{x}_i, \mathbf{u} \rangle \geq 0\}| \leq (1-\delta) \cdot m/2, \\ 1 & \text{otherwise.} \end{cases}$$

We say $\mathbf{u}$ passes the filter f if $f(\mathbf{u}) = 0$. The above filtering procedure is iterated K times using filters $f_1, \ldots, f_K$ defined as f is above, where each acts on the output filtered list of the prior filter. We examine a simplified variant of this sieve in Chapter 3 and Chapter 5. In this simplified variant we set $K = m = 1$ and replace the parameter δ with θ defining a spherical cap around $\mathbf{x} = \mathbf{x}_1$. Namely, a vector $\mathbf{u}$ in the list L passes the filter if $\mathbf{u} \in C^{n-1}(\mathbf{x}, \theta)$, and we search within $L \cap C^{n-1}(\mathbf{x}, \theta)$ for reducible pairs. This simplified filter is then repeated t times. Repeated here does not mean iterated, i.e. we do not filter again within the same spherical cap, but instead sample another $\mathbf{x} \leftarrow \mathsf{U}(S^{n-1})$ to act as a spherical cap centre, t times. The parameters are

[8] In their notation m is dimension and n the filter parameter, but we transpose them here.

therefore θ and the number of spherical caps centres t. See Algorithm 12 of Chapter 5 for more details.

The Becker–Ducas–Gama–Laarhoven sieve uses spherical caps around a list vector to determine which of many filters to consider. Given a list $L \subset \Lambda$ it defines t uniformly sampled spherical cap centres $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_t\} \subset S^{n-1}$ which will be used to describe filters. It determines $L_i = L \cap C^{n-1}(\mathbf{x}_i, \theta_2)$ for some θ_2 and all $\mathbf{x}_i \in X$. Here L_i represents the list vectors that pass the filter defined by $\mathbf{x}_i$. Then, given a vector $\mathbf{u} \in L$ with which one wants to form reducible pairs, it determines which spherical cap centres are close to $\mathbf{u}$. This is achieved by forming the set $X_\mathbf{u} = X \cap C^{n-1}(\mathbf{u}, \theta_1)$ for some θ_1. For all $\mathbf{x}_i \in X_\mathbf{u}$, the spherical cap centres near to $\mathbf{u}$, it checks for reducible pairs $(\mathbf{u}, \mathbf{v})$ with $\mathbf{v} \in L_i$. The parameters are therefore θ_1, θ_2 and the number of spherical cap centres t. See Algorithm 13 of Chapter 5 for more details.

k-Sieves

So far we have considered sieves that, perhaps after some preprocessing, check whether a pair of list vectors is reducible. These sieves have space complexity of $(4/3)^{n/2+o(n)}$ which is exponential in the dimension of the lattice in which we wish to find short vectors. For high dimensions this space complexity may become prohibitive. The idea of a k-sieve [BLS16] is to trade increased time complexity for decreased space complexity by considering k tuples of list vectors at once. A k tuple that satisfies the length constraints discussed below is said to be reducible, similar to the definition for a pair being reducible. In the $k = 3$ case we may therefore consider $\mathbf{u}, \mathbf{v}, \mathbf{w} \in L$ and check whether $\|\mathbf{u} \pm \mathbf{v} \pm \mathbf{w}\| < \max\{\|\mathbf{u}\|, \|\mathbf{v}\|, \|\mathbf{w}\|\}$, where $\|\mathbf{u} \pm \mathbf{v} \pm \mathbf{w}\|$ is a shorthand for all four possibilities. To think of this in terms of Figure 2.1 fix $\mathbf{u}$ and form four planes, one from each of $\pm\mathbf{v} \pm \mathbf{w}$. From every triple, one then has four possibilities to fall into the shaded region. On the other hand, to consider each triple naïvely one has to perform three nested loops over L rather than two in the 2-sieve case, increasing the time complexity by an exponential factor. We note that we can also still search for reducible *pairs* from L, but that since the aim is to reduce the memory complexity as far as possible, and in particular exponentially below $(4/3)^{n/2+o(n)}$, we do not expect to find many and discount them from the rest of this section. In Chapter 3 the 3-sieve we implement is also able to perform this search for reducible pairs with minimal overhead.

A k-sieve can be performed in both the style of an NV sieve and of a Gauss sieve, though the latter case contains a little subtlety [BLS16]. In a Gauss style 2-sieve we ensure that there exist no reducible pairs in the list by checking $\|\mathbf{u} \pm \mathbf{v}\| \geq \max\{\|\mathbf{u}\|, \|\mathbf{v}\|\}$ for all distinct $\mathbf{u}, \mathbf{v} \in L$. Let $\|\mathbf{u} \pm \mathbf{v}\| \geq \max\{\|\mathbf{u}\|, \|\mathbf{v}\|\}$, $\|\mathbf{u}\| \leq \|\mathbf{v}\|$

and Λ be generated by $\mathbf{u}$ and $\mathbf{v}$. Then we have $\|\mathbf{u}\| = \lambda_1(\Lambda)$ and $\|\mathbf{v}\| = \lambda_2(\Lambda)$, see e.g. [Gal12, Sec. 17.1]. We say that $(\mathbf{u}\ \mathbf{v})$ is a Gauss–Lagrange reduced basis for Λ, which is the two dimensional specialisation of a Minkowski reduced basis, recall Definition 2.4.3. For $n > 4$ we cannot in general achieve $\Lambda = \Lambda(\mathbf{B})$ with $\mathbf{B} = (\mathbf{b}_1 \cdots \mathbf{b}_n)$ and $\|\mathbf{b}_i\| = \lambda_i(\Lambda)$. Instead the correct notion for a Gauss style k-sieve, which attempts to keep a list L such that no combination of k vectors in L can form something shorter, is for each k tuple of vectors in L to be Minkowski reduced. More precisely, we require that for any pairwise distinct $\mathbf{u}_1, \ldots, \mathbf{u}_k \in L$ there exists some permutation $\sigma \in S_k$ such that $\mathbf{B} = (\mathbf{u}_{\sigma(1)} \cdots \mathbf{u}_{\sigma(k)})$ is a Minkowski reduced basis for $\Lambda = \Lambda(\mathbf{B})$. The results of [Tam76] tell us that for $k \leq 4$ this is possible by only considering scalars of absolute value one, i.e. that if $\mathbf{u}_1, \ldots, \mathbf{u}_4$ are such that $\|\mathbf{u}_1\| \leq \cdots \leq \|\mathbf{u}_4\|$ and

$$\begin{aligned}
\|\mathbf{u}_1 \pm \cdots \pm \mathbf{u}_4\| &\geq \max\{\|\mathbf{u}_1\|, \ldots, \|\mathbf{u}_4\|\}, \\
\|\mathbf{u}_i \pm \mathbf{u}_j \pm \mathbf{u}_k\| &\geq \max\{\|\mathbf{u}_1\|, \ldots, \|\mathbf{u}_4\|\} \text{ for } i, j, k \text{ pairwise distinct}, \\
\|\mathbf{u}_i \pm \mathbf{u}_j\| &\geq \max\{\|\mathbf{u}_1\|, \ldots, \|\mathbf{u}_4\|\} \text{ for } i \neq j,
\end{aligned}$$

then $(\mathbf{u}_1 \cdots \mathbf{u}_4)$ is a Minkowski reduced basis. One can ensure this condition, and hence a Minkowski reduced basis, for $k \leq 4$ using the algorithms of [Sem01, NS09].

In practice 3-sieves appear most useful, and works such as [HK17, HKL18] and Chapter 4 which consider arbitrary k focus on asymptotic performance. Here the distinction between NV and Gauss style sieves is unimportant, and as such we do not attempt the Minkowski reduced criterion given above. Instead we consider only $\mathbf{u}_1 \pm \cdots \pm \mathbf{u}_k$ regardless of k. In [HK17] a k-sieve is shown to be a particular case of a more general problem called the k configuration problem. The configuration problem framework both allows a smooth time memory trade off for k-sieves and also gives a lower time complexity when considering the same space complexity as [BLS16]. The work of [HKL18] explores these tradeoffs more completely and also applies the locality sensitive filters of [BDGL16] described above. In Chapter 4 we introduce the configuration framework and examine the ideas of [HK17, HKL18] using quantum search routines.

2.5.4 The SimHash prefilter

Charikar's locality sensitive hashing (LSH) scheme [Cha02] is a family of hash functions $\mathcal{H}$ defined on $\mathbb{R}^d$ for which, given $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$

$$\Pr_{h \leftarrow \mathsf{U}(\mathcal{H})}[h(\mathbf{u}) = h(\mathbf{v})] = 1 - \frac{\theta(\mathbf{u}, \mathbf{v})}{\pi}. \tag{2.19}$$

It is used in the implementation of Chapter 3 and considered extensively in Chapter 5, where quantum circuits are designed for it and a new heuristic analysis of it is given in the lattice sieving setting. It is used as a form of prefiltering, orthogonal to the techniques used by the various sieves introduced in Section 2.5.3. The hash function family is defined by

$$\mathcal{H} = \left\{ h_{\mathbf{r}} \colon \mathbb{R}^d \to \{0,1\},\ \mathbf{u} \mapsto \operatorname{sgn}(\langle \mathbf{r}, \mathbf{u} \rangle) \colon \mathbf{r} \in S^{d-1} \right\},$$

where $\operatorname{sgn}(x) = 1$ if $x \geq 0$ and $\operatorname{sgn}(x) = 0$ if $x < 0$. We have (2.19) since $\theta(\mathbf{u}, \mathbf{v})/\pi$ is the probability that the plane defined by $h \leftarrow \mathsf{U}(\mathcal{H})$ separates $\mathbf{u}$ and $\mathbf{v}$. Note that for all $h \in \mathcal{H}$, all $\mathbf{u} \in \mathbb{R}^d$, and all $\alpha \in \mathbb{R}^+$ we have $h(\mathbf{u}) = h(\alpha \cdot \mathbf{u})$. This invariance allows us to view $\mathcal{H}$ as defined over S^{d-1} rather than $\mathbb{R}^d$, a fact we make use of in Chapter 5.

The original analysis [Cha02, Sec. 3] of this hash family shows how it relates to the similarity of two subsets A, B of some set C. By considering the characteristic vectors x_A, x_B, where e.g. $x_A = (x_c)_{c \in C}$ with $x_c = 1$ if $c \in A$ and $x_c = 0$ otherwise, it is shown that

$$\Pr_{h \leftarrow \mathsf{U}(\mathcal{H})}[h(x_A) = h(x_B)] = 1 - \frac{1}{\pi} \cdot \arccos\left(\frac{|A \cap B|}{\sqrt{|A| \cdot |B|}} \right).$$

Charikar also observed that one can estimate $\theta(\mathbf{u}, \mathbf{v})/\pi$ by choosing a random hash function $h = (h_1, \ldots, h_n) \in \mathcal{H}^n$ and measuring the Hamming distance between $h(\mathbf{u}) = (h_1(\mathbf{u}), \ldots, h_n(\mathbf{u}))$ and $h(\mathbf{v}) = (h_1(\mathbf{v}), \ldots, h_n(\mathbf{v}))$. Each bit $h_i(\mathbf{u}) \oplus h_i(\mathbf{v})$ is Bernoulli distributed with parameter $p = \theta(\mathbf{u}, \mathbf{v})/\pi$. In the limit of large n, the normalised Hamming weight $wt(h(\mathbf{u}) \oplus h(\mathbf{v}))/n$ converges to a normal distribution with mean p and standard deviation $\sqrt{p(1-p)/n} \leq 1/2\sqrt{n}$.

In the sieving literature, the process of approximating $\theta(\cdot, \cdot)$ using a threshold on the value of $wt(h(\mathbf{u}) \oplus h(\mathbf{v}))$ is known as the 'XOR and population count trick' [ADH+19a, Duc18a, FBB+15]. Functions in $\mathcal{H}^n$ are also used in Laarhoven's HashSieve [Laa15]. We write $\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}; h)$ for a search filter of this type, and say that $\mathbf{u}$ and $\mathbf{v}$

pass the filter if the output is 0

$$\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}; h) = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} h_i(\mathbf{u}) \oplus h_i(\mathbf{v}) \leq k, \\ 1 & \text{otherwise.} \end{cases}$$

When `popcount` is used in sieves, a heuristic assumption is made that one can *fix* the function h and *vary* the pairs (i.e. sample $\mathbf{u}, \mathbf{v} \leftarrow \mathsf{U}(S^{d-1})$) and expect the same behaviour as described above. When the n hash functions are fixed we write $\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v})$. The threshold, k, is chosen based on the desired false positive and false negative rates. Heuristically, if one's goal is to detect points at angle at most θ, one should take $k/n \approx \theta/\pi$. If $k/n \ll \theta/\pi$ then the false negative rate will be large, and many neighbouring pairs will be missed. An important consequence of missing potential reductions is that the list size required to iterate various sieves increases. If $k/n \gg \theta/\pi$ then the false positive rate will be large. We ultimately check the angle between two vectors that pass a `popcount` filter using inner products, and so a large false positive rate will lead to (relatively) expensive inner products being calculated often.

2.6 Quantum search routines

In Chapters 4 and 5 we make use of a number of quantum search routines. These use quantum memories and quantum circuits to search a list of elements for an element that satisfies a certain predicate. The two algorithms we will introduce here are Grover's algorithm [Gro96] and amplitude amplification [BHMT02]. Both will be applied in the quantum circuit model, and require quantum access to classical memory.

2.6.1 Qubits, measurement, quantum circuits, and qRAM

A quantum circuit is a system made of wires and quantum gates that carry and manipulate quantum information. For a general introduction to the topics of this section see [NC11, Sec. 1, Sec. 2]. A qubit is a unit vector in $\mathbb{C}^2$ and is the object that encodes (on the logical layer) the quantum information present in one wire of a quantum circuit. It can be represented by

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$$

where each $\alpha_j \in \mathbb{C}$ is an amplitude such that $\sum_j |\alpha_j|^2 = 1$ and $|0\rangle$ and $|1\rangle$ represent an orthonormal basis for $\mathbb{C}^2$. A combination of orthonormal basis vectors is often called a quantum state or superposition. The quantum information of a quantum circuit with n wires is encoded into n qubits, or a unit vector in $(\mathbb{C}^2)^{\otimes n}$. It can be represented as

$$|\psi\rangle = \sum_{j=0}^{2^n-1} \alpha_j \, |j_n\rangle \otimes \cdots \otimes |j_1\rangle$$

where each $\alpha_j \in \mathbb{C}$ is an amplitude such that $\sum_j |\alpha_j|^2 = 1$ and $j_n \cdots j_1$ is j represented in binary. We can equivalently write $|j\rangle$, $|j_n\rangle \cdots |j_1\rangle$, or $|j_n \cdots j_1\rangle$ for $|j_n\rangle \otimes \cdots \otimes |j_1\rangle$. These form an orthonormal basis for $(\mathbb{C}^2)^{\otimes n}$. This particular orthonormal basis is called the computational basis. Letting $N = 2^n$ and $j \in [N-1]_0$ we have $|j\rangle = \mathbf{e}_{j+1}$.

Initialisation of an n qubit state is the process of creating some $|\psi\rangle = \sum_j \alpha_j \, |j\rangle$, e.g. $|0\rangle$ or $(1/\sqrt{2}) \cdot (|0\rangle - |1\rangle)$. Usually it is assumed that initialised states will be a computational basis state, e.g. $|j\rangle$ for some j.

A measurement is an operation applied to a quantum state that collapses (perhaps part of it) to a classical state. The classical state obtained via this process is probabilistic, depending on the amplitudes of the quantum state. We will only concern ourselves with measurement in the computational basis. Given some quantum state $|\psi\rangle = \sum_j \alpha_j \, |j\rangle$ we define the measurement operators $\{M_j\}_{j=0}^{2^n-1}$ as $M_j = |j\rangle \langle j|$. Note that M_j is the all zero matrix with a one in the $(j+1) \times (j+1)^{\text{th}}$ position. The measurement of $|\psi\rangle$ with respect to M_j is

$$\langle\psi| \, M_j^\dagger M_j \, |\psi\rangle = \langle\psi| \, M_j^2 \, |\psi\rangle = \langle\psi| \, M_j \, |\psi\rangle = |\alpha_j|^2 ,$$

where one has $M_j^\dagger M_j = M_j^2 = M_j$. These measurement operators tell us that the probability of the quantum state returning j when measured is $|\alpha_j|^2$, which also explains why we required the sum of these values to equal one. By considering $j = j_n \cdots j_1$, the binary decomposition of j, we may measure only part of a quantum state. For example if $n = 2$ and we wish to measure the first qubit only then we apply the measurement $M_{00} + M_{01}$ to determine the probability that it will be zero

$$\langle\psi| \, (M_{00} + M_{01})^\dagger (M_{00} + M_{01}) \, |\psi\rangle = \langle\psi| \, (M_{00} + M_{01}) \, |\psi\rangle = |\alpha_{00}|^2 + |\alpha_{01}|^2 ,$$

and similarly $M_{10} + M_{11}$ to determine the probability that it will be one. After this partial measurement the remaining quantum state will need to be renormalised by a factor of $\left(|\alpha_{00}|^2 + |\alpha_{01}|^2\right)^{-1/2}$.

Ignoring the initialisation and measurement of qubits, a quantum circuit is a sequence of unitary operations on the n qubits, one per unit time. A unitary operation is any transformation $\mathbf{U} \in \mathbb{C}^{N\times N}$ such that $\mathbf{U}^\dagger\mathbf{U} = \mathbf{U}\mathbf{U}^\dagger = \mathbf{I}_N$. In particular, any transformation $\mathbf{U}$ of the information of a quantum circuit is reversible via the application of the unitary $\mathbf{U}^\dagger$. The outcome of applying a unitary $\mathbf{U}$ to a quantum state $|\psi\rangle$ is given by the multiplication $\mathbf{U}|\psi\rangle$.

To realise the unitary operations above we use quantum gates. These are themselves unitary operations that may act on fewer than n wires of a quantum circuit. Only one gate may act on a given wire in a given time step, and unitaries on the full circuit are formed by parallel and sequential compositions of quantum gates. A set of quantum gates is called *universal* if every unitary on an arbitrary number of wires can be formed (or more properly, approximated arbitrarily closely) by finite parallel and sequential compositions of gates from the gate set. When combined with the appropriate qubit initialisation and measurement such a gate set allows one to describe any quantum computation. In Chapter 4 we will make use of an unspecified universal gate set, whereas in Chapter 5 we will specifically use the Clifford+T gate set. The Clifford+T gate set is formed of the following one and two qubit unitaries

$$\mathbf{H} = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}, \quad \mathbf{CNOT} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}, \quad \mathbf{T} = \begin{pmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{pmatrix}.$$

This is a universal gate set that is closely related to quantum error correcting codes, a topic of interest for that chapter. In particular, in Chapter 4 we assume a noiseless model of quantum computation, that is we assume quantum computation does not suffer from errors. In Chapter 5 we specifically attempt to cost the extra computation required to overcome errors in quantum computation. The gate $\mathbf{H}$ may also be called a *Hadamard* gate.

Alongside the Clifford+T gate set, we will make use of a unit cost quantum table look up operation. We call this operation qRAM [GLM08] for quantum (access to classical) random access memory. Given some classical registers $(R_0, \ldots, R_{2^n-1})$ each encoding an ℓ bit binary string, a classical RAM is a circuit that takes as input an index j and returns the ℓ bits stored in R_j. The qRAM operation is described by a

gate that can return arbitrary superpositions over the registers as follows

$$\sum_{j=0}^{2^n-1} \alpha_j |j\rangle |x\rangle \xrightarrow{\text{qRAM}} \sum_{j=0}^{2^n-1} \alpha_j |j\rangle |x \oplus R_j\rangle.$$

Here x is an arbitary ℓ bit string. Therefore qRAM is an $n + \ell$ qubit gate that uses n qubits to index the registers, and ℓ for the contents of a register. Note that even though the classical memory may be large, e.g. our 2^n registers, the width of the quantum circuit enacting the qRAM gate is polynomial in n (provided also that ℓ is).

A quantum circuit is depicted as a series of horizontal lines to be read from left to right, with each line representing a wire carrying a qubit. A quantum circuit, when thought of as a graph, is directed (from left to right) and acyclic, i.e. no feedback loops are allowed, unlike a classical circuit. Similarly, there can be no fan in or fan out operations, as the first of these is not unitary and the latter violates the no cloning theorem [Die82, WZ82]. In a time step either no gate is applied to a wire (depicted by the identity wire), or a gate is applied to a wire (depicted by some box with a label, or reserved symbol). A gate that acts on several qubits will span several wires. The width of a circuit is the maximum number of wires over all time steps. The depth of circuit is the longest path from an input to an output.

Any classical circuit can be efficiently simulated by a quantum circuit using a Toffoli gate and ancilla qubits, see [NC11, Sec. 3.2.5]. Thus we have a quantum circuit with the same number of gates and a similar width. A Toffoli gate can be formed from some small constant number of Clifford+T gates [AMMR13].

2.6.2 Grover's algorithm and amplitude amplification

Here we introduce a quantum mechanical search algorithm for unstructured search called Grover's algorithm, and a generalisation of it called amplitude amplification. However first we define the style of search problem we will consider, and give a brief analysis of classical exhaustive search.

Predicates and exhaustive search

A predicate on $\{0, 1, \ldots, N-1\}$ is a function $f : \{0, 1, \ldots, N-1\} \to \{0, 1\}$. The kernel, or set of roots, of f is $\mathsf{Ker}\,(f) = \{x : f(x) = 0\}$. We write $|f|$ for $|\mathsf{Ker}\,(f)|$. A black box search algorithm finds a root of a predicate without exploiting any structure present in the description of the predicate itself.

Exhaustive search is an example of a classical black box search algorithm. Here one simply queries f in any order (providing no queries are repeated), e.g. $f(0), f(1), \ldots$, until a root is returned or the entire domain has been queried. Let f be a function sampled uniformly from all functions mapping $[N-1]_0$ to $\{0,1\}$ that have M roots. Exhaustive search on such an f will succeed with probability $1 - \binom{N-j}{M} / \binom{N}{M} \geq 1 - (1 - M/N)^j$ by the j^{th} query. This is independent of knowledge of M. Using the negative hypergeometric distribution, the expected number of queries is $(N+1)/(M+1)$.

Grover's algorithm

Grover's algorithm [Gro96] is a quantum black box search algorithm. It provides a quadratic improvement in terms of query complexity compared to exhaustive search. Here a query means the application of a unitary that encodes f, rather than classical query to f on a given value. Let $2^{n-1} < N \leq 2^n$ and consider a predicate $f\colon [N-1]_0 \to \{0,1\}$. We define the following unitaries

- a unitary that puts the domain of f into equal superposition $\mathbf{D}\left|0\right\rangle = \frac{1}{\sqrt{N}} \sum_j \left|j\right\rangle$,
- a unitary that negates the phase of a non root $\mathbf{R}_f \left|j\right\rangle = (-1)^{f(j)} \left|j\right\rangle$,
- a unitary $\mathbf{R}_0 = \mathbf{I}_N - 2\left|0\right\rangle\left\langle 0\right|$, the identity matrix except its top left entry is -1,[9]
- a unitary $\mathbf{G}(f) = \mathbf{D}\mathbf{R}_0\mathbf{D}^\dagger\mathbf{R}_f$ called a Grover iteration, or a query.

Note that to define these unitaries we need only define them on an orthonormal basis. Therefore in the case of $\mathbf{R}_f$ we have e.g. $\mathbf{R}_f\mathbf{D}\left|0\right\rangle = \frac{1}{\sqrt{N}} \sum_j (-1)^{f(j)} \left|j\right\rangle$.

Grover's algorithm measures the superposition $\mathbf{G}(f)^k\mathbf{D}\left|0\right\rangle$ for some well chosen k. The analysis of [BBHT98] shows that for a predicate f with M roots, if we define $\sin^2\theta = M/N$ and take $k = \lfloor \pi/4\theta \rfloor$ then measuring $\mathbf{G}(f)^k\mathbf{D}\left|0\right\rangle$ returns a root of f with probability greater than $1 - M/N$. This k is less than $(\pi/4)\cdot\sqrt{N/M}$, and so Grover's algorithm finds a root with probability at least $1 - M/N$ using at most $\sqrt{N/M}$ queries to $\mathbf{G}(f)$. We summarise in the following theorem.

Theorem 2.6.1. *If f is a predicate on a domain of size N with M roots, $\mathbf{D}$ and $\mathbf{G}(f)$ are defined as above, and $\sin^2\theta = M/N$ then measuring the state $\mathbf{G}(f)^k\mathbf{D}\left|0\right\rangle$ with $k = \lfloor \pi/4\theta \rfloor \leq (\pi/4)\cdot\sqrt{N/M}$ returns a root of f with probability at least $1 - M/N$.*

[9] The unitaries we have defined here are the same as defining $\mathbf{R}_f$ to negate the phase of a *root* and taking $\mathbf{R}_0 = 2\left|0\right\rangle\left\langle 0\right| - \mathbf{I}_N$.

Note the following important fact about Grover's algorithm. The state after applying j Grover iterations to a uniform superposition over N elements is

$$\mathbf{G}(f)^j\mathbf{D}\,|0\rangle = \sum_{\ell\in\mathsf{Ker}\,(f)} \alpha(j,N)\,|\ell\rangle + \sum_{\ell\notin\mathsf{Ker}\,(f)} \beta(j,N)\,|\ell\rangle.$$

That is, the amplitudes of the roots are equal, and similarly the amplitudes of the non roots are equal. This means, given that the measured state is e.g. a root, it will be a uniformly selected root, and similarly for non roots. The exact values for $\alpha(j,N)$ and $\beta(j,N)$ are given in [BBHT98, Sec. 3].

Amplitude amplification

In [BHMT02] the authors note that the $\mathbf{D}$ unitary can be replaced by any unitary that makes no measurements, and, if it were measured, would return a root of f with positive probability. Call such a unitary $\mathbf{A}$, i.e. measuring $\mathbf{A}\,|0\rangle$ returns a root of f with probability $a > 0$. Let $\mathbf{G}(\mathbf{A}, f) = \mathbf{A}\mathbf{R}_0\mathbf{A}^\dagger\mathbf{R}_f$ so that $\mathbf{G}(\mathbf{D}, f) = \mathbf{G}(f)$, the Grover iteration. The analysis of [BBHT98, Thm. 2] shows that if θ_a is such that $\sin^2\theta_a = a$ then measuring $\mathbf{G}(\mathbf{A}, f)^k\mathbf{A}\,|0\rangle$ with $k = \lfloor\pi/4\theta_a\rfloor$ returns a root of f with probability at least $\max\{a, 1-a\}$. We summarise in the following theorem.

Theorem 2.6.2. *Let f be a predicate on a domain of size N with M roots, $\mathbf{A}$ and $\mathbf{G}(\mathbf{A}, f)$ be defined as above, and $\sin^2\theta_a = a$ where $a > 0$ is the probability of measuring $\mathbf{A}\,|0\rangle$ and receiving a root of f. If $\mathbf{G}(\mathbf{A}, f)^k\mathbf{D}\,|0\rangle$ for $k = \lfloor\pi/4\theta_a\rfloor \leq \pi/(4\sqrt{a})$ is measured then a root of f is returned with probability at least $\max\{a, 1-a\}$.*

An unknown number of solutions

In Theorem 2.6.1 and Theorem 2.6.2 we assumed knowledge that may not be available in a general search operation. Namely, in Theorem 2.6.1 we assumed we knew the number M of roots the predicate f had, and this allowed us to determine the correct number of Grover iterations to apply. Similarly, in Theorem 2.6.2 we assumed we knew the probability with which we would receive a root of f if we measured some superposition $\mathbf{A}\,|0\rangle$. This probability itself will depend on the number of roots, as well as the particular unitary $\mathbf{A}$. It is possible to retain the same asymptotic complexity for both Grover's algorithm and amplitude amplification when M or a are not known.

The idea is to choose k uniformly from intervals of growing size until success. In the case of Grover's algorithm the strategy is made explicit in [BBHT98, Sec. 4], and

ultimately as [BBHT98, Thm. 3]. For amplitude amplification the strategy is outlined as the algorithm QSearch [BHMT02, Sec. 2], and ultimately as [BHMT02, Thm. 3].

Chapter 3

The General Sieve Kernel, /ʒe.si.ka/

The General Sieve Kernel was born out of a confluence of kindred desires: to consider a lattice sieve as more than an object that spits out a short vector, and instead as a machine with state that shuffles around a basis; to see how far the 'dimensions for free' techniques of [Duc18a] could be pushed, especially within this stateful machine framework; to experiment on various suggestions [LM18, Duc18a] regarding amortising the cost of SVP oracles within tours of lattice reduction; and to *restore a little sanity to the world* (emphasis, implications, and childish humour my own) by showing that there exist practical instances where an asymptotically exponential time lattice sieve runs faster than asymptotically superexponential time lattice point enumeration.

Happily, we were successful in all of these goals, though we did not achieve the levels of amortisation for the SVP oracles that we had initially hoped.

The crux of the idea is that a lattice sieve, as an unavoidable, as far as we know, facet of its design, maintains an exponentially sized database of lattice vectors. As the sieve iterates its search for tuples, specifically pairs and triples in this work, of vectors that can be combined to form new and shorter vectors, the lengths of *all* the vectors in the database become progressively shorter, on average. That is, by the time the sieve terminates, the database does not include a single, useful, short vector and exponentially many long and useless vectors, but instead a great many 'shortish' vectors, the utility of which extends beyond the usefulness of only the shortest.

How should one use, when performing lattice reduction or more generally when attempting to solve SVP type problems, these extra resources?

In the case of SVP type problems, this is exactly one of the contributions of [Duc18a]. For any lattice, and a shortest vector of that lattice, in each projected sublattice there

exists a projection of this shortest vector. The game becomes determining in which 'natural' projected sublattices the projection of this shortest vector is likely to be short in, and in particular, short enough to appear in the database of a completed sieve. Given such a database we will not necessarily know which entry is the projection of this shortest vector, so instead we must apply some pseudoinverse projection operation to this database, or a well chosen subset of candidates. This operation, 'lifting', must maintain the shortness of the candidates, and the extent to which this can be achieved is closely tied to the geometry of the basis being used to lift. We described this procedure as Algorithm 5. The conclusion is that one can achieve a sublinear number of dimensions for free, that is, one can sieve in a projected sublattice a sublinear number of dimensions smaller than the lattice, and lift vectors from the database of this sieve. If the basis vectors being lifted over, i.e. the sublinear number that are being taken for free, are sufficiently well reduced, then one expects to find a short vector in the lattice. This analysis relies on several heuristics introduced in Section 2.4.2.

Another technique for SVP type problems was introduced in [LM18], and is called 'progressive sieving'. The idea is that a sieve which has a database already seeded with somewhat short vectors will terminate more quickly than one starting from scratch. This is not only because the sieve will not have to sample as many vectors, but also because shorter lattice vectors are more useful in a sieve database than longer ones (recall Figure 2.1 and the discussion surrounding it). The question is then how to find somewhat short vectors cheaply, and the answer given is to sieve in *progressively* higher dimensional sublattices. Specifically one takes a sequence of sublattices where each sublattice strictly contains the previous sublattice, e.g. $\Lambda_1 \subsetneq \Lambda_2 \subsetneq \cdots \subsetneq \Lambda_n = \Lambda$, and sieves in order within these sublattices. Crucially, the database of lattice vectors is kept between these sieving operations, and since each subsequent sublattice contains the previous one, we begin the new sieving operation with a large number of shortish vectors. By the time the full dimension of the lattice is reached, the database will have many short vectors from a sublattice as little as one rank smaller, and while many more sieves are performed and the asymptotic complexity is unchanged, in practice this procedure is significantly more efficient in terms of time.

In the lattice reduction case the literature also already contained some suggestions. Indeed, continuing with the work of [LM18], the same idea – that preseeded sieve databases are good – can be applied in the context of BKZ reduction. Consecutive blocks in BKZ are represented by projected sublattices with a lot of overlap, in particular the second in a pair of consecutive blocks introduces one new basis vector and projects against one extra basis vector compared to the first in the pair. If one

can sieve in one block, insert the short vector found, and then move the vectors in the sieve database into this new, but similar, projected sublattice, all while keeping them relatively short, then a tour of BKZ may become more efficient. Still riffing on the idea that subsequent blocks in BKZ reduction overlap, the idea of taking some larger blocksize within BKZ, say by some additive factor k, and then skipping k blocks, is suggested in [Duc18a]. This strategy is justified by the sieving procedures described therein achieving a stronger notion of reduction than is strictly given by BKZ. This stronger notion of reduction also leads to the suggestion that one can perform less work in blocks following the first to achieve the same number of dimensions for free. A final idea, with rough pedigree [Sch03, BL06, FK15, TKH18], is to exhaustively check the global effect on the basis of many different possible insertions within a lattice reduction routine, and to use some scoring function to determine which is best for the particular task at hand. This may lead to a shortest vector not always being chosen for insertion, and the indices for insertions not following a strict pattern. One can imagine a situation where the shortest insert for some index has worse projections, for some definition of worse, than an ever so slightly longer insert at the same index, and therefore wanting to choose this latter insert.

A lattice sieve is, to some greater or lesser extent, naturally well equipped to be used in all of the above ideas. Indeed sieves are used to enact all of the ideas in [Duc18a, LM18], and they produce a large number of potential insertion candidates, some of which will not be a shortest vector in a given lattice, and therefore work well with the ideas of [TKH18]. The design of the General Sieve Kernel was focussed on the easy development and prototyping of such ideas, and similar cousins.

After introducing the stateful machine approach to lattice sieves, the work that follows describes our implementation efforts, along with several lattice reduction and SVP type problem solver procedures, and provides a large body of experimental results regarding their efficacy and performance. Often the line between SVP type problem solver and lattice reduction is blurred; before finding a short vector many, non terminal, insertions may well be made in an opportunistic manner, and without any reference to standard notions of e.g. BKZ reduction. Similarly, one requires lattice reduction to find short vectors, and short vectors to apply lattice reduction. It really is turtles all the way down! To test the introduced procedures against what has become somewhat of a standard benchmark, the work also breaks (what were then unbroken) records in the Darmstadt SVP [GS10] and LWE [GY15] challenges.

Contributions

The author of this thesis contributed equally with the other authors to Sections 3.2 and 3.3, equally with Léo Ducas to Section 3.4, equally with Léo Ducas and Martin Albrecht to Section 3.5, and equally with the other authors to Section 3.7, with a special emphasis on the BKZ and LWE experiments and methods. The author of this thesis contributed equally with Léo Ducas and Martin Albrecht to the Python layer of the implementation.

Acknowledgements

We thank Kenny Paterson for discussing an early draft of this chapter before its initial submission to EUROCRYPT. We also thank Pierre Karpman for running some of our experiments.

3.1 The General Sieve Kernel and new records in lattice reduction

The work presented here is an amended and annotated version of what is published as

Albrecht M.R., Ducas L., Herold G., Kirshanova E., Postlethwaite E.W., Stevens M. (2019) The General Sieve Kernel and New Records in Lattice Reduction.

In: Ishai Y., Rijmen V. (eds) Advances in Cryptology – EUROCRYPT 2019. Lecture Notes in Computer Science, vol 11477. Springer, Cham.
`https://doi.org/10.1007/978-3-030-17656-3_25`

The appendices are available at `https://eprint.iacr.org/2019/089`.

The maintained implementation is available at `https://github.com/fplll/g6k`.

3.2 Introduction

Sieving algorithms have seen remarkable progress over the last few years. Briefly, these algorithms find a shortest vector in a lattice by considering exponentially many lattice vectors and searching for sums and differences that produce shorter vectors. Since the introduction of sieving algorithms in 2001 [AKS01], a long series of works, e.g. [MV10c, BGJ15, HK17], have proposed asymptotically faster variants; the asymptotically fastest of which has a heuristic time complexity of $2^{0.292d+o(d)}$, with d the dimension of the lattice [BDGL16].

Such algorithms for finding short vectors are used in lattice reduction algorithms. These improve the 'quality' of a lattice basis (see Section 2.4) and are used in the cryptanalysis of lattice based cryptography.

On the other hand, lattice reduction libraries such as [dt21a, AWHT16] implement enumeration algorithms, which also find a shortest vector in a lattice. These algorithms perform an exhaustive search over all lattice points within a given target radius by exploiting the properties of projected sublattices. Enumeration has a worst case time complexity of $d^{\frac{1}{2e}d+o(d)}$ [Kan83, HS07] but requires only polynomial memory.

While, with respect to running time, sieving already compares favourably in relatively low dimensions to simple enumeration[1] (Fincke–Pohst enumeration [FP85] without pruning), the Darmstadt Lattice Challenge 'halls of fame' for both approximate SVP [GS10] and LWE [GY15] challenges have been traditionally dominated by results obtained using enumeration. Sieving has therefore not, so far, been competitive in practical dimensions when compared to state of the art enumeration with heavy preprocessing [Kan83, MW15] and (extreme) pruning [GNR10] as implemented in e.g. FPLLL/FPyLLL [dt21a, dt21b]. Here, 'pruning' means to forego exploring the full search space in favour of focussing on likely candidates. The extreme pruning variant proceeds by further shrinking the search space, and rerandomising the input and restarting the search on failure. In this context 'heavy preprocessing' means running strong lattice reduction, such as the BKZ algorithm [SE94, CN11], which in turn runs enumeration in smaller dimensions, before performing the full enumeration. In short, enumeration currently beats sieving 'in practice' despite having an asymptotically worse running time. Thus [MW15], relying on the then state of the art, estimated the crossover point between sieving and enumeration for solving the Shortest Vector Problem (SVP) as dimension $d = 146$ (or in the thousands, assuming that extreme pruning can be combined with heavy preprocessing without a loss of performance).

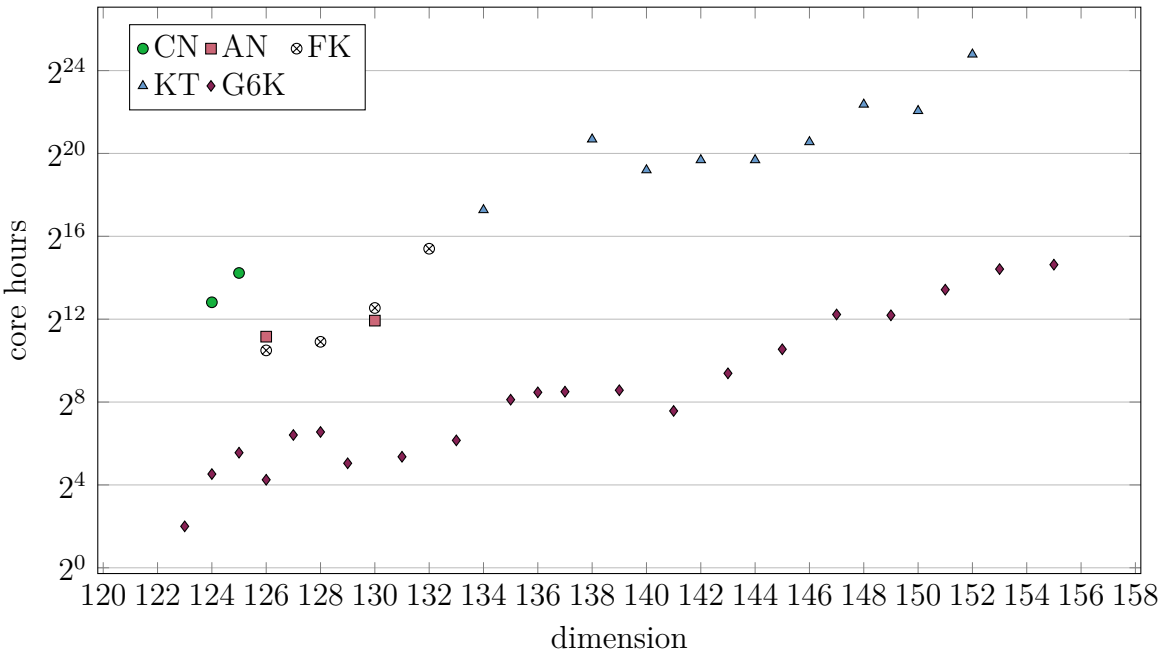
Contribution

In this work, we report performance records for achieving various lattice reduction tasks using sieving. For exact SVP, we are able to outperform the pruned enumeration of FPLLL/FPyLLL by dimension 70. For the Darmstadt SVP Challenges we solve previously unsolved challenges in dimensions $\{151, 153, 155\}$ (see Figure 3.1 and Table 3.2), and our running times are at least 400 times smaller than the previous records for comparable instances.

We also solved new instances $(n, \alpha) \in \{(40, 0.005), (50, 0.015), (55, 0.015), (60, 0.01), (65, 0.01), (75, 0.005)\}$ of the Darmstadt LWE challenge (see Table 3.3). For this, we adapted the strategy of [LN13], which consists of running one large enumeration after a BKZ tour of small enumerations, to G6K. This slightly generalises the prediction of [ADPS16, AGVW17].

[1] For example, the Gauss sieve implemented in FPLLL (`latsieve`) beats FPLLL's *unpruned* SVP oracle (`fplll -a svp`) in dimension 50.

[2] Their latest record (SVP-152) from Oct. 2018 is only reported in the HoF. It reports a computation time of 800K CPU hours. According to personal communications with the authors, this translates to $36 \cdot 800\text{K} = 28.8\text{M}$ core hours.



CN: Chen–Nguyen (HoF), BKZ+enum;
AN: Aono–Nguyen (HoF), BKZ+enum;
FK: Fukase–Kashiwabara, RSR [FK15];
KT: Kashiwabara–Teruya, RSR [TKH18];[2]
G6K: `WorkOut` with `bgj1` sieve (this work).
'HoF' means data was extracted from the Darmstadt SVP Challenge Hall of Fame [GS10].
Raw data embedded.

Figure 3.1 New Darmstadt SVP Challenge records.

Our sieving performance is enabled by building on, generalising and extending previous works. In particular, the landscape of enumeration and sieving started to change recently with [Duc18a, LM18]. For example, [Duc18a] speculated that the crossover point, for solving SVP, between the SubSieve proposed therein and pruned enumeration would be around $d = 90$ if combined with faster sieving than [MV10c]. A key ingredient for this performance gain was the realisation of several 'dimensions for free' by utilising heavy preprocessing and 'Babai lifting' (given in this thesis as Algorithm 5) over said free dimensions. This may be viewed as a hybrid of pruned enumeration with sieving, and is enabled by strong lattice reduction preprocessing. In other words, we may consider these improvements as applying lessons learnt from enumeration to sieving algorithms. It is worth recalling here that the fastest enumeration algorithm relies on the input basis being quasi HKZ reduced [Kan83], but prior to [Duc18a, LM18] sieving was largely oblivious to the quality of the input basis. Furthermore, both [Duc18a, LM18] suggest to exploit the fact that sieving algorithms hold a database of many short vectors, for example by recycling them in future sieving steps. Thus, instead of treating sieving as an SVP oracle outputting a single vector, they implicitly treat it as a stateful machine where the state comprises the current basis and a database of many relatively short vectors.

G6K, an abstract stateful machine

In this work, we embrace and push forward in this direction. After recalling some preliminaries in Section 3.3, we propose the General Sieve Kernel (G6K, pronounced /ʒe.si.ka/) in Section 3.4, an abstract machine for running sieving algorithms, and driving lattice reduction. We define several basic instructions on this stateful machine that not only allow new sieving strategies to be simply expressed and easily prototyped, but also lend themselves to the easy inclusion and extension of previous works. For example, the progressive sieves from [Duc18a, LM18] can be concisely written as

$$\texttt{Reset}_{0,0,0},\ (\texttt{ER},\ \texttt{S})^d,\ \texttt{I}_0$$

where `S` means to sieve, $\texttt{I}_0$ means to insert the shortest vector found into the basis, `ER` means to increase the sieving dimension and `Reset` initialises the machine.

Beyond formalising previous techniques, our machine provides new instructions, namely `EL`, which allows one to increase the sieving dimension 'towards the left' (of the basis), and an insertion instruction `I` which is no longer terminal: it is possible to resieve after an insertion, contrary to [Duc18a]. These instructions increase the

range of implementable strategies and we make heavy use of them to achieve the above results.

The General Sieve Kernel also introduces new tricks to further improve efficiency. First, all vectors encountered during the sieve can be lifted 'on-the-fly' (as opposed to only the final set of vectors in [Duc18a]) offering a few extra dimensions for free and thus improved performance. Additionally, G6K keeps insertion candidates for many positions so as to allow a posteriori choices of the most reducing insertion, akin to Deep LLL [SE94] and the latest variants of Random Sampling Reduction (RSR) [TKH18], enabling stronger preprocessing.

Lattice reduction with G6K

Using these instructions, in Section 3.5 we then create reduction strategies for various tasks (SVP, BKZ like reduction). These strategies encapsulate and extend the contributions and the suggestions made in [Duc18a, LM18], further exploiting the features of G6K. Using the instructions of our abstract stateful machine, our fundamental operation, named the `Pump`, may be written as

$$\texttt{Reset}_{\kappa,\kappa+\beta,\kappa+\beta},\ (\texttt{EL},\ \texttt{S})^{\beta-f},\ (\texttt{I},\ \texttt{S}^s)^{\beta-f}.$$

While previous works mostly focus on recursive lattice reduction within sieving, we also explicitly treat and test utilising sieving within the BKZ algorithm. Here, we report both negative and positive results. On the one hand, we report that, at least in our implementation, the elegant idea of a sliding window sieve for BKZ [LM18] performs poorly and offer a discussion as to why. We also find that the strategy from [Duc18a], consisting of 'overshooting' the block size β of BKZ by a small additive factor combined with 'jumping' over the same number of indices in a BKZ tour, does not provide a beneficial quality vs. time trade off. On the other hand, we find that from the second block of a BKZ tour onwards, or always in the progressive BKZ case, cheaper sieving calls (involving less preprocessing) suffice. We also find that opportunistically increasing the number of dimensions for free beyond the optimal values for solving SVP improves the quality vs. time trade off. Thus, we vindicate the suggestion to move beyond treating sieving merely as an SVP oracle in BKZ.

Implementation

In Section 3.6, we then propose and describe an open source, tweakable, multithreaded, low level optimised implementation of G6K, featuring several sieve variants [MV10c,

BGJ15, HK17]. Our implementation is carefully optimised to support multiple cores in all time consuming operations, is highly parameterised and makes heavy use of the SimHash test [Cha02, FBB+15, Duc18a]. It combines a C++ kernel with a Python control module. Thus, our higher level algorithms are all implemented in Python for easy experimentation. Our implementation is written with a view towards being extensible and reusable and comes with documentation and tests. We consider hackable and usable software a contribution in its own right.

Performance and records

Using and tuning our implementation of G6K then allows us to obtain the variety of performance records for solving lattice challenges described above. We describe our approach in Section 3.7. There, we also describe our experiments for the aforementioned BKZ strategies.

Complementary information on the performance of our implementation is provided in [ADH+19b]: in [ADH+19b, App. A] we give a feature by feature improvement report, and [ADH+19b, App. B] assesses the parallelism efficiency.

Discussion

A natural question is how our results affect the security of lattice based schemes, especially the NIST PQC candidates. Most candidates have been extremely conservative, and thus we do not expect the classical security claim of any scheme to be directly affected by our results. We note, however, that our results on BKZ substantiate further the prediction made in several analyses of NIST PQC candidates that the cost of the SVP oracle can be somewhat amortised in BKZ [PAA+17, Sec 4.2.6]. Thus, our results provide further evidence that the $8\,d \cdot C_{SVP}$ cost model [ACD+18] is an overestimate, but they nevertheless do not reach the lower bound given by the 'core hardness' estimates. However, we stress that our work justifies the generally conservative approach and we warn against security estimates based on a state of the art that is still in motion.

On the other hand, the memory consumption of sieving eventually becomes a difficult issue for implementation, and could incur slowdowns due to memory access delays and bandwidth constraints. However, it is not so clear that these difficulties are insurmountable, especially to an attacker with access to custom hardware. For example Kirchner claimed [Kir16] that simple sieving algorithms such as the Nguyen–Vidick sieve are implementable by a circuit with Area = Time = $2^{0.2075n+o(n)}$. Ducas further conjectured [Duc18b] that `bgj1` (a simplified version of [BGJ15]) can be implemented

with Area $= 2^{0.2075n+o(n)}$ and Time $= 2^{0.142n+o(n)}$. More concretely, the algorithms that we have implemented mostly consider contiguous streams of data, making the use of disks instead of RAM plausibly not so penalising.

One may also argue that such an area requirement on its own is already unreasonable. Yet, such arguments should also account for what amount of walltime is considered reasonable. For example, the walltime of a bruteforce search costing 2^{128} CPU cycles on 2^{64} cores at 4GHz runs for 2^{64} cycles $= 2^{32}$ seconds ≈ 134 years; larger walltimes with fewer cores can arguably be considered irrelevant for practical attacks.

3.3 Preliminaries

3.3.1 Notations and basic definitions

We index from zero. The below repeats some preliminaries from this thesis, but mostly to highlight this temporary change of first index. We write a matrix $\mathbf{B}$ as $\mathbf{B} = (\mathbf{b}_0 \cdots \mathbf{b}_{n-1})$ where $\mathbf{b}_i$ is the i^{th} column vector of $\mathbf{B}$. We denote by $\mathbf{B}^* = (\mathbf{b}_0^* \cdots \mathbf{b}_{n-1}^*)$ the Gram–Schmidt orthogonalisation of the matrix $\mathbf{B} = (\mathbf{b}_0 \cdots \mathbf{b}_{n-1})$. By performing $\mathbf{b}_i \leftarrow \mathbf{b}_i - \lfloor \mu_{i,j} \rceil \mathbf{b}_j$ type operations we can size reduce a basis vector (Algorithm 1) and size reduce a basis (Algorithm 2). By considering a vector $\mathbf{t} \in \mathbb{R}^d$ we can also size reduce an arbitrary vector (Algorithm 3) which leads us to Babai's Nearest Plane algorithm [Bab86] for CVP type problems (Algorithm 4). Finally, these techniques are used to 'Babai lift' in the dimensions for free techniques [Duc18a] used in this work (Algorithm 5). In the above algorithms the range 1 to n is appropriately shifted to 0 to $n-1$, and most subtly in Line 3 of Algorithm 5 we have $\mathbf{B}' \in \mathbb{R}^{d \times \ell}$ and $\mathbf{B}'' \in \mathbb{R}^{d \times (n-\ell)}$.

We also define the orthonormal vectors $\mathbf{b}_i^\circ = \mathbf{b}_i^* / \langle \mathbf{b}_i^*, \mathbf{b}_i^* \rangle$ and extend this to the orthonormal basis $\mathbf{B}^\circ$ columnwise. For $i \in \{0, \ldots, n-1\}$, we denote the projection orthogonally to the span of $(\mathbf{b}_0 \cdots \mathbf{b}_{i-1})$ by π_i. In particular $\pi_i(\mathbf{b}_i) = \mathbf{b}_i^*$ and π_0 is the identity. When discussing the insertion operation in Section 3.4 we will write $\pi_{\mathbf{B},i}$ to be precise about which basis we are projecting against. For $0 \leq \ell < r \leq n$, we denote by $\mathbf{B}_{[\ell:r]}$ the local projected basis, $(\pi_\ell(\mathbf{b}_\ell) \cdots \pi_\ell(\mathbf{b}_{r-1}))$. When the basis is clear from context $\Lambda_{[\ell:r]}$ denotes $\Lambda(\mathbf{B}_{[\ell:r]})$, the lattice generated by $\mathbf{B}_{[\ell:r]}$. If $r = n$ we refer to $\mathbf{B}_{[\ell]}$ and $\Lambda_{[\ell]}$. Note that $\Lambda_{[d-i]}$ therefore represents a block of rank i that extends to the end of the basis, rather than $\Lambda_{[d-i+1]}$ as when indexing from 1. Similarly, note that when indexing from 0 we have $\|\mathbf{b}_i^*\| = \gamma^i \cdot \|\mathbf{b}_0\|$ in Definition 2.4.9, and that the expected squared length under the projection π_i in Definition 2.4.10 is $(m-i) \cdot \sigma^2$.

We refer to the *left* (resp. the *right*) of a *context* $[\ell : r]$ and by 'the context $[\ell : r]$' implicitly refer also to $\Lambda_{[\ell:r]}$ and $\mathbf{B}_{[\ell:r]}$. More generally, we speak of the *left* (resp. the *right*) as a direction to refer to smaller or earlier (resp. larger or later) indices and of contexts becoming larger as $r - l$ grows.

3.3.2 Sieving, lattice reduction and heuristics

Recall the definitions of an LLL reduced basis, an HKZ reduced basis, and a BKZ reduced basis from Section 2.4.1, as well as the Gaussian heuristic of Definition 2.2.25, and the contents of Section 2.4.2.

Sieving algorithms build databases of lattice vectors, exponentially sized in the lattice dimension. In the simplest sieves, it is checked whether the sum or difference of a pair of database vectors is shorter than one of the summands or differands. More importantly for G6K as an abstract stateful machine is the heuristic 'saturation' property of sieving discussed in Definition 2.5.6 that, after sieving in some Λ, this database contains many elements of $\{\mathbf{w} \in \Lambda\colon\ \|\mathbf{w}\| \leq R \cdot \mathrm{gh}(\Lambda)\}$. Here R is a small constant determined by the sieve (see Section 3.6.1). It is this information that G6K will leverage when changing context and inserting.

> In this work the termination condition of a sieve ensures that the database of the sieve contains a constant fraction of the expected number of lattice vectors of length no greater than $R \cdot \mathrm{gh}(\Lambda)$. This replaces previous conditions such as counting the number of 'collisions' [MV10c]. The name given to this condition is the 'saturation' condition, and a sieve database that satisfies it is said to be 'saturated'.

3.4 The General Sieve Kernel

3.4.1 Design principles

In this section we propose the General Sieve Kernel (Version 1.0), an abstract machine supporting a wide variety of lattice reduction strategies based on sieving algorithms. It minimises the sieving computation effort for a given reduction quality by

- offering a mechanism to recycle short vectors from one context to somewhat short vectors in an overlapping context, therefore already starting the sieve

closer to completion. This formalises and generalises some of the ideas proposed in [Duc18a, LM18].

- being able to lift vectors to a larger context than the one currently considered. These vectors are considered for insertion at earlier positions. But as an extension to [Duc18a], which only lifted the final database of vectors, G6K is able to lift all vectors encountered during the sieve. From this, we expect a few extra dimensions for free.[3]

- deferring the decision of where to insert a short vector until after the search effort. This is contrary to formal definitions of more standard reduction algorithms, e.g. BKZ or Slide [GN08a] reduction, and inspired by Deep LLL and recent RSR variants [TKH18].

The underlying computations per vector are reasonably cheap, typically linear or quadratic in the dimension of the vector currently being considered. The most critical operation, namely the SimHash test [Cha02, FBB+15, Duc18a], consists in practice of about a dozen x86 non vectorised instructions for vectors of dimension roughly one hundred.

3.4.2 Vectors, contexts and insertion

All vectors considered by G6K live in one of the projected lattices $\Lambda_{[\ell:r]}$ of a lattice Λ. More specifically, they are represented in basis $\mathbf{B}_{[\ell:r]} \in \mathbb{R}^{d\times n}$ as integral vectors $\mathbf{v} \in \mathbb{Z}^n$ where $n = r - \ell$, i.e. we have $\mathbf{w} = \mathbf{B}_{[\ell:r]} \cdot \mathbf{v}$ for some $\mathbf{w} \in \Lambda_{[\ell:r]} \subset \mathbb{R}^d$. Throughout, we may represent the (projected) lattice vector $\mathbf{w}$ by the vector $\mathbf{v}$. It is convenient and efficient to also keep a representation, $\mathbf{v}^\circ \in \mathbb{R}^n$, of $\mathbf{w}$ in the orthonormalised basis $\mathbf{B}^\circ_{[\ell:r]}$; $\mathbf{w} = \mathbf{B}^\circ \cdot \mathbf{v}^\circ$. This conversion costs $O(n^2)$.

Below we list the three operations that *extend* or *shrink* a vector to the left or to the right. While these operations act on an element of the lattice, i.e. $\mathbf{w}$, we define them by their actions on the integer representatives. For example if $\mathbf{w} = \mathbf{B}_{[\ell:r]} \cdot \mathbf{v}$ then $\mathrm{er}(\mathbf{w}) = \mathbf{B}_{[\ell:r+1]} \cdot (v_0, \ldots, v_{n-1}, 0)^t$.

[3] Lifting is somewhat more expensive than checking the sum and difference of a pair of vectors. We are therefore careful to only lift a fraction of all encountered vectors, namely only those with length below, say, $\sqrt{1.8} \cdot \mathrm{gh}(\Lambda_{[\ell:r]})$.

- Extend Right (inclusion) er : $\Lambda_{[\ell:r]} \to \Lambda_{[\ell:r+1]}$

$$(v_0, \ldots, v_{n-1}) \mapsto (v_0, \ldots, v_{n-1}, 0)$$
$$(v_0^\circ, \ldots, v_{n-1}^\circ) \mapsto (v_0^\circ, \ldots, v_{n-1}^\circ, 0)$$

- Shrink Left (projection) sl : $\Lambda_{[\ell:r]} \to \Lambda_{[\ell+1:r]}$

$$(v_0, \ldots, v_{n-1}) \mapsto (v_1, \ldots, v_{n-1})$$
$$(v_0^\circ, \ldots, v_{n-1}^\circ) \mapsto (v_1^\circ, \ldots, v_{n-1}^\circ)$$

- Extend Left (Babai lift) el : $\Lambda_{[\ell:r]} \to \Lambda_{[\ell-1:r]}$

$$(v_0, \ldots, v_{n-1}) \mapsto (\lfloor -c \rceil, v_0, \ldots, v_{n-1})$$
$$(v_0^\circ, \ldots, v_{n-1}^\circ) \mapsto ((c + \lfloor -c \rceil) \cdot \left\|\mathbf{b}_{\ell-1}^*\right\|, v_0^\circ, \ldots, v_{n-1}^\circ),$$
$$\text{where } c = \sum_{j=0}^{n-1} \mu_{\ell+j,\ell-1} \cdot v_j.$$

This final operation, el, is Algorithm 5 where we set $\mathbf{B} = \mathbf{B}_{[\ell-1:r]}$ and $\mathbf{B}_{[1]} = \mathbf{B}_{[\ell:r]}$. The list $L \subset \mathbf{B}_{[1]}$ has a single element $\mathbf{w}_{[1]} = \mathbf{B}_{[1]} \cdot \mathbf{v}_{[1]}$, and we write $\mathbf{v}_{[1]} = (v_0, \ldots, v_{n-1})$. This single element $\mathbf{w}_{[1]}$ is the vector of $\Lambda_{[\ell:r]}$ we are applying el to. Note that since we are counting from 0, on Line 3 of Algorithm 5 we have $\mathbf{B}' \in \mathbb{R}^{d \times 1}$ and $\mathbf{B}'' \in \mathbb{R}^{d \times (n-1)}$ even though we are considering $\mathbf{B}_{[1]}$. We have $\mathbf{B}' = (\pi_{\ell-1}(\mathbf{b}_{\ell-1})) = (\mathbf{b}_{\ell-1}^*)$ and $\mathbf{B}'' = (\pi_{\ell-1}(\mathbf{b}_l) \cdots \pi_{\ell-1}(\mathbf{b}_{r-1}))$. We construct the target $\mathbf{t} = \mathbf{B}'' \cdot \mathbf{v}_{[1]}$ and, tracing through to Algorithm 3, must determine $-\mathbf{t}_{\mathbf{B}'}$, the negation of its component in $\mathrm{span}_{\mathbb{R}}(B')$. We have

$$\mathbf{t} = \sum_{j=0}^{n-1} v_j \cdot \pi_{\ell-1}(\mathbf{b}_{\ell+j}) = \sum_{j=0}^{n-1} v_j \left(\mathbf{b}_{\ell+j}^* + \sum_{k=\ell-1}^{\ell+j-1} \mu_{\ell+j,k} \mathbf{b}_k^* \right).$$

The component of this in $\mathbf{B}'$ is exactly the terms including $\mathbf{b}^*_{\ell-1}$, hence we find

$$-\mathbf{t}_{\mathbf{B}'} = -\left(\sum_{j=0}^{n-1} \mu_{\ell+j,\ell-1} \cdot v_j\right) \mathbf{b}^*_{\ell-1}.$$

More concisely we write $-\mathbf{t}_{\mathbf{B}'} = -c \cdot \mathbf{b}^*_{\ell-1}$ for c the bracketed term in the expression above. Tracing back up to Algorithm 5 we find $\mathbf{w}' = \lceil -c \rfloor \mathbf{b}^*_{\ell-1}$ on Line 6. Note that we subtract a multiple of $\mathbf{b}^*_{\ell-1}$ because $\mathbf{B}' = (\mathbf{b}^*_{\ell-1})$. Therefore $\mathbf{w}' + \mathbf{t}$ is described exactly by the coefficients $(\lfloor -c \rceil, v_0, \ldots, v_{n-1})^t$ in basis $\mathbf{B}_{[\ell-1:r]}$. For the relation in the orthonormal basis one must find the constant γ such that

$$\gamma \cdot \mathbf{b}^\circ_{\ell-1} + \sum_{j=0}^{n-1} v^\circ_j \cdot \mathbf{b}^\circ_{\ell+j} = \lceil -c \rfloor \mathbf{b}^*_{\ell-1} + \sum_{j=0}^{n-1} v_j \cdot \pi_{\ell-1}(\mathbf{b}_{\ell+j})$$

given that

$$\sum_{j=0}^{n-1} v^\circ_j \cdot \mathbf{b}^\circ_{\ell+j} = \sum_{j=0}^{n-1} v_j \cdot \pi_\ell(\mathbf{b}_{\ell+j}).$$

A fairly tedious calculation shows that $\gamma = c + \lceil -c \rfloor$. We have size reduced, or 'Babai lifted', over a single dimension – to lift over more dimensions el is iterated the desired number of times. Note that this procedure does not require the $\mu_{i,j}$ relating $\mathbf{B}$ and $\mathbf{B}^*$ to be updated; we are ultimately *using* the basis to perform Algorithm 3, rather than *altering* it.

While it would seem natural to also define a Shrink Right operation, we have not found a geometrically meaningful way of doing so. Moreover, we have no algorithmic purpose for it.

Insertion

Performing an insertion (the elementary lattice reduction operation) of a vector is less straightforward. For $i \leq \ell < r$, $n' = r - i$, $n = r - \ell$ an insertion of a vector $\mathbf{w}$ at position i is a local change of basis making $\mathbf{w} = \mathbf{B}_{[i:r]} \cdot \mathbf{v}$ the first vector of the new local projected basis, i.e. applying a unimodular matrix $\mathbf{U} \in \mathbb{Z}^{n' \times n'}$ to $\mathbf{B}_{[i:r]}$ such that $(\mathbf{B}_{[i:r]} \cdot \mathbf{U})[0] = \mathbf{w}$. While doing so, we would like to recycle a database of vectors living in the context $[\ell : r]$. This prevents us from using the usual strategy of forming a

generating set and LLL reducing it, as on Line 12 of Algorithm 7, as the effect this may have on the database of vectors we wish to recycle may be complex.

In the case $i = \ell$, this causes no difficulties, and one could apply any change of basis $\mathbf{U}$ to the database.

> In particular when $i = \ell$ we want a unimodular $\mathbf{U}$ such that $\mathbf{w} = \mathbf{B}_{[\ell:r]} \cdot \mathbf{U} \cdot (1, 0, \ldots, 0)^t$ given that $\mathbf{w} = \mathbf{B}_{[\ell:r]} \cdot \mathbf{v}$, which implies the first column of $\mathbf{U}$ must be $\mathbf{v}$. Note that any basis $\mathbf{B}$ such that $\Lambda(\mathbf{B}) = \mathbb{Z}^n$ is unimodular. By Lemma 2.4.2, provided there is no $\alpha \in (0, 1)$ such that $\alpha \cdot \mathbf{v} \in \mathbb{Z}^n$, we may extend $\mathbf{v}$ to a basis for $\mathbb{Z}^n$, and therefore form the required unimodular matrix $\mathbf{U}$. If such an α exists we immediately have a shorter $\alpha \cdot \mathbf{w} \in \Lambda_{[\ell:r]}$.
> We can easily recycle other lattice vectors in the database; if $\mathbf{w} = \mathbf{B}_{[\ell:r]} \cdot \mathbf{v}$ is any lattice vector in our database, after this transform we similarly transform $\mathbf{v} \mapsto \mathbf{U}^{-1} \cdot \mathbf{v}$.

But to exploit dimensions for free, we will typically have $i < \ell$, which is more delicate. If we can ensure that

$$\mathrm{span}_{\mathbb{R}}\left((\mathbf{B}_{[i:r]} \cdot \mathbf{U})[0 : \ell - i + 1]\right) = \mathrm{span}_{\mathbb{R}}\left(\mathbf{B}_{[i:\ell]} \cup \{\mathbf{w}\}\right) \tag{3.1}$$

then one can simply project all the database vectors orthogonally to $\mathbf{w}$, to end up with a database in a new smaller context $[\ell + 1 : r]$. If it holds that $\mathbf{v}[j] = \pm 1$ for some $j \in \{\ell - i, \ldots, n' - 1\}$ an appropriate matrix $\mathbf{U}$ can be constructed as

$$\mathbf{U} = \left(\begin{array}{c|cc} & \mathbf{I}_{j\times j} & \mathbf{0}_{j\times n'-j-1} \\ \mathbf{v} & \mathbf{0}_{1\times j} & \mathbf{0}_{1\times n'-j-1} \\ & \mathbf{0}_{n'-j-1\times j} & \mathbf{I}_{n'-j-1\times n'-j-1} \end{array}\right). \tag{3.2}$$

> This $\mathbf{U}$ is a row permutation of an integer lower triangular matrix with the diagonal all ones (resp. $n' - 1$ ones and one minus one) if $\mathbf{v}[j] = 1$ (resp. $\mathbf{v}[j] = -1$). It is therefore unimodular. Condition (3.1)

is satisfied by computation,

$$\begin{aligned}
\left(\mathbf{B}_{[i:r]} \cdot \mathbf{U}\right)[0] &= \mathbf{w}, \\
\left(\mathbf{B}_{[i:r]} \cdot \mathbf{U}\right)[1] &= \mathbf{B}_{[i:r]}[0] = \mathbf{B}_{[i:l]}[0], \\
&\vdots \\
\left(\mathbf{B}_{[i:r]} \cdot \mathbf{U}\right)[\ell - i] &= \mathbf{B}_{[i:r]}[\ell - i - 1] = \mathbf{B}_{[i:l]}[\ell - i - 1], \\
\left(\mathbf{B}_{[i:r]} \cdot \mathbf{U}\right)[\ell - i + 1] &= \mathbf{B}_{[i:r]}[\ell - i], \\
&\vdots \\
\left(\mathbf{B}_{[i:r]} \cdot \mathbf{U}\right)[j] &= \mathbf{B}_{[i:r]}[j - 1], \\
\left(\mathbf{B}_{[i:r]} \cdot \mathbf{U}\right)[j + 1] &= \mathbf{B}_{[i:r]}[j + 1], \\
&\vdots \\
\left(\mathbf{B}_{[i:r]} \cdot \mathbf{U}\right)[n' - 1] &= \mathbf{B}_{[i:r]}[n' - 1]
\end{aligned}$$

As we have chosen $j \geq \ell - i$, $\left(\mathbf{B}_{[i:r]} \cdot \mathbf{U}\right)[0], \ldots, \left(\mathbf{B}_{[i:r]} \cdot \mathbf{U}\right)[j]$ contains $\mathbf{w}$ through $\mathbf{B}_{[i:l]}[\ell - i - 1]$ as above, and hence the first $\ell - i + 1$ columns of $\mathbf{B}_{[i:r]} \cdot \mathbf{U}$ satisfy (3.1).

We now wish to calculate how to represent the sieve database vectors after $\mathbf{w}$ is inserted. Let $\mathbf{y} = \mathbf{B}_{[i:r]} \cdot \mathbf{x}$ be such that $\pi_{\mathbf{B},\ell}(\mathbf{y}) \in \Lambda_{[\ell:r]}$ is a sieve database vector before $\mathbf{w}$ is inserted in the basis. We take $\mathbf{x} = (\overbrace{0, \ldots, 0}^{\ell - i}, *, \ldots, *) \in \mathbb{Z}^{n'}$ without loss of generality. Let also $\mathbf{U}^{\text{FULL}}$ be the unimodular matrix $\mathbf{U}$ extended to act on the entire basis $\mathbf{B}$, rather than just the context $[i : r]$, as follows

$$\mathbf{U}^{\text{FULL}} = \begin{pmatrix} \mathbf{I}_i & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{U} & 0 \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{d-r} \end{pmatrix}.$$

Finally, let the new basis be $\mathbf{C} = \mathbf{B} \cdot \mathbf{U}^{\text{FULL}}$. After the insertion we wish to calculate $\pi_{\mathbf{C},\ell+1}(\mathbf{y}) \in \Lambda'_{[\ell+1:r]}$ with $\Lambda' = \Lambda(\mathbf{C})$ as the corresponding new database vector. Indeed, the first $\ell + 1$ columns of $\mathbf{C}$ are $\mathbf{b}_0, \ldots \mathbf{b}_{i-1}, \mathbf{w}, \mathbf{b}_i, \ldots \mathbf{b}_{\ell-1}$ and so $\pi_{\mathbf{C},\ell+1}(\mathbf{y})$ is exactly the projection of $\pi_{\mathbf{B},\ell}(\mathbf{y})$ against $\mathbf{w}$. Since $\mathbf{B}_{[i:r]} \cdot \mathbf{U}$ satisfies (3.1), this is

what we require. We have

$$\begin{aligned}\pi_{\mathbf{C},\ell+1}(\mathbf{y}) &= (\pi_{\mathbf{w}}^{\perp} \circ \pi_{\mathbf{B},\ell})(\mathbf{y}) \\ &= (\pi_{\mathbf{w}}^{\perp} \circ \pi_{\mathbf{B},\ell})(\mathbf{B}_{[i:r]} \cdot \mathbf{U} \cdot \mathbf{U}^{-1} \cdot \mathbf{x}) \\ &= (\pi_{\mathbf{w}}^{\perp} \circ \pi_{\mathbf{B},\ell})(\mathbf{B}_{[i:r]} \cdot \mathbf{U}) \cdot \mathbf{U}^{-1} \cdot \mathbf{x}.\end{aligned}$$

The first $\ell - i + 1$ columns of $\mathbf{B}_{[i:r]} \cdot \mathbf{U}$ are $\mathbf{w}, \pi_{\mathbf{B},i}(\mathbf{b}_i), \ldots, \pi_{\mathbf{B},i}(\mathbf{b}_{\ell-1})$ which each become $\mathbf{0}$ under the projection $\pi_{\mathbf{w}}^{\perp} \circ \pi_{\mathbf{B},\ell}$. Hence one calculates the bottom $r - \ell - 1$ entries of $\mathbf{U}^{-1} \cdot \mathbf{x}$ to store the new database vector from $\Lambda'_{[\ell+1:r]}$ in terms of basis $\mathbf{C}_{[\ell+1:r]}$.

However, it is important that the local projected bases remain somewhat reduced. If not, numerical stability issues may occur. Moreover, the condition that $\mathbf{v}$ contains a ± 1 in the context $[\ell : r]$ is often not satisfied without sufficient reduction. While we must be careful to not alter the vector space inside the sieving context, we can nevertheless perform a full size reduction (upper triangular matrix $\mathbf{T}$ with unit diagonal) on the whole of $\mathbf{B}_{[i:r]}$, as well as two local LLL reductions $\mathbf{U}_{\mathrm{L}}$ and $\mathbf{U}_{\mathrm{R}}$ on $\mathbf{B}_{[i:\ell+1]}$ and $\mathbf{B}_{[\ell+1:r]}$.

$$\mathbf{U}' = \mathbf{U} \cdot \mathbf{T} \cdot \begin{pmatrix} \mathbf{U}_{\mathrm{L}} & \mathbf{0} \\ \mathbf{0} & \mathbf{U}_{\mathrm{R}} \end{pmatrix}. \tag{3.3}$$

Note that $\mathrm{span}_{\mathbb{R}}\left((\mathbf{B}_{[i:r]} \cdot \mathbf{U}')[0 : \ell - i + 1]\right) = \mathrm{span}_{\mathbb{R}}\left((\mathbf{B}_{[i:r]} \cdot \mathbf{U})[0 : \ell - i + 1]\right)$, so that condition (3.1) is preserved.

3.4.3 G6K: a stateful machine

The General Sieve Kernel is defined by the following internal states and instructions.

State

- A lattice basis $\mathbf{B} \in \mathbb{Z}^{d \times d}$, updated each time an insert is made (Section 3.4.2). Associated with it is its Gram–Schmidt Orthonormalisation basis $\mathbf{B}^{\circ}$.
- Positions $0 \leq \kappa \leq \ell \leq r \leq d$. We refer to the context $[\ell : r]$ as the *sieving context*, and $[\kappa : r]$ as the *lifting context*. We define $n = r - \ell$ (the sieving dimension).
- A database *db* of N vectors in $\Lambda_{[\ell:r]}$ (preferably short).
- Insertion candidates $\mathbf{c}_\kappa, \ldots, \mathbf{c}_\ell$ where $\mathbf{c}_i \in \Lambda_{[i:r]}$ or $\mathbf{c}_i = \perp$.

Instructions

- Initialisation ($\texttt{Init}_{\mathbf{B}}$): initialise the machine with a basis $\mathbf{B} \in \mathbb{Z}^{d \times d}$.
- Reset ($\texttt{Reset}_{\kappa,\ell,r}$): empty database, and set (κ, ℓ, r).
- Sieve (S): run some chosen sieving algorithm. During execution of the algorithm, well chosen visited vectors are lifted from $\Lambda_{[\ell:r]}$ to $\Lambda_{[\kappa:r]}$ (by iterating el just on these vectors). If such a lift improves (i.e. is shorter than) the best insertion candidate $\mathbf{c}_i$ at position i, then it replaces $\mathbf{c}_i$. We call this optional[4] feature *on-the-fly lifting*.
- Extend Right, Shrink Left, Extend Left (ER, SL, EL): increase or decrease ℓ or r and apply er, sl or el to each vector of the database. All three operations maintain the insertion candidates (except for EL which drops $\mathbf{c}_\ell$).
- Insert (I): choose the best insertion candidate $\mathbf{c}_i$ for $\kappa \leq i \leq \ell$, according to a score function, and insert it at position i. The sieving context changes to $[\ell + 1 : r]$ and the database is updated as described in Section 3.4.2. If no insertion candidate is deemed suitable, then we simply run SL to ensure that the sieving context will end up as expected.[5] When we write $\texttt{I}_i$, we mean that insertion is only considered at position i.
- Grow or Shrink ($\texttt{Resize}_N$): change the database to a given size N. When shrinking, remove the longest vectors from the database. When growing, sample new vectors (using some unspecified sampling algorithm[6]). Typically, we will not explicate the calls to these operations, and assume that calling a sieve includes resizing the database to the appropriate size, for example $N = O((4/3)^{n/2})$ for the 2-sieves of [NV08, MV10c, BGJ15].

Our implementation of this machine offers more functionality, such as the ability to monitor its state and therefore the behaviour of the internal sieve algorithm, and to tune the underlying algorithms.

[4] The alternative being to only consider the vectors of the final database for lifting.

[5] We can view SL as the trivial insertion of the vector $\mathbf{w}$ represented by $\mathbf{v} = (1, 0, \ldots, 0)$, with $i = \ell$, $j = 0$, and the projection still occurring.

[6] When possible we prefer to sample by summing random pairs of vectors from the database.

3.5 Reduction algorithms using G6K

Equipped with this abstract machine, we can now reformulate, improve and generalise strategies for lattice reduction with sieving algorithms. In the following we will assume that the underlying sieve algorithm has a time complexity proportional to C^n, for $C > 1$ and n the dimension of the SVP instance, and we also define $C' = 1/(1 - 1/C)$. This second constant approximates the multiplicative overhead $\sum_{i=1}^{n} C^i/C^n$ encountered on iterating sieves in dimensions 1 to n.

Indeed we have the following, where the approximation error tends to 0 as $n \to \infty$ so long as $C > 1$,

$$\begin{aligned}\sum_{i=1}^{n} C^i/C^n &= \frac{1}{C^n} \cdot (C + \dots C^n) \\ &= \frac{1}{C^{n-1}} \cdot \left(1 + \dots + C^{n-1}\right) \\ &= \frac{1}{C^{n-1}} \cdot \frac{C^n - 1}{C - 1} \\ &\approx \frac{1}{1 - 1/C}.\end{aligned}$$

Note that this overhead grows when C decreases. More concretely, depending on the sieve, C can range from 4/3 down to $\sqrt{3/2}$, giving $C' = 4$ up to $C' \approx 5.45$.

3.5.1 The pump

In this section we propose a sequence of instructions called the `Pump`. They encompass the progressive sieving strategy proposed in [Duc18a, LM18] as well as the dimensions for free and multi insertion tricks of [Duc18a]. The original progressive sieving strategy can be written as

$$\texttt{Reset}_{0,0,0},\ (\texttt{ER},\ \texttt{S})^d,\ \texttt{I}_0. \tag{3.4}$$

Similarly, a $\texttt{SubSieve}_f$ which attempts a partial HKZ reduction using sieving with f dimensions for free can be written as

$$\texttt{SubSieve}_f : \texttt{Reset}_{0,f,f},\ (\texttt{ER},\ \texttt{S})^{d-f},\ \texttt{I}_0, \texttt{I}_1, \dots, \texttt{I}_{d-f-1}.^7 \tag{3.5}$$

[7] This sequence refers to $\texttt{SubSieve}^+(\Lambda, f)$ with Sieve being progressive [Duc18a].

We note that due to the newly introduced `EL` operation, it is also possible to perform the progressive sieving from right to left

$$\texttt{Reset}_{0,d,d}, (\texttt{EL},\ \texttt{S})^{d-f},\ \texttt{I}_0, \texttt{I}_1, \ldots, \texttt{I}_{d-f-1}. \tag{3.6}$$

Perhaps surprisingly, the left variant of progressive sieving performs substantially better in experiments. In combination with certain sieving methods, the right variant can even fail completely, this will be discussed in more detail in Section 3.5.5.

To arrive at `Pump`, note first that G6K maintains insertion candidates at many positions. We can therefore relax the insertion positions of (3.6) and choose those that appear to be optimal. The choice of insertion position is discussed in Section 3.5.4.

Secondly, due to on-the-fly lifting, we note that the sequence (3.6) considers many more insertion candidates for the first insertion than for subsequent insertions. Moreover, we noticed that after several insertions the database only contained vectors much longer than recent inserts. By sieving also during the 'descent phase', i.e. when inserting and shrinking the sieve context, we remedy this imbalance and expect to obtain a more strongly reduced basis, ideally obtaining an HKZ reduced context.

In summary, we define the parametrised $\texttt{Pump}_{\kappa,f,\beta,s}$ as the following sequence

$$\texttt{Pump}_{\kappa,f,\beta,s} :\ \texttt{Reset}_{\kappa,\kappa+\beta,\kappa+\beta},\ \overbrace{(\texttt{EL},\ \texttt{S})^{\beta-f}}^{\text{pump-up}},\ \overbrace{(\texttt{I},\ \texttt{S}^s)^{\beta-f}}^{\text{pump-down}}. \tag{3.7}$$

where $0 \le \kappa \le \kappa + \beta \le d$, $0 \le f \le \beta$, and where $s \in \{0, 1\}$ controls whether we sieve during pump-down. One may expect the cost of these extra sieves to be close to a multiplicative factor of 2, but experimentally the factor can reach 3 for certain sieves (e.g. `bgj1`), as more collisions[8] seem to occur during the descent phase. This feature is mostly useful for weaker reduction tasks such as BKZ, see `PumpNJumpBKZTour` below.

3.5.2 SVP

To solve the shortest vector problem on the full lattice, starting from an LLL reduced basis $\mathbf{B}$, we proceed as in [Duc18a], that is, we iterate $\texttt{Pump}_{0,f,d,s}$ for decreasing values of f. While only the last `Pump` delivers the shortest vector, the previous iterations provide a strongly reduced basis (near HKZ reduced), which allows more dimensions for free to be achieved. We expect to obtain further dimensions for free due to on-the-fly lifting.

[8] A collision is when a new vector $\mathbf{w}$ to be inserted into the database equals $\pm\mathbf{w}'$ for some $\mathbf{w}'$ already present in the database.

Similarly, for solving SVP in context $[\kappa : \kappa + \beta]$ (e.g. as a block inside BKZ), we instead make calls to $\texttt{Pump}_{\kappa,f,\beta,s}$ with iteratively decreasing values of f.

Note that we can decrease f in larger increments than 1 to balance the cost of the basis reduction effort and the search for the shortest vector itself. Indeed, with increments of 1, the overhead factor C' for $C = \sqrt{3/2}$ is $C' \approx 4.45$. Decreasing f by 2 gives an overhead of $C' = 1/(1 - C^{-2}) = 3$ and by 3 gives $C' = 1/(1 - C^{-3}) \approx 2.19$. Such speedups are worth losing 1 or 2 dimensions for free.

We therefore define `WorkOut` as the following sequence of `Pump`

$$\begin{aligned}\texttt{WorkOut}_{\kappa,\beta,f,f^+,s} \ : \texttt{Pump}_{\kappa,\beta-f^+,\beta,s},\ \texttt{Pump}_{\kappa,\beta-2f^+,\beta,s},\\ \texttt{Pump}_{\kappa,\beta-3f^+,\beta,s},\ \ldots,\ \texttt{Pump}_{\kappa,f,\beta,s},\end{aligned} \tag{3.8}$$

where $f^+ \geq 1$ is the increment for f mentioned above, and f is the final amount of dimensions for free one is attempting to attain. From experiments on the exact SVP problem and SVP Challenges, we found it worthwhile to deactivate sieving in the descent phase ($s = 0$), though activating it ($s = 1$) is preferable in other settings, or to use less memory at a larger time cost. Similarly, for certain tasks (e.g. the SVP Challenges, $1.05 \cdot v_d^{-1/d}$-HSVP) we found the optimal increment, f^+, to be 2 or 3. This parameter also drives a time memory trade off; setting f^+ to 1 saves on memory by allowing for a larger final f, but at a noticeable cost in time.

When solving exact SVP, it is not clear when to stop this process because we are never certain that a vector is indeed the shortest vector of a lattice (except maybe by running a prohibitively costly non pruned enumeration). In these cases, one should therefore guess, from experimental data, a good number f of dimensions for free. Note that it is is rarely critical to solve exact SVP, and lattice reduction algorithms such as BKZ tolerate approximations.

In some cases, such as the Darmstadt SVP Challenge, we do not have to solve exact SVP, but rather find a vector of a prescribed norm, near the Gaussian heuristic. In this case we do not need to predetermine f and simply iterate the `Pump` within `WorkOut` until satisfaction; rather than set a final value for f one can continue to decrease the number of attempted dimensions for free by f^+ after each `Pump` until a short enough vector is found. As a consequence, we also add an extra option to the `Pump` to allow early aborts when it finds a satisfying candidate $\mathbf{c}_\kappa$. In practice we observe significant savings from this feature, i.e. we observe the `Pump` aborting before reaching its topmost dimension, or at the beginning of the descent phase.

3.5.3 BKZ

Having determined the appropriate parameters f, f^+, s for solving SVP-β (made implicit in the following), a naïve implementation of BKZ[9] is given by the following program

$$\begin{aligned}\texttt{NaiveTour}_\beta : \texttt{WorkOut}_{0,\beta},\ \ \texttt{WorkOut}_{1,\beta+1},\ \ldots \\ \texttt{WorkOut}_{d-\beta,d},\ \ldots,\ \texttt{WorkOut}_{d-1,d}.\end{aligned} \tag{3.9}$$

Several strategies to amortise the cost of sieving inside BKZ were suggested in [Duc18a, LM18]. These aimed to reduce the cost of a tour of BKZ-β below $d-\beta$ (ignoring the smaller final blocks) times the cost of SVP in dimension β. Again, these strategies are implementable as a sequence of G6K instructions.

Namely, the sliding window strategy of [LM18] can be expressed as

$$\texttt{SlidingWindowTour}_\beta : \texttt{Reset}_{0,0,0},\ (\texttt{ER},\ \texttt{S})^\beta,\ (\texttt{I}_\ell,\ \texttt{S},\ \texttt{ER},\ \texttt{S})^{d-\beta},\ (\texttt{I}_\ell,\ \texttt{S})^\beta. \tag{3.10}$$

It is also possible to combine this strategy with the dimensions for free of [Duc18a]. However, there are two caveats. First, it relies on extend right, which is currently problematic in our implementation of G6K, see Section 3.5.5. Secondly, even if this issue is solved, we remark that inside a BKZ tour it is preferable to run LLL on the full basis periodically. From the sandpile point of view [MV10a, HPS11], not doing so implies that a 'bump' accumulates at the right of the reduced blocks, as we try to push the sand to the right. We see no clear strategies to recycle the vectors of a block when calling a full LLL.

Alternatively, [Duc18a] identified two other potential amortisations. First, it is noted that a `WorkOut` (or even just a `Pump`) in a block $[\kappa : \kappa+\beta]$ leaves the next block $[\kappa+1 : \kappa+\beta+1]$ already quite well reduced. It may therefore not be necessary to do a full `WorkOut` on this next block. Instead simply running the last `Pump` of this `WorkOut` may be sufficient, thereby saving up to a factor of C' in the running time.

The second suggestion of [Duc18a] consists of overshooting the blocksize β, so that a `Pump` in dimension $\beta' > \beta$ attempts to HKZ reduce a larger block. In particular for parameter $j \geq 1$, let $\beta' = \beta + j - 1$ and after a $\texttt{Pump}_{\kappa,f,\beta'}$ *jump* by j blocks. This decreases the number of calls to the `Pump` to d/j and may also slightly improve the quality of the reduction, but increases the cost of the `Pump` calls by a factor of C^{j-1}. It is argued that such a strategy could give a speed-up factor ranging from 2.2 to 3.6

[9] Though note that many more insertions will be made than in the standard definition of BKZ.

for a fixed basis reduction quality. By combing the two suggestions of [Duc18a] we therefore perform the following sequence

$$\texttt{PumpNJumpTour}_{\beta',f,j} : \texttt{Pump}_{0,f,\beta'},\ \texttt{Pump}_{j,f,\beta'},\ \texttt{Pump}_{2j,f,\beta'}, \ldots \tag{3.11}$$

We alter the version above to allow for more opportunism. Since choosing f to almost certainly solve exact SVP in blocks is costly, we instead embrace the idea of achieving the most basis reduction from a given sieving context. Extending the lift context, i.e. taking a smaller κ, makes the lift operation more expensive, but gives more insertion candidates, and therefore a new trade off to be optimised over. Note that while $\texttt{Pump}_{\kappa',f+\kappa-\kappa',\beta+\kappa-\kappa'}$ for $\kappa' < \kappa$ takes more dimensions for free than $\texttt{Pump}_{\kappa,f,\beta}$, it still provides the same insertion candidates, $\mathbf{c}_\kappa, \ldots, \mathbf{c}_{\kappa+f}$. This is because the sieving contexts do not shrink. It also provides new insertion candidates $\mathbf{c}_{\kappa'}, \ldots, \mathbf{c}_{\kappa-1}$. Therefore, provided we take care in the first few blocks, the quality cannot decrease. To achieve this one starts with a `Pump` taking no dimensions for free and moves the sieving context right until the desired f' is attained, before continuing as before. Set $f' > f$, $\beta' = \beta + f' - f$ (to fix the sieve context sizes), $\beta'' = \beta' + j - 1$ (to allow jumping),[10] and perform

$$\begin{aligned}\texttt{PumpNJumpTour}_{\beta'',f',j} : \texttt{Pump}_{0,0,\beta''-f'},\ \texttt{Pump}_{0,j,\beta''-f'+j}, \ldots,\ \texttt{Pump}_{0,f',\beta''},\\ \texttt{Pump}_{j,f',\beta''},\ \texttt{Pump}_{2j,f',\beta''}, \ldots\end{aligned} \tag{3.12}$$

3.5.4 Scoring for inserts

The issue of deciding where in a basis to insert given candidates throughout reduction is discussed in [TKH18], in the context of the SVP Challenges. Until the actual shortest vector is found, the purpose of these insertions is to improve the basis quality. Inserting at an early position may degrade the quality of later positions, because we do not know a priori how inserting $\mathbf{c}_i$ will affect $\mathbf{B}_{[\ell:r]}$ for $i < \ell < r$. However, such an insert will be improved upon rarely, as fewer sufficiently short vectors will be found for early positions. Therefore one must find a good trade off between making long lasting yet weak and potentially damaging improvements at early positions, and short lived yet strong improvements at later positions.

One way to achieve this is to use the scoring proposed in [TKH18], a function over the whole basis which measures the global effect of each potential insert, i.e. one that at least checks how inserting $\mathbf{c}_i$ would affect each $\mathbf{B}_{[\ell:r]}$. We use a simplified variant of

[10] For Figure 3.4 we choose yet more opportunism and do not increase β' to β''.

this scoring which scores the improvement of each potential insert according to the following local condition

$$\varsigma_\theta(i) = \begin{cases} 0, & \text{if } \mathbf{c}_i = \perp \\ \theta^{-i} \cdot \|\mathbf{b}_i^*\|^2 / \|\mathbf{c}_i\|^2, & \text{otherwise} \end{cases} \tag{3.13}$$

for some constant $\theta \geq 1$, and take the maximum over i. Setting $\theta = 1$ corresponds to always choosing the 'most improving' candidate, in terms of the ratio of lengths. Setting θ quite large (say 10) corresponds to always inserting at the earliest position. If $\max\{\varsigma_\theta(i) \colon i \in \{\kappa, \ldots, \ell\}\} = 0$, perform `SL`.

To optimise θ, we ran $\texttt{WorkOut}_{0,d,f}$ for $f = 30$ and $d = 110$, measured $\gamma = \mathrm{gh}(\Lambda)/\mathrm{gh}(\Lambda_{[f]})$, and chose $\theta = 1.04$ which minimised this quantity γ. We recall [Duc18a] that γ must be below a certain threshold to guarantee the success of exact SVP in dimension d with f dimensions for free.

The optimal value of θ may differ depending on other parameters, e.g. dimension, approximation factor, and use case, e.g. exact SVP, α-HSVP, BKZ, and the question of optimising insertion strategies requires more theoretical and experimental attention. We hope that our open source implementation will ease such future research.

3.5.5 Issue with extend right

As mentioned earlier, our current implementation does not support the `ER` operation very well. In more detail, the issue is that after running a sieve in the context $[\ell : r]$, and applying `ER`, the vectors in the database are padded with a 0 to be defined over the context $[\ell : r + 1]$; geometrically, these vectors remain in the context $[\ell : r]$, and so will all their potential combinations considered by the sieve. While we do sample some fresh vectors, i.e. from the context $[\ell : r + 1]$, to increase the database size after calling `ER`, the fraction of these fresh vectors in the database is rather small

$$\frac{\sqrt{4/3}^{n+1} - \sqrt{4/3}^{n}}{\sqrt{4/3}^{n+1}} = 1 - \sqrt{3/4} \approx 13\% \text{ of database vectors.}$$

This alone slows down the Gauss sieve when used in right progressive sieving compared to left progressive sieving, and this percentage of fresh vectors is even smaller in sieves that use a smaller database; e.g. approximately 12% in a 3-sieve with minimal memory $\left(3\sqrt{3}/4\right)^{n/2}$, see Section 3.6.1.

The situation is even worse in the faster sieves we implement. Indeed, apart from the reference Gauss sieve, our sieves are not guaranteed to maintain a full rank database, that is, if $\{\mathbf{w}_i\}_i$ are the vectors that make up some database in $\Lambda_{[\ell:r]}$, then $\mathrm{span}_{\mathbb{R}}(\{\mathbf{w}_i\}_i) \cong \mathbb{R}^m$ for $m < n$ can occur. This is because, for performance purposes, we relax the replacement condition. In the standard Gauss sieve, $\mathbf{w}_1 \pm \mathbf{w}_2$ may only replace $\mathbf{w}_1$ or $\mathbf{w}_2$ if it is shorter. We relax this and allow $\mathbf{w}_1 \pm \mathbf{w}_2$ to replace the current longest vector $\mathbf{w}_{\text{long}}$ in the database. Fresh vectors are much longer than the recycled ones, therefore they are quickly replaced by combinations of recycled vectors, effectively meaning there is little representation of the newly introduced basis vector after an `ER`.

While we tried to implement countermeasures to avoid losing rank, they had a noticeable impact on performance, and were not robust. For this work, we therefore avoid the use of extend right, as procedures based on extend left already perform well. We leave it as an open problem to develop appropriate variants of fast sieve algorithms that avoid this issue.

3.6 Implementation details

3.6.1 Sieving

We implemented several variants of sieving, namely: a Gauss sieve [MV10c], a relaxation of the Nguyen–Vidick sieve [NV08], a restriction of the Becker–Gama–Joux sieve [BGJ15] and a 3-sieve [BLS16, HK17]. All exploit the SimHash speed up [Cha02, FBB+15, Duc18a].

The first two were mostly implemented for reference and testing purposes, and therefore are not multithreaded. Nevertheless, we fall back to the Gauss sieve in small dimensions for efficiency and robustness; as discussed earlier, the Gauss sieve is immune to loss of rank, which we sometimes experienced with other sieves in small dimensions (say, $n < 50$), even when not using `ER`.

The termination condition for the sieves follows [Duc18a], namely, they stop when we have obtained a given ratio of the expected number of vectors of norm less than $R \cdot \mathrm{gh}(\Lambda_{[\ell:r]})$. The saturation radius is dictated by the asymptotics of the algorithm at hand, namely, R is such that the sieve uses a database of $N = O(R^n)$ vectors. In particular $R = \sqrt{4/3}$ for all implemented sieves, except for the 3-sieve for which one can choose $R^2 \in \left[3\sqrt{3}/4, 4/3\right] \approx [1.299, 1.333]$. Note that $2^{0.2075} \approx \sqrt{4/3}$ and $2^{0.1887} \approx \sqrt{3\sqrt{3}/4}$.

Nguyen–Vidick Sieve (`nv`) and Gauss Sieve (`Gauss`)

The Nguyen–Vidick sieve finds pairs of vectors $(\mathbf{w}_1, \mathbf{w}_2)$ from the database, whose sum or difference gives a shorter vector, i.e. $\|\mathbf{w}_1 \pm \mathbf{w}_2\| < \max\{\|\mathbf{w}\| : \mathbf{w} \in \mathtt{db}\}$. Once such a pair is found, the longest vector from the database gets replaced by $\mathbf{w}_1 \pm \mathbf{w}_2$. The size of the database is a priori fixed to the asymptotic heuristic minimum $2^{0.2075n+o(n)}$ required to find enough such pairs. The running time of the Nguyen–Vidick sieve is quadratic in the database size.

The Gauss sieve algorithm, similar to the Nguyen–Vidick sieve, searches for pairs with a short sum, but the replacement and the order in which we process the database vectors differ. More precisely, the database now is (implicitly) divided into two parts, the so called 'list' part and the 'queue' part. This separation is encoded in the ordering, with the list part being the first τ vectors. Both parts are kept separately sorted. The list part has the property that the shortness of $\mathbf{w}_1 \pm \mathbf{w}_2$ has been checked for all pairs of vectors $(\mathbf{w}_1, \mathbf{w}_2)$ in the list. We then only check pairs $(\mathbf{w}_1, \mathbf{w}_2)$, where $\mathbf{w}_1$ comes from the queue part and $\mathbf{w}_2$ from the list part. As opposed to Nguyen–Vidick sieve, a reduction is found when the longest vector of the pair $(\mathbf{w}_1, \mathbf{w}_2)$ is replaced by the shortest of $\mathbf{w}_1 \pm \mathbf{w}_2$, not the longest in the database. In the case where the list vector $\mathbf{w}_2$ gets replaced, the result of the reduction $\mathbf{w}_1 \pm \mathbf{w}_2$ is put into the 'queue' part and the search is continued with the same 'queue' vector $\mathbf{w}_1$. Otherwise, if the queue vector $\mathbf{w}_1$ was the longest and is replaced, we restart comparing this new $\mathbf{w}_1$ with all list vectors. A vector is moved from the 'queue' to the 'list' part once no reduction with the 'list' vectors can be found. Asymptotically, the running time and the database size for the Gauss sieve is the same as for the Nguyen–Vidick sieve, but it performs better in practice.

Becker–Gama–Joux Sieve (`bgj1`)

The sieve algorithm from [BGJ15] accelerates the Nguyen–Vidick sieve [NV08] from $2^{0.415n+o(n)}$ down to $2^{0.311n+o(n)}$ by using locality sensitive filters, while keeping the memory consumption to the bare minimum for a 2-sieve, namely $2^{0.2075n+o(n)}$.

This optimal complexity is reached using recursive filtering, however we only implemented a variant of this algorithm with a single level of filtration (hence the name `bgj1`). We leave it to future work to implement the full algorithm and determine when the second level of filtration becomes interesting.

We briefly describe our simplified version. The algorithm finds reducing pairs in the database by successively filling buckets according to a filtering rule, and doing

all pairwise tests inside a bucket. Concretely, it chooses a uniform direction $\mathbf{d} \in \mathbb{R}^n$, $\|\mathbf{d}\| = 1$, and puts in the bucket all database vectors taking (up to sign) a small angle with $\mathbf{d}$, namely all $\mathbf{w}$ such that $|\langle \mathbf{w}, \mathbf{d} \rangle| > \alpha \cdot \|\mathbf{w}\|$.

We choose α so that the size of the buckets is about the square root of the size of the database (asymptotically, $\alpha^2 \to 1 - \sqrt{3/4} \approx 0.366^2$).

> Assuming $\mathbf{w} \in S^{n-1}$ then $\mathbf{w}$ satisfying $\langle \mathbf{w}, \mathbf{d} \rangle > \alpha \cdot \|\mathbf{w}\|$ are exactly $\mathbf{w} \in C_n(\theta_\alpha)$ such that $\cos \theta_\alpha = \alpha$ and $\theta_\alpha \in [0, \pi/2]$. Equivalently, $\mathbf{w} \in \mathcal{C}_n(\alpha)$, the spherical caps of height α from [BDGL16]. As [BGJ15] is a 2-sieve we assume asymptotically that the database has size $(4/3)^{n/2}$ and that these points are distributed i.i.d. uniformly over the surface of S^{n-1}. Recalling Section 2.5.2 and since $\mu^{n-1}(S^{n-1}) = 1$, to have buckets of size $(4/3)^{n/4}$, i.e. square root the database size, we want $\mu^{n-1}(\mathcal{C}_n(\alpha)) = (4/3)^{-n/4}$. By appealing to [BDGL16, Lem 2.1], we solve (ignoring polynomial factors) for $\alpha = \sqrt{1 - \sqrt{3/4}}$.

This choice balances the cost of populating the bucket (through testing the filtering condition) and exploring inside the bucket (checking for pairwise reductions). Both cost $O(N) = 2^{0.2075n+o(n)}$; though in practice we found it faster to make the buckets slightly larger, namely around $3.2\sqrt{N}$. Also note that we can apply a SimHash prefiltering before actually computing the inner product $\langle \mathbf{w}, \mathbf{d} \rangle$, but using a larger threshold for the bucketing prefilter than for the reduction prefilter within a bucket.

Following the heuristic arguments from the literature, and in particular the wedge volume formula [BDGL16, Lem 2.2], we conclude that this sieve succeeds after about $(2/\sqrt{3} - 1/3)^{-n/2} \approx 2^{0.142n+o(n)}$ buckets, for a total complexity of $2^{0.349n+o(n)}$.

> In particular we consider two vectors $\mathbf{w}_1, \mathbf{w}_2 \in S^{n-1}$ at some angle less than $\pi/3$, the maximum that allows reduction. When taking buckets defined by $\alpha^2 = 1 - \sqrt{3/4}$ and θ_α as above, the wedge $W^{n-1}(\mathbf{w}_1, \theta_\alpha, \mathbf{w}_2, \theta_\alpha)$ describes the region of S^{n-1} in which a bucket centre may lie such that $\mathbf{w}_1, \mathbf{w}_2$ are both contained in the bucket. Following the notation of [BDGL16] this is equivalent to the wedge $\mathcal{W}_{\mathbf{w}_1,\alpha,\mathbf{w}_2,\alpha}$. Given the heuristic of Definition 2.5.4 we assume $\theta(\mathbf{w}_1, \mathbf{w}_2) \approx \pi/3$. When calculating the measure of

this wedge we therefore have $\mu^{n-1}(\mathcal{W}_{\mathbf{w}_1,\alpha,\mathbf{w}_2,\alpha}) = \mathcal{W}_n(\alpha,\alpha,\pi/3) \in \left(2/\sqrt{3}-1/3\right)^{n/2+o(n)}$ by [BDGL16, Lem. 2.2]. The reciprocal of this value gives the number of bucket centres we expect to require, and the cost of searching in a bucket is $O(N)$.

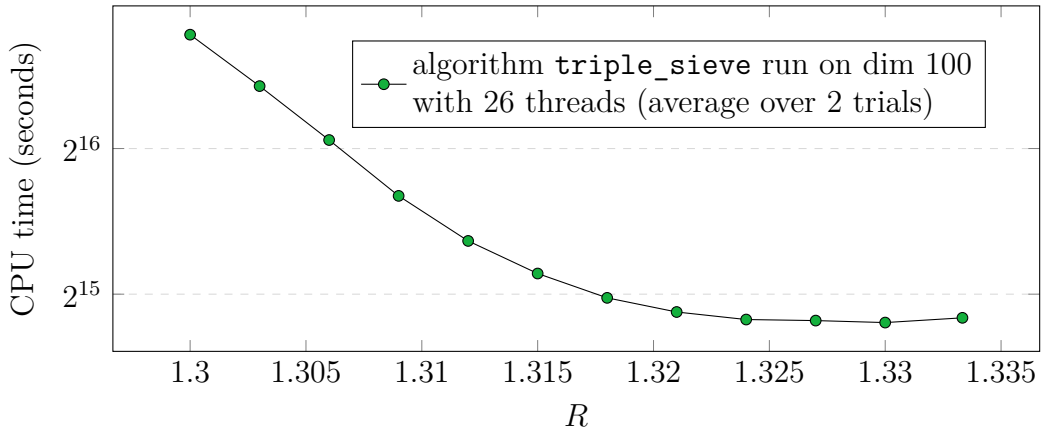
3-sieve (`triple_sieve`)

In its original versions [BLS16, HK17], the 3-sieve algorithm aims to reduce memory consumption at the cost of a potential increase in the running time. The 3-sieve algorithm searches not for pairs, but for *triples* of vectors, whose sum gives a shorter vector (hence the name 3-sieve). Clearly, for a fixed size list of vectors, there are more possible triples than pairs and, therefore, we can perhaps start with a shorter list and still find enough reductions. However, a (naïve) search now costs three iterations over the list. To speed up the naïve search, we can apply filtering techniques similar to the ones used for `bgj1`. In particular, the 3-sieve algorithm with filtering described in [HK17] requires memory $2^{0.1887n+o(n)}$ and runs in time $2^{0.396n+o(n)}$.

For any vector $\mathbf{x}$ from the database, the 3-sieve algorithm of [HK17] filters the database by collecting all vectors $\mathbf{w}$ with a large enough inner product $|\langle\mathbf{x},\mathbf{w}\rangle|$. For all pairs of these collected vectors $(\mathbf{w}_1,\mathbf{w}_2)$, a 3-sieve checks if $\|\mathbf{x}\pm\mathbf{w}_1\pm\mathbf{w}_2\|$ gives a shorter vector. Such an inner product test, as in `bgj1`, helps to identify 'promising' vectors which are likely to result in a length reduction. The only subtlety lies in the fact that in order for a triple to give a reduction, the vectors $\mathbf{x},\mathbf{w}_1,\mathbf{w}_2$ should be far apart, not close to each other as in a 2-sieve. We handle this by adjusting the inner product test and choosing the $\pm$ signs appropriately.

The version of the 3-sieve implemented in G6K splits the database into 'list' and 'queue' parts in the same way as the Gauss sieve above. Further, it combines 2-sieves and 3-sieves. Notice that the filtering process of a 3-sieve is basically the same as bucketing in `bgj1`, with a bucket centre defined by a database[11] vector $\mathbf{x}$. When processing the bucket, we check not only whether a pair $(\mathbf{w}_1,\mathbf{w}_2)$ from the bucket gives a shorter vector, but also whether a triple $(\mathbf{x},\mathbf{w}_1,\mathbf{w}_2)$ may. This additional check has no noticeable impact on performance (we know in which case we potentially are from the signs of the inner products alone), but has the potential to find more shorter vectors.

[11] This relies on the fact that we do not use recursive filtering in `bgj1` – the asymptotically optimal choice from [BGJ15] mandates choosing the buckets centres in a structured way, which is not compatible with choosing them as `db` elements.



The x-axis is the parameter R such that the database size is set to $3.2 \cdot R^{n/2}$. In particular, the rightmost point corresponds to the size of the database being set to $3.2 \cdot (4/3)^{n/2}$; for the leftmost point this value is $3.2 \cdot (3\sqrt{3}/4)^{n/2}$. Raw data embedded.

Figure 3.2 Time memory trade off for our implementation of the 3-sieve algorithm.

As a result, in this combined version of the sieve, we can find more reductions than in a 2-sieve if we keep the same database size as for a 2-sieve. In such a memory regime, most of the reductions will come from 2-reductions. Setting a smaller database makes the algorithm look for more 3-reductions as 2-reductions become less likely.

As `triple_sieve` finds more reductions than `bgj1` with the same database sizes, we may decrease the size of the database and check how the running time degrades. The results of these experiments are shown in Figure 3.2. The leftmost point corresponds to the minimal memory regime for a 3-sieve, namely when the database size is set to $2^{0.1887n+o(n)}$, while the rightmost point is for the `bgj1` memory regime, that is the database size is set to $2^{0.2075n+o(n)}$. It turns out that in moderate dimensions (i.e. 80–110), `triple_sieve` performs slightly better if the database size is a bit less than $2^{0.2075n+o(n)}$. Furthermore, these experiments are consistent with theoretical results on the high memory regime for 3-sieves: in [HKL18] it was proven that the running time of a 3-sieve drops quickly if allowed slightly more memory, as Figure 3.2 shows.

3.6.2 The three layers: `C++ / Cython / Python`

Our implementation consists of three layers.

C++11

The lowest level routines are implemented in C++11. In particular, at this level we define a Siever class which realises G6K for all sieves considered in this work: Gauss,

NV, BGJ1 and 3-sieve. The general design is similar to FPLLL where algorithms are objects operating on matrices and Gram–Schmidt objects. In particular, different sieves are realised as methods on the same object (and thus the same database) allowing the caller to pick which sieve to run in a given situation. For example, in small dimensions it is beneficial to run the Gauss sieve and this design decision allows the database to be reused between different sieves. Our C++ layer does not depend on any third party libraries (except pthreads). On the other hand, our C++ layer is relatively low level.

Cython

Cython is a glue language for interfacing between CPython (the C implementation of the Python programming language) and C/C++. We use Cython for this exact purpose. Our Cython layer is relatively thin, mainly making our C++ objects available to the Python layer and translating to and from FPyLLL data structures [dt21b]. The most notable exception is that we implemented the basis change computation of the *insert* instruction `I`, see (3.2) and (3.3), in Cython instead of C++. The reason being that we call LLL on the lifting context when inserting (the Cython function `split_lll`) which is realised by calling FPyLLL. That is, while our C++ layer has no external dependencies, the Cython layer depends on FPyLLL.

Python

All our high level algorithms are implemented in (C)Python (2). Our code does not use the functional style abstractions from Section 3.4, but a more traditional object oriented approach where methods are called on objects which hold the state. We do provide some syntactic sugar, though, enabling a user to construct new instructions from basic instructions in a function composition style similar to the notation in Section 3.4. Nevertheless, this simplified abstraction is not able to fully exploit all the features of our implementation, and significant savings may be achieved by using the full expressivity of our library.

3.6.3 Vector representation and data structures

The data structures of G6K have been designed for high performance sieving operations and we have tried to minimise memory usage where possible. For high performance we retain the following information about each vector $\mathbf{w} = \mathbf{B}_{[\ell:r]} \cdot \mathbf{v} = \mathbf{B}^{\circ}_{[\ell:\, r]} \cdot \mathbf{v}^{\circ}$ as an *entry* `e` in the sieve database `db`:

- `e.x`: the vector $\mathbf{v}$ as 16-bit integer coordinates for the basis $\mathbf{B}_{[\ell:r]}$;

- `e.yr`: a 32-bit floating point vector to efficiently compute $\langle \mathbf{w}, \mathbf{w}' \rangle$, for some other lattice vector $\mathbf{w}'$. It is a renormalised version of $\mathbf{v}^\circ$;
- `e.cv` (compressed vector): a 256-bit SimHash of $\mathbf{w}$;
- `e.uid` (unique identifier): a 64-bit hash of $\mathbf{v}$;
- `e.len`: the squared length $\|\mathbf{w}\|$ as a 64-bit floating point number.

The entire database `db` is stored contiguously in memory, although unordered. This memory is preallocated and has size equal to the maximum database size within each `Pump`, to avoid additional memory usage caused by reallocations of the database whenever it grows.

To be able to quickly determine whether a potential new vector is already in the database we additionally maintain a C++ `unorderedset` (i.e. a hash table) `uiddb` containing 64-bit hashes `uid` of `e.x` for all entries in `db`.[12] This hash $\texttt{uid} = H(\texttt{x})$ of `x` is simply computed as the inner product of `x` with a global random vector in the ring $\mathbb{Z}/2^{64}\mathbb{Z}$, which has the additional benefit that $H(\texttt{x}_1 \pm \texttt{x}_2)$ can be computed more efficiently as $H(\texttt{x}_1) \pm H(\texttt{x}_2)$. This allows us to cheaply discard collisions without even having to compute $\texttt{x}_1 \pm \texttt{x}_2$.

To maintain a sorted database we utilise a compressed database `cdb` that only stores the 256-bit SimHash, 32-bit floating point length, and the 32-bit `db`-index of each vector. This requires only 40 bytes per vector and everything is also stored contiguously in memory. It is optimised for traversing the database in order of increasing length and applying the SimHash as a prefilter, since accessing the full entry in `db` only occurs a fraction of the time.

For the multithreaded `bgj1`-sieve, the compressed database `cdb` is maintained generally sorted in order of increasing length. Initially `cdb` is sorted, then, during sieving, vectors are replaced one by one starting from the back of `cdb`. It is only resorted when a certain fraction of entries have been replaced. Since we only insert a new vector if its length is below the minimum length of the range of to be replaced vectors in `cdb`, this approach ensures that we always replace the largest vector in `db`. In the sieve variants that split the database into queue and list ranges, we regularly sort the individual ranges. In our multithreaded `triple_sieve`, the vectors removed during a replacement are chosen iteratively from the backs of the two ranges.

Most sieving operations use buckets that are filled based on locality sensitive filters. In `bgj1`, we use the same datastructure as `cdb` for the buckets, and thus copy those

[12] This `unorderedset` is in fact split into many parts to eliminate most blocking locks during a multithreaded sieve.

compressed entries in contiguous memory reserved for that bucket. For `triple_sieve`, we also store information about the actual inner product $\langle \mathbf{x}, \mathbf{w} \rangle$ of the bucket elements $\mathbf{w}$ with the bucket centre $\mathbf{x}$ inside the bucket.

3.6.4 Multithreading

G6K is able to efficiently use multithreading for nearly all operations; a detailed efficiency report can be found in [ADH+19b, App. B]. Global per entry operations such as `EL`, `ER`, `SL` and `I`-postprocessing are simply distributed over all available threads in the global threadpool.

During multithreaded sieving we guarantee that all write operations to entries in `db`, `cdb` and the best lift database are executed in a thread safe manner using atomic operations and write locks. (The actual locking strategies differ per implementation.) We always perform all heavy computations before locking and let each thread locally buffer pending writes and execute these writes in batches to avoid bottlenecks in exclusive access of these global resources.

Threads reading entries in `db` and `cdb` do not use locking and can thus potentially read partially overwritten entries. While this may result in some wasted computation, no faulty vectors will be inserted in the `db`: for every new vector we completely recompute its full entry `e` from `e.x` including its length and verify it is actually shorter than the length of the to be replaced vector before actually replacing it.

Safely resorting `cdb` during sieving is the most complicated operation, since threads do not block on reading `cdb`. Our implementations in G6K resolve this as follows. We let one thread resort `cdb` and use locking to prevent any insertions (or concurrent resorting) by other threads. We keep the old `cdb` untouched as a shadow copy for other threads, while computing a new sorted version that we then atomically publish. Afterwards, other threads will then eventually switch to the newer version. Insertions are always performed using `cdb` and never using a shadow copy, even if e.g. a thread is still using a shadow copy for its main operations, e.g. when building a bucket.

3.7 New lattice reduction records

The experiments reported in this section are based on `bgj1`-sieving, except those on BKZ and LWE which are based on `triple_sieve` in the high memory regime. Here triple sieve uses a database of size $N = \Theta\left((4/3)^{n/2}\right)$, recall Figure 3.2. The switch to `triple_sieve` occurred when improvements made it faster than `bgj1` (especially with

Machine	CPUs	base freq.	cores	threads	HTC*	RAM
L	4xIntel Xeon E7-8860v4	2.2Ghz	72	72	No	512GiB
S	2xIntel Xeon Gold 6138	2.0Ghz	40	80	Yes	256GiB
C	2xIntel Xeon E5-2650v3	2.3Ghz	20	40	Yes	256GiB
A	2xIntel Xeon E5-2690v4	2.6Ghz	28	56	Yes	256GiB

* HTC: Hyperthreading Capable.

Table 3.1 Details of the machines used for experiments.

pump-down sieve, $s = 1$). While it seemed wasteful to rerun all the experiments, we nevertheless now recommend `triple_sieve` over `bgj1` for optimal performance within our library. The details of the machines used for our various experiments are given in Table 3.1.

3.7.1 Exact SVP

We first report on the efficiency of our implementation of `WorkOut` with $s = 0$, $f = 0$, and $f^+ = 1$ when solving exact SVP. The comparison with pruned enumeration is given in Figure 3.3a. While fitted curves are provided, we highlight that they are significantly below asymptotic predictions of $2^{0.349d+o(d)}$ for `bgj1` and thus unreliable for extrapolation.[13] Based on these experiments, we report a crossover with enumeration around dimension 70. Note that we significantly outperform the estimates of a crossover at dimension 90 made in [Duc18a].

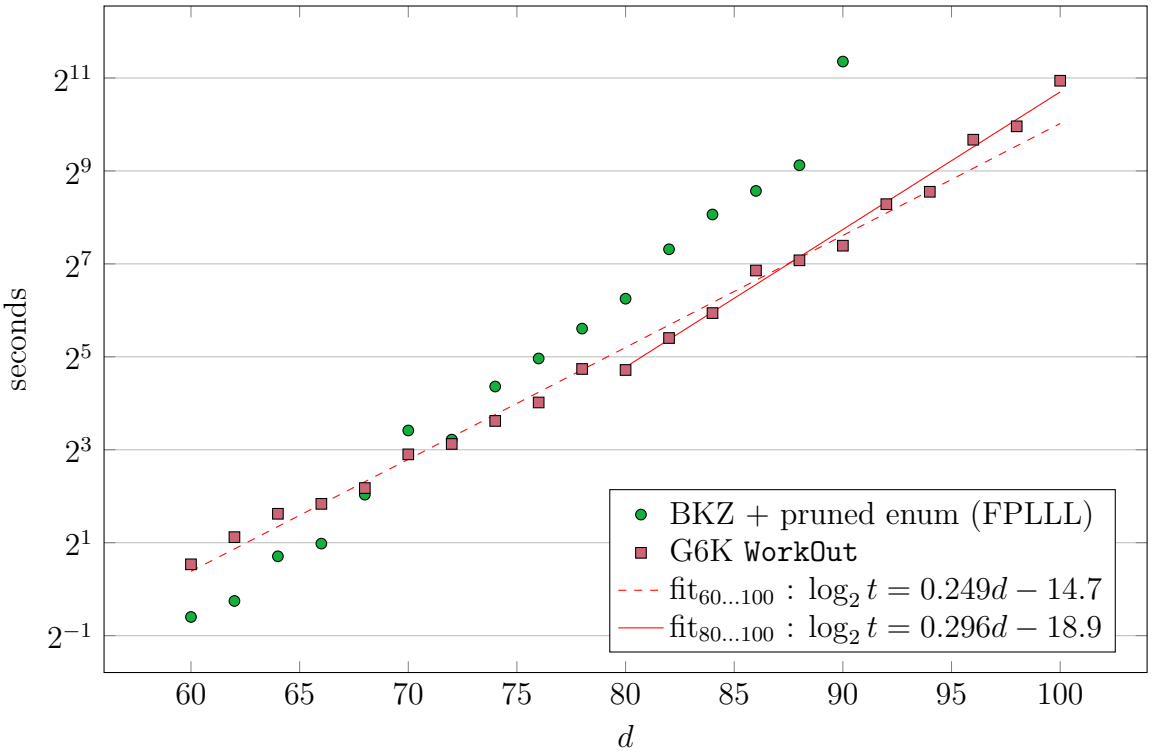
While our improved speed compared to [Duc18a] is mostly due to having implemented a faster sieving algorithm, the new features of G6K also contribute to this improved efficiency (see [ADH+19b, App. A] for a detailed comparison). In particular the on-the-fly lifting strategy offers a few extra dimensions for free as depicted in Figure 3.3b. That is, our new implementation is not only faster but also consumes less memory.

3.7.2 Darmstadt SVP challenges

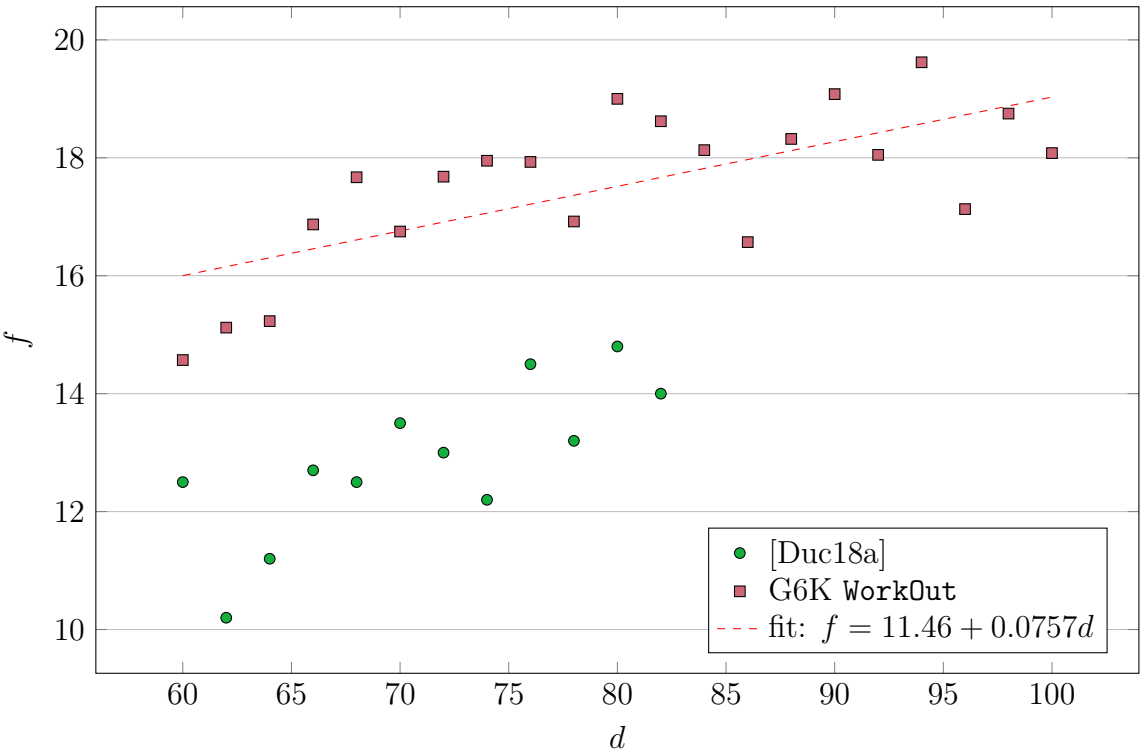
The detailed performance of our implementation when solving Darmstadt SVP Challenges is given in Table 3.2. For some challenges, we also continued the experiments until no shorter vectors were found, hoping to have solved exact SVP on those instances.

[13] This mismatch with theory can be explained by various kinds of overheads, but mostly by the dimensions for free trick: as $f = \Theta(d/\log d)$ is quasilinear, the slope will only very slowly converge to the asymptotic prediction.

(a) Average time in seconds to solve exact SVP.



(b) Average number of dimensions for free when solving exact SVP.



The running time was averaged over 60 trials (12 trials on random bases of 5 different lattices from [GS10]). Each instance was monothreaded, but ran in parallel (20thread/20cores, not hyperthreaded) on machine C. Raw data embedded.

Figure 3.3 Performance for exact SVP.

New solutions to $1.05 \cdot v_d^{-1/d}$-HSVP

SVP dim	Norm	Hermite factor	Sieve max dim	Wall time	Total CPU time	Memory usage	Machine
155	3165	1.00803	127	$14d$ $16h$	$1056d$	†246 GiB	L
153	3192	1.02102	123	$11d$ $15h$	$911d$	†139 GiB	S
151	3233	1.04411	124	$11d$ $19h$	$457.5d$	†160 GiB	C
149	3030	0.98506	117	$60h$ $7m$	$4.66kh$	†59 GiB	S
147	3175	1.03863	118	$123h$ $29m$	$4.79kh$	67.0 GiB	C
145	3175	1.04267	114	$39h$ $3m$	$1496h$	37.7 GiB	C
143	3159	1.04498	110	$17h$ $23m$	$669h$	21.3 GiB	C
141	3138	1.04851	105	$4h$ $59m$	$190h$	10.6 GiB	C
139	3111	1.04303	108	$9h$ $56m$	$380h$	16.2 GiB	C
137	3093	1.04472	107	$9h$ $26m$	$362h$	14.1 GiB	C
136	3090	1.04937	108	$9h$ $16m$	$354h$	16.2 GiB	C
135	3076	1.04968	108	$7h$ $21m$	$277.4h$	16.1 GiB	C
133	3031	1.04133	103	$1h$ $59m$	$71.7h$	8.0 GiB	C
131	2959	1.02362	100	$1h$ $11m$	$41.5h$	5.3 GiB	C
129	2988	1.03813	98	$54m$	$33.2h$	4.2 GiB	C
128	3006	1.04815	102	$2h$ $32m$	$94.9h$	7.6 GiB	C
127	2972	1.04244	101	$2h$ $17m$	$85.0h$	6. GiB	C
126	2980	1.04976	100	$31m$	$19.2h$	5.6 GiB	C
125	2948	1.04393	99	$1h$ $18m$	$47.6h$	5.2 GiB	C
124	2937	1.04032	98	$39m$	$23.9h$	4.4 GiB	C
123	2950	1.04994	93	$7m$	$4.0h$	2.2 GiB	C

New candidate solutions to Exact SVP

SVP dim	Norm	Hermite factor	Sieve max dim	Wall time	Total CPU time	Memory usage	Machine
136	2934	0.99621	112	$18h$ $29m^*$	$704h$	28.5 GiB	C
135	2958	1.00920	108	$6h$ $26m^*$	$244h$	16.2 GiB	C
133	2909	0.99940	103	$2h$ $59m^*$	$112.4h$	12.1 GiB	C
131	2904	1.00465	108	$7h$ $51m^*$	$302.6h$	16.1 GiB	C
129	2875	0.99878	106	$5.2h^*$	$199.3h$	12.0 GiB	C

*: Continued from previously reduced basis for the $1.05 \cdot v_d^{-1/d}$-HSVP solution.
†: Not measured, estimate.

Table 3.2 Performance on the Darmstadt SVP Challenges.

We also compare the running time of our experiments with prior works in Figure 3.1. We warn the reader that the experiments of Table 3.2 are rather heterogeneous – different machines, different software versions, and different parametrisations were used – and therefore discourage extrapolations. Moreover the design decisions below and the probabilistic nature of the algorithm explain the non monotonic time and space requirements.

The parameters were optimised towards speed by trial and error on many smaller instances ($d \approx 100$). More specifically we ran `WorkOut` with parameters $f = \lceil 16+d/12 \rfloor$, $f^+ = 3$, $s = 1$; choosing $f^+ = 1$ or 2 would cost more time and less memory.[14] The loop was set to exit as soon as a vector of the desired length was found, and if it reached the minimal number of dimensions for free, f, it would repeat this largest `Pump` until success (this repetition rarely happened more than three times). The sieve max dim column reports the actual dimension $d - f_{\text{last}}$ of the last `Pump`.

3.7.3 BKZ

To test `PumpNJumpTour` we compare its quality vs. time performance against FPyLLL's implementation of BKZ 2.0 [CN11], and against `NaiveTour` (see Figure 3.4). We generate lattice bases of the form[15] $\mathbf{B}_\mathbf{R}$ from (2.2) for $(n, m, q) = (90, 180, 2^{30})$ and $\mathbf{R} \leftarrow \mathsf{U}(\mathbb{Z}_q^{90\times 90})$. We prereduce the bases using one FPyLLL BKZ tour for each blocksize from 20 to 59 and then report the cumulative time taken by further progressive tours of several BKZ variants. That is, we start timing and perform a tour of BKZ-60, followed by a tour of BKZ-61, and so forth.

Contrary to exact SVP, we find it beneficial for the running time to activate sieving during pump-down for all G6K based BKZ experiments. We further find that `triple_sieve` is noticeably faster than `bgj1`; it seems that the former suffers fewer collisions than the latter when sieving during the pump-down phase.

For all G6K based BKZ experiments we choose the number of dimensions for free following the experimental fit of Figure 3.3b, that is $f = 11.5 + 0.075\beta$. We also introduce a parameter $e = f' - f$ to concretise the more opportunistic `PumpNJumpTour` variant discussed at the end of Section 3.5.3.

To measure quality we use an averaged quality measurement, namely, the slope metric of FPyLLL. This slope, ρ, is a least squares fit of the $\log \|\mathbf{b}_i^*\|^2$. For comparison this metric is preferable to the typical root Hermite factor as it displays much less

[14] The expression $f = \lceil 16 + d/12 \rfloor$ is a local approximation, as we asymptotically expect $f = \Theta(d/\log d)$ for $O(1)$-SVP [Duc18a].

[15] Using FPyLLL's `IntegerMatrix.random(180, 'qary', q=2**30, k=90)`

variance. In the GSA model, the slope ρ relates to the root Hermite factor by $\delta_\beta = \exp(-\rho/4)$.

> In particular, in the GSA model after BKZ-β reduction we have
>
> $$\log\left(\left\|\mathbf{b}_{i+1}^*\right\|^2\right) = \log\left(\delta_\beta^{-4} \cdot \|\mathbf{b}_i^*\|^2\right),$$
>
> and slope
>
> $$\rho = \left(\log\left(\|\mathbf{b}_i^*\|^2\right) - \log\left(\left\|\mathbf{b}_{i+1}^*\right\|^2\right)\right) / \left(i - (i+1)\right).$$
>
> Together these give the relationship above.

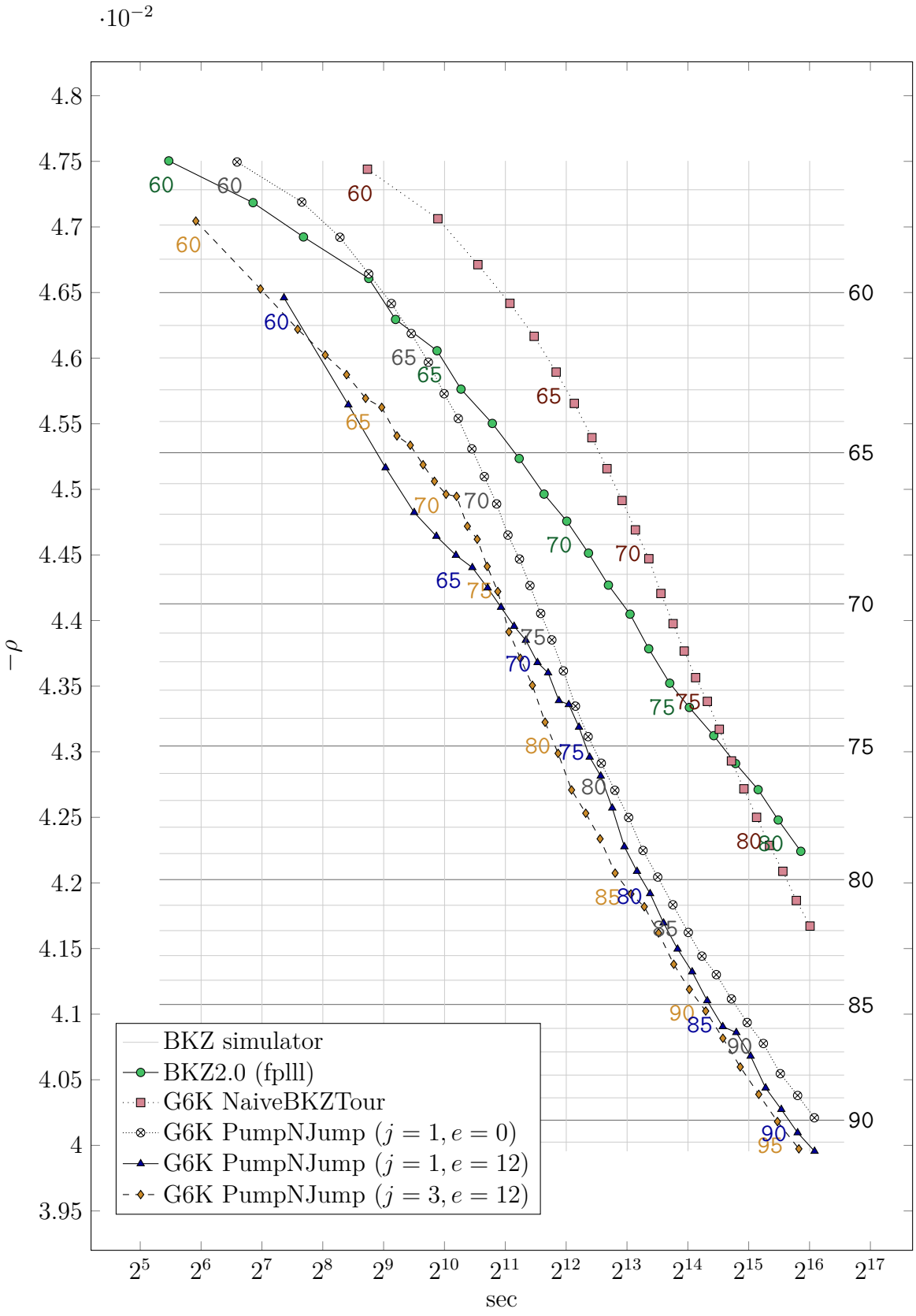
We also provide the predictions for progressive tours given by the BKZ simulator of [CN11, Wal14] as horizontal bars, numbered on the right.

Conclusion

These experiments confirm that it is possible to outperform a naïve application of an SVP-β oracle to obtain a quality equivalent to BKZ-β in less time. Indeed, $\texttt{PumpNJumpTour}_{\beta,f,1}$ is about 4 times faster than $\texttt{NaiveTour}_{\beta,f}$ for the same reduction quality. Furthermore, the opportunistic variant with $e = 12$ gives even better quality per time, and also only requires a smaller β for the same quality, therefore decreasing memory consumption. These experiments also suggest that jumps $j > 1$ are not beneficial as they require similar running time per quality, but with a larger memory consumption.

3.7.4 LWE

The Darmstadt LWE challenges [GY15] are labeled by (n, α), where n denotes the dimension of a secret in $\mathbb{Z}_q$, for some q, and α is a noise rate. See [BBG+16] for full details. Concretely the challenges are given as $(\mathbf{A}, \mathbf{b})$ where $\mathbf{As} + \mathbf{e} \equiv \mathbf{b} \bmod q$ with $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ for $m = n^2$, $\mathbf{s} \in \mathbb{Z}_q^n, \mathbf{e} \in \mathbb{Z}^m$ and $\mathbf{b} \in \mathbb{Z}_q^m$. The value of q is set to the smallest prime number greater than m. Each entry of $\mathbf{e}$, the error, is sampled independently from the discrete Gaussian distribution over the integers with mean $\mu = 0$ and standard deviation $\sigma = \alpha \cdot q$, that is, the distribution over $\mathbb{Z}$ which assigns probability proportional to $\exp\left(-\frac{1}{2} \cdot \frac{|x|^2}{\sigma^2}\right)$ to $x \in \mathbb{Z}$. The entries of $\mathbf{A}$ and $\mathbf{s}$ are sampled independently and uniformly from $\mathbb{Z}_q$. The value m represents the number of 'samples'



The time and slope are averaged over 8 instances for each algorithm. Each instance was monothreaded, but ran in parallel (40threads/40cores, not hyperthreaded) on machine S. We label the point by β for all multiples of 5. Raw data embedded.

Figure 3.4 Performance of BKZ like algorithms.

we have available to us. We will use far fewer than n^2, typically a value between n and $2n$, but still refer to however many samples we do use as m.

Our method for solving LWE is via embedding $\mathbf{e}$ into a uSVP instance [Kan87, BG14b] but using the success condition originally given in [ADPS16] and experimentally justified in [AGVW17]. We also use the embedding coefficient $t = 1$ following [ADPS16, AGVW17]. Concretely we embed the vector $(\mathbf{e}\|1)$ into a dimension $d = m + 1$ embedding basis of the form (2.16). For notational ease we still refer to the embedded vector as $\mathbf{e}$.

We approximate the minimal β such that after BKZ-β reduction, $\|\pi_{d-\beta}(\mathbf{e})\| < \left\|\mathbf{b}^*_{d-\beta}\right\|$. Therefore in the final full block of a BKZ-β tour, $\pi_{d-\beta}(\mathbf{e})$ will be inserted at index $d - \beta$ in the sense of Line 7 of Algorithm 7. The success condition in [ADPS16, AGVW17] using Definition 2.4.6 of the root Hermite factor is

$$\sqrt{\beta} \cdot \sigma < \delta_\beta^{2\beta-d-1} \cdot \mathrm{vol}(\Lambda)^{1/d}. \tag{3.14}$$

It is shown in [AGVW17, Sec. 4.3] that after this insertion, size reduction in the sense of Algorithm 1 on $\mathbf{b}_{d-\beta}$ is enough to recover $\mathbf{e}$ from $\pi_{d-\beta}(\mathbf{b}_{d-\beta}) = \pi_{d-\beta}(\mathbf{e})$ with high probability over the randomness of the basis.

There is no a priori reason why the β used for BKZ reduction and the dimension of the SVP call (currently the last full block in some BKZ-β tour) which first finds a projection of $\mathbf{e}$, should be the same. For enumeration based algorithms it is customary to run one large enumeration after the smaller enumerations inside BKZ, see [LN13]. To apply this to sieving we alter the above inequality to allow a 'decoupling' of these quantities and then balance the expected total time cost.

Let β continue to denote the BKZ block size and η denote the dimension of an SVP call on the lattice $\Lambda_{[d-\eta]}$. We obtain the following success condition

$$\sqrt{\eta} \cdot \sigma < \delta_\eta^{\eta-1} \cdot \delta_\beta^{\eta-d} \cdot \mathrm{vol}(\Lambda)^{1/d}. \tag{3.15}$$

The left hand side is an approximation of the length $\pi_{d-\eta}(\mathbf{e})$ using Definition 2.4.10 and the right hand side an approximation of the Gaussian heuristic of $\Lambda_{[d-\eta]}$. Indeed

$$\begin{aligned} \mathrm{gh}(\Lambda_{[d-\eta]}) &\approx \sqrt{\eta/2\pi e} \cdot (\pi\eta)^{1/2\eta} \cdot \mathrm{vol}(\Lambda_{[d-\eta]})^{1/\eta} \\ &= \sqrt{\eta/2\pi e} \cdot (\pi\eta)^{1/2\eta} \cdot \left(\prod_{i=d-\eta}^{d-1} \|\mathbf{b}_i^*\| \right)^{1/\eta}. \end{aligned} \tag{3.16}$$

(n, α)	Estimated (β, η, d)	Successful (β, ν, ν')	CPU time	Wall time	M.
$(65, 0.010)$	$(108, 137, 244)$	$(112, 124, 120)$	$2553h$	$60h$	A
$(55, 0.015)$	$(106, 135, 219)$	$(110, 125, 103)$	$2198h$	$34h\ 50m$	S
$(40, 0.030)$	$(102, 133, 179)$	$(108, 120, 111)$	$1116h$	$17h\ 43m$	S
$(75, 0.005)$	$(88, 118, 252)$	$(88, 112, 107)^{\ddagger}$	$591h$	$12h\ 26m$	S
$(60, 0.010)$	$(92, 122, 222)$	$(94, 112, 106)^{\dagger}$	$579h$	$11h\ 59m$	S
$(50, 0.015)$	$(87, 118, 194)$	$(81, 111, 95)$	$8h\ 36m$	$1h\ 23m$	S

†: There was also a failed search after $\beta = 90$, with $\nu = 115$.
‡: There was also a failed search after $\beta = 84$, with $\nu = 115$.

Table 3.3 Performance on Darmstadt LWE challenges.

From (2.15) we have $\delta_\eta^{\eta-1} = \sqrt{\eta/2\pi e} \cdot (\eta\pi)^{1/2\eta}$. By combining this with the GSA and the estimate the root Hermite factor gives for $\|\mathbf{b}_0\|$, (3.15) may be derived from (3.16).

> Note that when we are in this decoupling regime we are hoping that sufficient reduction has occurred during BKZ-β reduction such that we obtain $\pi_{d-\eta}(\mathbf{e}) = O_{\mathrm{SVP}}(\Lambda_{[d-\eta]})$. For $\eta \leq \beta$ this is equivalent to $\|\pi_{d-\eta}(\mathbf{e})\| < \left\|\mathbf{b}^*_{d-\eta}\right\|$, since each of these $\mathbf{b}^*_{d-\eta}$ were returned by previous O_{SVP} calls on $\Lambda_{[d-\eta]}$. In the case we are interested in, where $\eta > \beta$, we previously obtained $\mathbf{b}^*_{d-\eta} \leftarrow O_{\mathrm{SVP}}(\Lambda_{[d-\eta:d-\eta+\beta]})$ for $d - \eta + \beta < d$, i.e. a block that does not extend to the end of the basis. This means that an SVP solution $\mathbf{w} \leftarrow O_{\mathrm{SVP}}(\Lambda_{[d-\eta]})$ may be such that $\|\mathbf{w}\| < \left\|\mathbf{b}^*_{d-\eta}\right\|$. Therefore, it is not enough to approximate η for which $\|\pi_{d-\eta}(\mathbf{e})\| < \left\|\mathbf{b}^*_{d-\eta}\right\|$, and instead we approximate where $\|\pi_{d-\eta}(\mathbf{e})\| < \mathrm{gh}(\Lambda_{[d-\eta]})$. This Gaussian heuristic derivation in the $\beta = \eta$ case collapses to (3.14).

Implemented strategy and performance

To solve LWE instances in practice we implemented code which returns triples (β, η, d) that satisfy (3.15), and choose the number of LWE samples accordingly. We then run `PumpNJumpTour` with $s = 1$, $j = 1, e = 12$ and `triple_sieve` as the underlying sieve, and increase β progressively (choosing f as the experimental fit of Figure 3.3b). After

each tour, we measure the wall time T elapsed since the beginning of the reduction, and predict the maximal dimension ν reachable by pumping up within time T. We predict whether we expect to find the projected short vector in this pump (ignoring on-the-fly lifting), following the reasoning of [Duc18a]. That is, we check the inequality

$$\sqrt{\nu} \cdot \sigma \leq \sqrt{4/3} \cdot \mathrm{gh}(\Lambda_{[d-\nu]}). \tag{3.17}$$

If this condition is satisfied, we proceed to search for the LWE solution with a $\texttt{Pump}_{\kappa,f,\beta,s}$ using κ, $f = d - \nu - \kappa$, $\beta = d - \kappa$, and $s = 0$ for $0 \leq \kappa \leq d - \nu$, otherwise we continue BKZ reduction with larger β. This pump ultimately sieves in $\Lambda_{[d-\nu]}$ and lifts database vectors to $\Lambda_{[\kappa]}$.

> Allowing a range of values for κ is a practical cost saving mechanism, as the smaller κ is the more expensive the lifting operation. For $\kappa = 0$ this `Pump` reaches saturation for the projected sublattice $\Lambda_{[d-\nu]}$ and, believing $\pi_{d-\nu}(\mathbf{e})$ to be in this database, lifts database vectors to Λ. In the case where $0 < \kappa \leq d - \nu$ again we reach saturation in $\Lambda_{[d-\nu]}$ but instead lift the database to $\Lambda_{[\kappa]}$. If $\pi_{d-\nu}(\mathbf{e})$ is in the database before lifting and we satisfy $\pi_\kappa(\mathbf{e}) \approx \sqrt{d-\kappa} \cdot \sigma < \mathrm{gh}(\Lambda_{[\kappa]})$, then the projection $\pi_\kappa(\mathbf{e})$ of $\mathbf{e}$ will be inserted into the basis at index κ (rather than just be in the lifted database). We then expect LLL to recover $\mathbf{e}$, and thus solve the LWE instance. In practice we may take κ smaller than the inequality above suggests, say by 5 or so, to try and ensure we do not have to perform another costly pump because of just missing a successful lift of the solution.

If this search is triggered but fails, we go back to reducing the basis with progressive BKZ, and reset the timer T. The search may also succeed before reaching the maximum pump dimension ν, in which case we denote by ν' the dimension at which it stops.

Details of the six new Darmstadt LWE records are in Table 3.3. It should be noted that the CPU time/wall time ratio can be quite far from e.g. 80, the number of threads on machine S. This is because parallelism only kicks in for sieves in large dimensions (see [ADH+19b, App. B]), while the wall times of some of the computations were dominated by BKZ tours with medium blocksizes. One could tailor the parametrisation to improve the wall time further, but this would be in vain as we are mostly interested in the more difficult instances, which suffer little from this issue.

3.8 Conclusion

Since the publication of this work the experimental results of Section 3.7 have been used to inform the concrete security analysis of several submissions to the NIST post quantum standardisation procedure, e.g. [SAB+20, DKR+20]. In particular the experimental evidence for the number of attainable dimensions for free and the required number of vectors to sieve in a given dimension are considered.

The implementation has been incorporated into the open source lattice reduction project fplll,[16] and has been used to run experiments in various papers, e.g. [AH21, ABLR21]. In [AH21] the authors break through the 'lattice barrier' when solving the hidden number problem using lattices. This lattice barrier was a conjecture that said that below a certain number of bits of bias the hidden number problem could not be solved by lattice embedding techniques. Using a new definition of bounded distance decoding with a predicate (some knowledge on the sought after embedded lattice vector that makes it uniquely identifiable) and the ability to tune various parameters such as the saturation ratio of a sieve, the authors use G6K in experiments that show this conjecture is false. In [ABLR21] the authors use G6K to answer in the negative whether using sieving to instantiate an approximate SVP within lattice reduction led to significant gains. The time complexity given for enumeration in the introduction has also been improved in the work of [ABF+20]; the leading constant in the exponent has been improved from $1/(2e) \cdot d$ to $d/8$.

The implementation has also been used as the groundwork for a GPU based implementation.[17] Firstly the authors implement an asymptotically faster sieve [BDGL16] and then define a new filtering condition for close vectors based on the dual lattice. This dual filter is tailored to a GPU architecture [DSvW21]. Given these improvements the authors are able to solve SVP in dimension 180 [GS10], twenty-five dimensions higher than the record achieved in this work.

The future work directions for this chapter are quite open ended. The pump and workout strategies are natural, but they by no means exhaust the possibilities given the available instructions. A functional extend right instruction would allow one to experiment with the sliding window sieve of [LM18]. It would also be useful to understand how the myriad parameters can be better tuned for particular lattice problems. Given the new filter for close vectors in [DSvW21] and the use of a predicate to find a unique vector with properties beyond being short in [AH21], the design space for how one can select vectors to be lifted within G6K seems ripe.

[16] `https://github.com/fplll/g6k`
[17] `https://github.com/WvanWoerden/G6K-GPU-Tensor`

Chapter 4

Quantum Algorithms for the k-List Problem

In this chapter we consider quantum variants of a line of works that introduced what one might call 'low memory' lattice sieves. The memory requirement for a sieve is determined by the size of the list of lattice vectors it needs to maintain to find reducing pairs, or indeed reducing k-tuples in this chapter, and by any memory needed for locality sensitive techniques. The memory complexity of these 'low memory' sieves is still exponential in the dimension of the lattice, but prior to the first of these works [BLS16] it was not known how to reduce the size of the list below $(4/3)^{d/2+o(d)}$. The fundamental idea is to create more opportunities for reductions by looking at combinations of k lattice vectors at once, rather than just pairs. For a list L one has roughly $|L|^k$ k-tuples to consider, and in [BLS16] it is shown that for growing k the number of lattice vectors required to be stored in the list decreases by exponential factors.

Of course, what one gains in memory complexity can just as easily be lost in time complexity, and this is the case in this setting. Let L be a list of the size required for a 2-sieve, one that checks pairs, and L' be a list of the size required for a 3-sieve, one that checks triples. Then the cost to check all pairs in L is approximately $|L|^2$ and the cost of checking all triples in L' is approximately $|L'|^3$. Unfortuntely the L' required by a 3-sieve is such that $|L'| > |L|^{2/3}$, and we therefore have a larger time complexity in the 3-sieve case.

To try and minimise this increase in time complexity, while still obtaining the decreased memory complexity, a filtering condition is introduced in [BLS16] for the 3-sieve case. This filtering condition is an inner product constraint between the first two vectors in a triple, such that pairs of vectors which satisfy this *local* constraint are

more likely to find a third vector with which they form a reducing triple. In particular, under the standard heuristic lattice sieve assumptions, it is shown that one need only search for triples $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ of lattice vectors from the list such that $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle \geq 1/3$. In practice this is achieved by looping over pairs $(\mathbf{x}_1, \mathbf{x}_2)$ and then only searching for a $\mathbf{x}_3$ to create a reducing triple if $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle \geq 1/3$. Since most vectors sampled uniformly from a high dimensional sphere will be close to orthogonal, this filtering condition is often unsatisfied, and therefore avoids a final loop over L' to search for $\mathbf{x}_3$. We call this style of constraint local because it depends on only a pair of vectors, even though the algorithm is in general searching for a k-tuple of vectors that together satisfy some condition, namely that they are reducible.

Using inner product based local constraints within k-sieves is generalised in the work of [HK17]. Gram matrices called *configurations* are introduced, which determine what the inner product should be between any two elements of a k-tuple. This is reminiscent of the filtering approach of [BLS16] except rather than enforcing an inner product constraint on the first pair of elements in a tuple, there are now $O(k^2)$ such inner product constraints. The problem of finding reducing k-tuples is reduced to finding k-tuples that satisfy a well chosen configuration. For the purpose of this introduction, we will call such k-tuples *satisfying* k-tuples. The list size required to expect as many satisfying k-tuples as elements in the input list is derived as a function of the dimension of the lattice and the chosen configuration. By finding the configuration that minimises this list size, one obtains the memory optimal parameters for performing a k-sieve in this configuration framework. In doing so [HK17] are able to match the memory complexity of [BLS16] while significantly improving the time complexity.

The framing of a k-sieve as a configuration problem was studied further in [HKL18]. While [HK17] had asked the question 'what is the memory optimal configuration?', here the authors asked the question 'given any amount of memory (above the minimum implied by the memory optimal case), what is the time optimal configuration for which this is sufficient memory?'. Smooth time memory trade offs were therefore given in [HKL18]; pick a maximum allowable amount of memory, then check which combination of k and configuration gives the minimum time complexity for this amount of memory.

The works above approach the finding of k-tuples that satisfy the various constraints in different manners. Brute force search through lists is used in [BLS16, HK17], whereas [HKL18] also considers locality sensitive data structures for this task. In this chapter we replace brute force search with tailored quantum search algorithms built from nested Grover search and amplitude amplification. In particular we first produce

a quantum variant of the [BLS16] algorithm, and then give a hybrid version of it and [HKL18] which also makes use of quantum enumeration. To finish we switch gears by specialising to the $k = 3$ case and reducing the configuration problem to a graph theoretic problem, namely finding triangles. There are several quantum algorithms for this task, and we investigate their performance relative to the quantum variants of [BLS16] and [HKL18].

Contributions

The author of this thesis contributed equally with the other authors to Sections 4.2 and 4.3, equally with Elena Kirshanova to Section 4.4, equally with Elena Kirshanova and Subhayan Roy Moulik to Section 4.5, and equally with Elena Kirshanova and Erik Mårtensson to Section 4.6.

Acknowledgements

We are grateful to the organisers of the Oxford Post Quantum Cryptography Workshop, held at the Mathematical Institute, University of Oxford, March 18–22, 2019. This work began at the Quantum Cryptanalysis session of this workshop. We would like to acknowledge the fruitful discussions we had with Gottfried Herold during this session.

4.1 Quantum algorithms for the approximate k-List problem and their application to lattice sieving

The work presented here is an amended and annotated version of what is published as

Kirshanova E., Mårtensson E., Postlethwaite E.W., Moulik S.R. (2019) Quantum Algorithms for the Approximate k-List Problem and Their Application to Lattice Sieving.

In: Galbraith S., Moriai S. (eds) Advances in Cryptology – ASIACRYPT 2019. ASIACRYPT 2019. Lecture Notes in Computer Science, vol 11921. Springer, Cham.
https://doi.org/10.1007/978-3-030-34578-5_19

The omitted portions of this work are available at https://eprint.iacr.org/2019/1016.

4.2 Introduction

The Shortest Vector Problem (SVP) is one of the central problems in the theory of lattices. For a given d dimensional Euclidean lattice, usually described by a basis, to solve SVP one must find a shortest non zero vector in the lattice. This problem gives rise to a variety of efficient, versatile, and (believed to be) quantum resistant cryptographic constructions [AD97, Reg05]. To obtain an estimate for the security of these constructions it is important to understand the complexities of the fastest known algorithms for SVP.

There are two main families of algorithms for SVP, (1) algorithms that require $2^{\omega(d)}$ time and poly(d) memory; and (2) algorithms that require $2^{\Theta(d)}$ time and memory. The first family includes lattice enumeration algorithms [Kan83, GNR10]. The second contains sieving algorithms [AKS01, NV08, MV10c], Voronoi cell based approaches [MV10b] and others [BGJ14, ADRS15]. In practice, it is only enumeration and sieving algorithms that are currently competitive in large dimensions [ADH$^+$19a, TKH18]. Practical variants of these algorithms rely on *heuristic* assumptions. For example we may not have a guarantee that the returned vector will solve SVP exactly (e.g. pruning techniques for enumeration [GNR10], lifting techniques for sieving [Duc18a]), or that our algorithm will work as expected on arbitrary lattices (e.g. sieving algorithms may fail on orthogonal lattices). Yet these heuristics are natural

for lattices often used in cryptographic constructions, and one does not require an exact solution to SVP to progress with cryptanalysis [ADH+19a]. Therefore, one usually relies on heuristic variants of SVP solvers for security estimates.

Among the various attractive features of lattice based cryptography is its potential resistance to attacks by quantum computers. In particular, there is no known quantum algorithm that solves SVP on an arbitrary lattice significantly faster than existing classical algorithms.[1] However, some quantum speedups for SVP algorithms are possible in general.

It was shown by Aono–Nguyen–Shen [ANS18] that enumeration algorithms for SVP can be sped up using the *quantum backtracking* algorithm of Montanaro [Mon18]. More precisely, with quantum enumeration one solves SVP on a d dimensional lattice in time $d^{\frac{1}{4e}d+o(d)}$, a square root improvement over classical enumeration. This algorithm requires $\mathrm{poly}(d)$ classical and quantum memory. This bound holds for both provable and heuristic versions of enumeration. Quantum speedups for sieving algorithms are considered by Laarhoven–Mosca–van de Pol [LMv13] and later by Laarhoven [Laa16]. The latter study presents various quantum sieving algorithms for SVP. One of them achieves time and classical memory complexity of order $2^{0.2653d+o(d)}$ and requires $\mathrm{poly}(d)$ quantum memory. This is the best known quantum time complexity for heuristic sieving algorithms. Provable single exponential SVP solvers are considered in the quantum setting by Chen–Chang–Lai [CCL18]. Based on [DRSD14, ADRS15], the authors describe a $2^{1.255d+o(d)}$ time, $2^{0.5d+o(d)}$ classical and $\mathrm{poly}(d)$ quantum memory algorithm for SVP. All heuristic and provable results rely on the classical memory being quantumly addressable, i.e. on qRAM.[2]

A drawback of sieving algorithms is their large memory requirements. Initiated by Bai–Laarhoven–Stehlé, a line of work [BLS16, HK17, HKL18] gives a family of heuristic sieving algorithms, called tuple lattice sieves, or k-sieves for some fixed constant k, that offer time memory trade offs. Such trade offs have proven important in the current fastest SVP solvers, as the ideas of tuple sieving offer significant speedups in practice, [ADH+19a]. In this work, we explore various directions for *asymptotic* quantum accelerations of tuple sieves.

[1] For some families of lattices, e.g. ideal lattices, there exist quantum algorithms that solve an SVP variant faster than classical algorithms [CDW17, PHS19]. We consider arbitrary lattices.

[2] A recent work [CL21] achieves time $2^{0.2570d+o(d)}$ but requires exponential rather than polynomial quantum memory, and for the qRAM operation to be applied to this quantum memory.

Our results

1. In Section 4.5 we show how to use a quantum computer to speed up the k-sieve of Bai–Laarhoven–Stehlé [BLS16] and its improvement due to Herold–Kirshanova–Laarhoven [HKL18] (Algorithms 8 and 9). One data point achieves a time complexity of $2^{0.2989d+o(d)}$, while requiring $2^{0.1395d+o(d)}$ classical memory and poly(d) width quantum circuits. Compared to the algorithm of [Laa16] with time and memory complexities $2^{0.2653d+o(d)}$ we almost halve the constant in the exponent for memory at the cost of a small increase in the respective constant for time.

2. In Section 4.6 we reduce the 3-sieve case to the graph theoretic problem of finding triangles. We show that using the triangle finding algorithm of [BdWD$^+$01] gives an algorithm that matches the complexity of the memory optimal version of Algorithm 9. We then go on to use the triangle finding algorithm of [GN17], the complexity of which depends on the *sparsity* of the graph we embed the 3-sieve problem into. By altering the sparsity of the graph we are able to perform a 3-sieve using $2^{0.3264d+o(d)}$ *queries*.[3] We do not convert this query complexity into time complexity, see the second open question below.

In [KMPM19b] we further perform the following analyses.

1. In Appendix B, borrowing ideas from [Laa16] we give a quantum k-sieve that also exploits nearest neighbour techniques, namely locality sensitive filtering (LSF). For $k = 2$, we recover Laarhoven's $2^{0.2653d+o(d)}$ time and memory quantum algorithm.

2. In Section 7 we describe a quantum circuit consisting only of gates from a universal gate set (e.g. CNOT and single qubit rotations) of depth $2^{0.1038d+o(d)}$ and width $2^{0.2075d+o(d)}$ that implements the 2-sieve as proposed classically in [NV08]. In particular we consider exponential *quantum* memory to make significant improvements to the number of time steps. Our construction adapts the parallel search procedure of [BBG$^+$13].

Our main results, quantum time memory trade offs for sieving algorithms, are summarised in Figure 4.1. When optimising for time a quantum 2-sieve with LSF remains the best algorithm. For $k \geq 3$ the speedups offered by LSF are less impressive,

[3] This means that the complexity of the algorithm is measured in the number of oracle calls to the adjacency matrix of the graph.

and one can achieve approximately the same asymptotic time complexity by considering quantum k-sieve algorithms (without LSF) with $k \geq 10$ and far less memory.

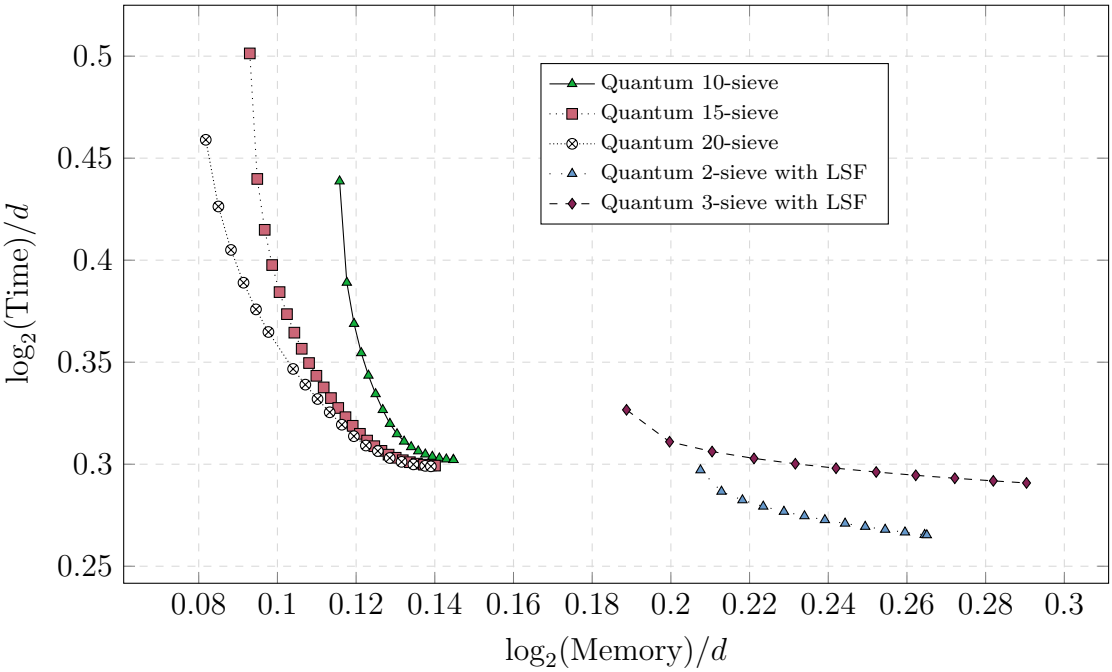


Figure 4.1 Time memory trade offs for Algorithm 8 with $k \in \{10, 15, 20\}$ and [KMPM19b, Alg B.2] with $k \in \{2, 3\}$. Each curve provides time memory trade offs for a fixed k, either with nearest neighbour techniques (the right two curves) or without (the left three curves). Each point on a curve (x, y) represents (Memory, Time) values, obtained by numerically optimising for time while fixing available memory. For example, we build the leftmost curve (dotted brown) by computing the memory optimal (Memory, Time) value for the 20-sieve and then repeatedly increase the available memory (decreasing time) until we reach the time optimal (Memory, Time) value. Increasing memory further than the rightmost point on each curve does not decrease time. The figures were obtained using an optimisation package provided by Maple™ [Map].

All the results presented in this work are asymptotic in nature: our algorithms have time, classical memory, and quantum memory complexities of order $2^{\mathsf{c}d+o(d)}$, $2^{\mathsf{c}'d+o(d)}$, and $\mathrm{poly}(d)$ respectively, for $\mathsf{c}, \mathsf{c}' \in \Theta(1)$, which we aim to minimise. We do not attempt to specify the $o(d)$ or $\mathrm{poly}(d)$ terms.

Our techniques

We now briefly describe the main ingredients of our results.

1. A useful abstraction of the k-sieve is the *configuration problem*, first described in [HK17]. It consists of finding k elements that satisfy certain pairwise inner product constraints from k exponentially large lists of vectors. Assuming $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ is a solution tuple, the i^{th} element $\mathbf{x}_i$ can be obtained via a brute force search either over the i^{th} input list [BLS16], see Figure 4.2a, or over a certain sublist of the i^{th} list [HK17, HKL18], see Figure 4.2b. We replace the brute force searches with calls to Grover's algorithm and reanalyse the configuration problem.

2. The configuration problem can be reduced to the k-clique problem in a graph with vertices representing elements from the lists given by the configuration problem. Vertices are connected by an edge if and only if the corresponding list elements satisfy some inner product constraint. We specialise to the $k = 3$ case and apply the quantum triangle finding algorithms of Buhrman et al. [BdWD+01] and Le Gall–Nakajima [GN17]. For this latter algorithm we exploit its dependence on sparsity by forming many graphs from unions of subsets of vertices of the original graph, which allows us to control the sparsity of these newly formed graphs.

Open questions

1. The classical configuration search algorithms of [HKL18] offer time memory trade offs for SVP by varying k (larger k requires less memory but more time). We observe in Section 4.4 that time optimal classical algorithms for the configuration problem hit a certain point on the time memory trade off curve once k becomes large enough, see Table 4.1. The same behaviour is observed in our quantum algorithms for the configuration problem, see Table 4.2. Although we provide some explanation of this, we do not rigorously prove that the trade off curve indeed stops on its time optimal side. We leave it as an open problem to determine the shape of the configuration problem for these time optimal instances of the algorithm. Another open problem originating from [HK17] is to extend the analysis to non constant k.

2. We do not give time complexities for our approach based on [GN17] in Section 4.6, instead reporting the query complexity. We leave open the question of determining e.g. the complexity of forming auxiliary databases used by the quantum random walks on Johnson graphs of [GN17], as well as giving the (quantum) memory requirements of these methods in our setting. To determine the number of

repetitions of the algorithm we also assume it outputs a uniform triangle from the graph, the proof (or disproof) of which we leave open.

4.3 Preliminaries

Recall we define $S^{d-1} = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| = 1\} \subset \mathbb{R}^d$ and $[n] = \{1, \ldots, n\}$. We use soft O notation to denote running times, in particular we let any function $f(d) \in 2^{\mathsf{c}d+o(d)}$ be such that $f(d) \in \widetilde{O}(2^{\mathsf{c}d})$, i.e. the notation suppresses subexponential factors.[4]

For some matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ we denote its $(i,j)^{\text{th}}$ entry as $\mathbf{A}_{i,j}$. For any $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{R}^d$ the *Gram matrix* $\mathbf{C} \in \mathbb{R}^{k\times k}$ is given by $\mathbf{C}_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$, the set of pairwise inner products. For $I \subset [k]$, we denote by $\mathbf{C}[I]$ the $|I| \times |I|$ submatrix of $\mathbf{C}$ obtained by restricting $\mathbf{C}$ to the rows and columns indexed by I. For a vector $\mathbf{x}$ and $i \in [d]$, we denote by $\mathbf{x}[i]$ the i^{th} entry of $\mathbf{x}$.

Lattices

Recall the introduction to lattices given in Section 2.2. In this work we deal solely with full rank lattices of rank and dimension d.

Quantum search

Recall the introduction given to Grover's algorithm and amplitude amplification given in Section 2.6.2. In particular recall the unitaries $\mathbf{D}, \mathbf{R}_0, \mathbf{R}_f$, the Grover and amplitude amplification iterations $\mathbf{G}(f)$ and $\mathbf{G}(\mathbf{A}, f)$, and the results of Theorem 2.6.1 and Theorem 2.6.2. The results of this paper rely on applying both Grover's algorithm and amplitude amplification.

Computational models

Our algorithms are analysed in the quantum circuit model [NC11, KLM07]. Each wire represents a qubit, i.e. a vector in a two dimensional complex Hilbert space, and we assert that we have a set of universal gates. We work in the noiseless quantum theory model, i.e. we assume there is no (or negligible) decoherence or other sources of noise in the computational procedures.

The algorithms given in Sections 4.5 and 4.6 are in the qRAM model and assume quantumly accessible classical memory [GLM08]. More concretely in this model we

[4]Note that this is not *exactly* the same as the usual meaning where $f(d) \in \widetilde{O}(g(d)) \Rightarrow$ there exists a $k \in \mathbb{N}$ such that $f(d) \in O(g(d) \cdot \log^k(g(d)))$

store all data, e.g. the list of vectors, in classical memory and only demand that this memory is quantumly accessible, i.e. that elements in the list can be efficiently accessed in coherent superposition. Concretely, we require the qRAM operation described in Section 2.6.1. This enables us to design algorithms that in principle do not require large quantum memories that can be implemented with only poly(d) qubits and with the $2^{\Theta(d)}$ sized lists stored in classical memory. Several works [BHT97, Kup13] suggest that this memory model is potentially easier to achieve than a full quantum memory.

In Section 4.6 we study the algorithms in the query model, which is the typical model for quantum triangle finding algorithms. Namely, the complexity of our algorithm is measured in the number of oracle calls to the adjacency matrix of a graph associated to a list of vectors.

4.4 Sieving as configuration search

In this section we describe previously known *classical* heuristic sieving algorithms. We will not go into detail or give proofs, which can be found in the relevant references.

Sieving algorithms receive a basis $\mathbf{B} \in \mathbb{R}^{d\times d}$ as input and start by sampling an exponentially large list L of (long) lattice vectors from $\Lambda(\mathbf{B})$. There are efficient algorithms for sampling lattice vectors, e.g. [Kle00]. The elements of L are then iteratively combined to form shorter lattice vectors, $\mathbf{x}_{\text{new}} = \mathbf{x}_1 \pm \mathbf{x}_2 \pm \ldots \pm \mathbf{x}_k$ such that $\|\mathbf{x}_{\text{new}}\| \leq \max_{i\leq k}\{\|\mathbf{x}_i\|\}$, for some $k \geq 2$. Newly obtained vectors $\mathbf{x}_{\text{new}}$ are stored in a new list and the process is repeated with this new list of shorter vectors. Namely, we consider Nguyen–Vidick style sieves in this work. It can be shown [NV08, Reg09a] that after poly(d) such iterations we obtain a list that contains a short vector. Therefore, the asymptotic complexity of sieving is determined by the cost of finding k-tuples whose combination produces shorter vectors. Under certain heuristics, specified below, finding such k-tuples can be formulated as the approximate k-List problem.

Definition 4.4.1 (Approximate k-List problem)**.** We are given k lists $L_1, \ldots, L_k \subset \mathbb{R}^d$ of equal exponential (in d) size $|L|$ and whose elements are i.i.d. uniform on S^{d-1}. The approximate k-List problem is to find $|L|$ solutions, where a solution is a k-tuple $(x_1, \ldots, x_k) \in L_1 \times \cdots \times L_k$ satisfying $\|\mathbf{x}_1 + \cdots + \mathbf{x}_k\| \leq 1$.

The assumption made in analyses of heuristic sieving algorithms [NV08] is that the lattice vectors in the new list after an iteration are thought of as i.i.d. uniform vectors on a thin spherical shell (essentially, a sphere), and, once normalised, on S^{d-1}. Hence sieves do not 'see' the discrete structure of the lattice from the vectors operated on.

The heuristic becomes invalid when the vectors become short. In this case we assume we have solved SVP. Thus, we may not find a *shortest* vector, but an approximation to it, which is enough for many cryptanalytic purposes.

We consider k to be constant. The lists $L_1, \dots, L_k$ in Definition 4.4.1 may be identical. Note that the approximate k-List problem only looks for solutions with $+$ signs, i.e. $\|\mathbf{x}_1 + \cdots + \mathbf{x}_k\| \leq 1$, while sieving looks for arbitrary signs. This is not an issue, as we may repeat an algorithm for the approximate k-List problem $2^k \in O(1)$ times in order to obtain all solutions.

Configuration search

Using a concentration result on the distribution of inner products of $\mathbf{x}_1, \dots, \mathbf{x}_k \in S^{d-1}$ shown in [HK17], the approximate k-List problem can be reduced to the configuration problem. In order to state this problem, we introduce configurations.

Definition 4.4.2 (Configuration)**.** The configuration $\mathbf{C} = \mathrm{Conf}(\mathbf{x}_1, \dots, \mathbf{x}_k)$ of k points $\mathbf{x}_1, \dots, \mathbf{x}_k \in S^{d-1}$ is the Gram matrix of the $\mathbf{x}_i$, i.e. $\mathbf{C}_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$.

A configuration $\mathbf{C} \in \mathbb{R}^{k \times k}$ is a symmetric positive semidefinite matrix. Note that the diagonal of $\mathbf{C}$ is formed of all ones. Rewriting the solution condition $\|\mathbf{x}_1 + \cdots + \mathbf{x}_k\|^2 \leq 1$, one can check that a configuration $\mathbf{C}$ for a solution tuple satisfies $\mathbb{1}^{\mathrm{t}}\mathbf{C}\mathbb{1} \leq 1$. We denote the set of such 'good' configurations by

$$\mathscr{C} = \{\mathbf{C} \in \mathbb{R}^{k \times k} \colon \mathbf{C} \text{ is symmetric positive semidefinite and } \mathbb{1}^{\mathrm{t}}\mathbf{C}\mathbb{1} \leq 1\}.$$

It has been shown [HK17] that rather than looking for k-tuples that form a solution for the approximate k-List problem, we may look for k-tuples that satisfy a constraint on their configuration. This gives rise to the following problem.

Definition 4.4.3 (Configuration problem)**.** Let $k \in \mathbb{N}$ and $\varepsilon > 0$. Suppose we are given a target configuration $\mathbf{C} \in \mathscr{C}$. Given k lists $L_1, \dots, L_k \subset \mathbb{R}^d$ all of exponential (in d) size $|L|$, whose elements are i.i.d. uniform from S^{d-1}, the configuration problem consists of finding a $1 - o(1)$ fraction of all solutions, where a solution is a k-tuple $(\mathbf{x}_1, \dots, \mathbf{x}_k)$ with $\mathbf{x}_i \in L_i$ such that $|\langle \mathbf{x}_i, \mathbf{x}_j \rangle - \mathbf{C}_{i,j}| \leq \varepsilon$ for all i, j.

Solving the configuration problem for a $\mathbf{C} \in \mathscr{C}$ gives solutions to the approximate k-List problem. For a given $\mathbf{C} \in \mathbb{R}^{k \times k}$ the number of expected solutions to the configuration problem is a function of $\det(\mathbf{C})$ as the following theorem shows.

Theorem 4.4.1 (Distribution of configurations [HK17, Theorem 1]). *If $\mathbf{x}_1, \ldots, \mathbf{x}_k$ are i.i.d. from S^{d-1} and $d > k$, then their configuration $\mathbf{C} = \mathrm{Conf}(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ follows a distribution with density function*

$$\mu = W_{d,k} \cdot \det(\mathbf{C})^{\frac{1}{2}(d-k)} \mathrm{d}\mathbf{C}_{1,2} \ldots \mathrm{d}\mathbf{C}_{d-1,d}, \tag{4.1}$$

where $W_{d,k} \in O_k(d^{\frac{1}{4}(k^2-k)})$ is an explicitly known normalisation constant that only depends on d and k. Here O_k denotes that the asymptotic statement is true for fixed k.

This theorem tells us that the expected number of solutions to the configuration problem for $\mathbf{C}$ is given by $\prod_i (|L_i|) \cdot (\det \mathbf{C})^{d/2}$. If we want to apply an algorithm for the configuration problem to the approximate k-List problem (and to sieving), we require that the expected number of output solutions to the configuration problem is equal to the size of the input lists. Namely, $\mathbf{C}$ and the input lists L_i of size $|L|$ should (up to polynomial factors) satisfy $|L|^k \cdot (\det \mathbf{C})^{d/2} = |L|$. This condition gives a lower bound on the size of the input lists. Using Chernoff bounds, one can show (see [HKL18, Lemma 2]) that increasing this bound by a poly(d) factor gives a sufficient condition for the size of input lists. In particular

$$|L| \in \widetilde{O}\left(\left(\frac{1}{\det(\mathbf{C})}\right)^{\frac{d}{2(k-1)}}\right) \tag{4.2}$$

is sufficient.

Classical algorithms for the configuration problem

The first classical algorithm for the configuration problem which chose $k \geq 2$ was given by Bai–Laarhoven–Stehlé [BLS16] (Figure 4.2a). It was later improved by Herold–Kirshanova [HK17] and by Herold–Kirshanova–Laarhoven [HKL18] (Figure 4.2b). These results present a family of algorithms for the configuration problem that offer time memory trade offs. In Section 4.5 we present quantum versions of these algorithms.

Both algorithms [BLS16, HKL18] process the lists from left to right but in a different manner. For each $\mathbf{x}_1 \in L_1$ the algorithm from [BLS16] applies a filtering procedure to L_2 and creates the 'filtered' list $L_2(\mathbf{x}_1)$. This filtering procedure takes as input an element $\mathbf{x}_2 \in L_2$ and adds it to $L_2(\mathbf{x}_1)$ if and only if $|\langle \mathbf{x}_1, \mathbf{x}_2 \rangle - \mathbf{C}_{1,2}| \leq \varepsilon$. Throughout, vectors in brackets indicate fixed elements with respect to which the list has been filtered. Having constructed the list $L_2(\mathbf{x}_1)$, the algorithm then iterates over it: for each $\mathbf{x}_2 \in L_2(\mathbf{x}_1)$ it applies the filtering procedure to L_3 with respect to

$\mathbf{C}_{2,3}$ and obtains $L_3(\mathbf{x}_1, \mathbf{x}_2)$. Filtering of the original lists $(L_1, \ldots, L_k)$ continues in this fashion until we have constructed $L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1})$ for fixed values $\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}$. Amongst the tuples of the form $(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}, \mathbf{x}_k)$ for $\mathbf{x}_k \in L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1})$ will be solutions to the configuration problem. The algorithms from [HK17, HKL18] apply more filtering steps. For a fixed $\mathbf{x}_1 \in L_1$, they not only create $L_2(\mathbf{x}_1)$, but also $L_3(\mathbf{x}_1), \ldots, L_k(\mathbf{x}_1)$. This speeds up the iteration over $\mathbf{x}_2 \in L_2(\mathbf{x}_1)$, where now the filtering step with respect to $\mathbf{C}_{2,3}$ is applied not to L_3, but to $L_3(\mathbf{x}_1)$, as well as to $L_4(\mathbf{x}_1), \ldots, L_k(\mathbf{x}_1)$. Each $L_i(\mathbf{x}_1)$ is smaller than L_i. This speeds up the construction of $L_3(\mathbf{x}_1, \mathbf{x}_2), \ldots, L_k(\mathbf{x}_1, \mathbf{x}_2)$. The algorithm continues with this filtering process until the last inner product check with respect to $\mathbf{C}_{k-1,k}$ and some $\mathbf{x}_{k-1} \in L_{k-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})$ is applied to all the elements from $L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})$ and the list $L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1})$ is constructed. This gives configuration problem solutions of the form $(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}, \mathbf{x}_k)$ for all $\mathbf{x}_k \in L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1})$. Pseudocode for the [HK17, HKL18] approach can be found in [KMPM19b, App A].

The fundamental difference between the BLS and HKL algorithms is that the first never constructs the lists $L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)$ for $i < j - 1$. As such the conditions $|\langle \mathbf{x}_i, \mathbf{x}_j \rangle - \mathbf{C}_{i,j}| \leq \varepsilon$ for $i < j - 1$ are not checked, see Figure 4.2a and set e.g. $(i, j) = (1, 3)$. Provided the input lists are sufficiently large, in the sense of Theorem 4.4.1 and below, then enough solutions to the configuration problem will be found by the BLS algorithm in the form of tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}, \mathbf{x}_k)$. Depending on the configuration there may also be many non solutions of this form. On the other hand, the HKL algorithm checks every condition demanded by the configuration problem, and as such any tuple $(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}, \mathbf{x}_k)$ will be a solution.

In this work we give two quantum algorithms, a quantum variant of BLS and of a quantum variant of HKL. The quantum BLS variant (Algorithm 8) is *like* the classical BLS algorithm in the sense that it never explicitly constructs lists $L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)$ for $i < j - 1$. It is *unlike* the classical BLS algorithm in that it does check all the filtering conditions required by a configuration $\mathbf{C}$, see Definition 4.5.1 of the function encoded by Grover iterations. The quantum HKL variant (Algorithm 9) is *like* the classical HKL algorithm in that it explicitly creates some filtered lists of the form $L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)$ for $i < j - 1$, and

checks all the filtering conditions required by a configuration $\mathbf{C}$. It is *unlike* the classical HKL algorithm in that it does not necessarily explicitly create all such filtered lists.

Important for our analyses throughout the paper is the result of [HKL18] that describes the sizes of all the intermediate lists that appear during the configuration search algorithms via the determinants of submatrices of the target configuration $\mathbf{C}$. The next theorem gives the expected sizes of these lists and the time complexity of the algorithm from [HKL18].

Theorem 4.4.2 (Intermediate list sizes [HKL18, Lemma 1] and time complexity)**.** *During a run of the configuration search algorithm described in Figure 4.2b, given an input configuration* $\mathbf{C} \in \mathbb{R}^{k\times k}$ *and lists* $L_1, \ldots, L_k \subset S^{d-1}$ *each of size* $|L|$*, the intermediate lists for* $1 \leq i < j \leq k$ *are of expected sizes*

$$\mathbb{E}[|L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)|] = |L| \cdot \left(\frac{\det(\mathbf{C}[1, \ldots, i, j])}{\det(\mathbf{C}[1, \ldots, i])}\right)^{d/2}. \tag{4.3}$$

The expected running time of the algorithm described in Figure 4.2b is

$$T^{\mathbf{C}}_{\text{k-Conf}} = \max_{1\leq i\leq k}\left[\prod_{r=1}^{i} |L_r(\mathbf{x}_1, \ldots, \mathbf{x}_{r-1})| \cdot \max_{i+1\leq j\leq k} |L_j(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})|\right]. \tag{4.4}$$

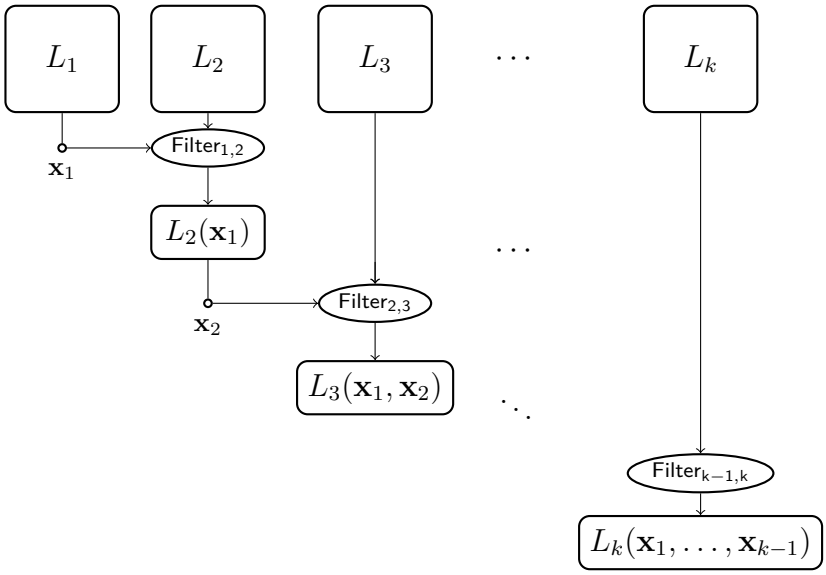
The quantum variants of both the BLS and HKL algorithms check all filtering conditions and output tuples that are solutions for an input configuration $\mathbf{C}$. They therefore both follow the expected intermediate list sizes of (4.3). They should also both be compared to the performance of the classical HKL algorithm (4.4) for which time optimal values are given in Table 4.1.

Finding a configuration for optimal runtime

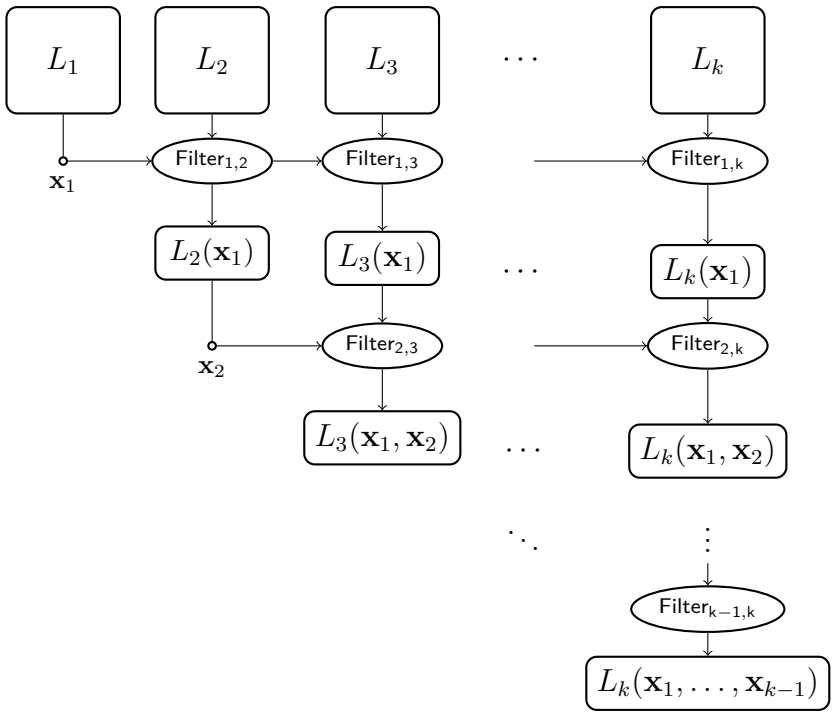
For a given i the square bracketed term in (4.4) represents the expected time required to create all possible filtered lists on a given 'level'.

Definition 4.4.4. Given some $k \geq 1$, a list $L \subset S^{d-1}$, vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{k-1} \in \mathbb{R}^d$ we say $L(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})$ is *on* or *at* level i. In particular, L is at level 1 and $L(\mathbf{x}_1, \ldots, \mathbf{x}_{k-1})$ is at level k.

Figure 4.2 Algorithms for the configuration problem. Procedures $\mathsf{Filter}_{i,j}$ receive as input a vector (e.g. $\mathbf{x}_1$), a list of vectors (e.g. L_2), and a real number $\mathbf{C}_{i,j}$. It creates another shorter list (e.g. $L_2(\mathbf{x}_1)$) that contains all vectors from the input list whose inner product with the input vector is within some small ε from $\mathbf{C}_{i,j}$.



(a) The algorithm of Bai et al. [BLS16] for the configuration problem.



(b) The algorithm of Herold et al. [HKL18] for the configuration problem.

k	2	3	4	5	6	...	16	17	18
Time	0.4150	0.3789	0.3702	0.3707	0.3716		0.3728	0.37281	0.37281
Space	0.2075	0.1895	0.1851	0.1853	0.1858		0.1864	0.18640	0.18640

Table 4.1 Asymptotic complexity exponents base 2 for the approximate k-List problem. The table gives optimised runtime, and the corresponding memory, exponents for the classical algorithm from [HKL18], see Figure 4.2b and [KMPM19b, App. A].

In particular one may think of a row of lists in Figure 4.2b.

> A given i in the outer max of (4.4) represents the cost of forming all possible filtered lists on level $i+1$, e.g. $i=1$ is
>
> $$|L_1| \cdot \max\{|L_j| : j \in \{2,\dots,k\}\}.$$
>
> This is the cost (up to a constant factor) of forming $L_2(\mathbf{x}_1),\dots,L_k(\mathbf{x}_1)$ for all $\mathbf{x}_1 \in L_1$.
> The final value $i=k$ in the outer max gives the expected number of total solutions tuples $(\mathbf{x}_1,\dots,\mathbf{x}_{k-1},\mathbf{x}_k)$ and therefore represents the cost of reading the output.

In order to find an optimal configuration $\mathbf{C}$ that minimises (4.4), we perform numerical optimisations using the Maple™ package [Map].[5] In particular, we search for $\mathbf{C} \in \mathscr{C}$ that minimises (4.4) under the condition that (4.2) is satisfied (so that we actually obtain enough solutions for the k-List problem). Figures for the optimal runtime and the corresponding memory are given in Table 4.1. The memory is determined by the size of the input lists computed from the optimal $\mathbf{C}$ using (4.2). Since the k-List routine determines the asymptotic cost of a k-sieve, the figures in Table 4.1 are also the respective constants for the complexities of k-sieves.

Interestingly, the optimal runtime constant turns out to be equal for large enough k. This can be explained as follows. The optimal $\mathbf{C}$ achieves the situation where all the expressions in the outer max of (4.4) are equal. This implies that creating all the filtered lists on level i asymptotically costs the same as creating all the filtered lists on level $i+1$ for $2 \le i \le k-1$. The cost of creating filtered lists $L_i(\mathbf{x}_1)$ on the second level is of order $|L|^2$. This value $|L|^2$ becomes (up to $\mathrm{poly}(d)$ factors) the running time of the whole algorithm (compare the Time and Space constants for $k = 16, 17, 18$

[5] The code is available at `https://github.com/ElenaKirshanova/QuantumSieve`.

in Table 4.1). The precise shape of $\mathbf{C} \in \mathscr{C}$ that makes the costs per level equal can be obtained by equating all the terms in the outer max of (4.4) and minimising the value $|L|^2$ under these constraints. Even for small k these computations become rather tedious and we do not attempt to express $\mathbf{C}_{i,j}$ as a function of k, which is, in principal, possible.

Finding a configuration for optimal memory

If we want to optimise for memory, the optimal configuration $\mathbf{C}$ has all its off diagonal elements $\mathbf{C}_{i,j} = -1/k$. Such a configuration is called *balanced*. It is shown in [HK17] that such $\mathbf{C}$ maximises $\det(\mathbf{C})$ among all $\mathbf{C} \in \mathscr{C}$, which, in turn, minimises the sizes of the input lists, see (4.2). However, such configurations do not lead to optimal running times since the costs per level are not equal.

4.5 Quantum configuration search

In this section we present several quantum algorithms for the configuration problem (Definition 4.4.3). As explained in Section 4.4, these directly translate to quantum sieving algorithms for SVP. We start with a quantum version of the BLS style configuration search [BLS16], then we show how to improve this algorithm by constructing intermediate filtered lists.

Recall that in the configuration problem we receive k lists $L_1, \ldots, L_k \subset S^{d-1}$, a configuration matrix $\mathbf{C} \in \mathbb{R}^{k \times k}$, and $\varepsilon > 0$ as input. We assume for simplicity each of $L_1, \ldots, L_k$ has size a power of two, namely $|L_i| = 2^t$. This is not necessary, but it allows us notational simplicity by creating uniform superpositions using only Hadamard gates. To describe our algorithms we define $f_{[i],j}$, a function that takes d dimensional vectors as input.

Definition 4.5.1. Let $k \geq 2$ and $1 \leq i < j \leq k$. Given some $L_1, \ldots, L_k \subset S^{d-1}$, and for some implicit and hardcoded $\mathbf{x}_1 \in L_1$, configuration $\mathbf{C} \in \mathbb{R}^{k \times k}$, and $\varepsilon > 0$, define

$$f_{[i],j} \colon L_2 \times \cdots \times L_i \times L_j \to \{0,1\}$$
$$(\mathbf{x}_2, \ldots, \mathbf{x}_i, \mathbf{x}_j) \mapsto \begin{cases} 0 & |\langle \mathbf{x}_\ell, \mathbf{x}_j \rangle - \mathbf{C}_{\ell,j}| \leq \varepsilon \ \text{ for all } \ell \in [i], \\ 1 & \text{else.} \end{cases}$$

Using $f_{[i],j}$ we perform a check for 'good' elements and may construct the lists $L_j(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i)$ from $L_j(\mathbf{x}_1, \ldots, \mathbf{x}_\ell)$ for any $0 \leq \ell \leq i$. We assume that any vector

encountered by the algorithm, e.g. an element of $L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)$, can be stored as $\bar{d}$ qubits. We denote by $|\mathbf{0}\rangle$ the $\bar{d}$-tensor of 0 qubits, i.e. $|\mathbf{0}\rangle = |0^{\otimes \bar{d}}\rangle$. Recall that $\mathbf{R}_{f_{[i],j}}$ denotes the part of the Grover iteration $\mathbf{G}(f_{[i],j})$ that negates a non root of $f_{[i],j}$, see Section 2.6.2. In particular we have

$$\mathbf{R}_{f_{[i],j}} |\mathbf{x}_2\rangle \cdots |\mathbf{x}_i\rangle |\mathbf{x}_j\rangle = |\mathbf{x}_2\rangle \cdots |\mathbf{x}_i\rangle (-1)^{f_{[i],j}(\mathbf{x}_2,\ldots,\mathbf{x}_i,\mathbf{x}_j)} |\mathbf{x}_j\rangle .$$

We are searching for elements of $L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)$ so we apply the rest of the Grover iteration, the $\mathbf{D}\mathbf{R}_0\mathbf{D}^\dagger$, to only the final register above. Our Grover iteration therefore is of the form

$$\mathbf{G}(f_{[i],j}) = (\mathbf{I}_{\bar{d}}^{\otimes(i-1)} \otimes \mathbf{D}) \cdot (\mathbf{I}_{\bar{d}}^{\otimes(i-1)} \otimes \mathbf{R}_0) \cdot (\mathbf{I}_{\bar{d}}^{\otimes(i-1)} \otimes \mathbf{D}^\dagger) \cdot \mathbf{R}_{f_{[i],j}}.$$

The level 1 lists $L_1, \ldots, L_k$ are stored classically and are assumed to be quantumly accessible. In particular, we assume that we can efficiently construct a uniform superposition over all elements from a given list L by first applying Hadamards followed by a qRAM table lookup. That is, we assume an efficient circuit for

$$|0^{\otimes t}\rangle |\mathbf{0}\rangle \to \frac{1}{\sqrt{|L|}} \sum_i |i\rangle |\mathbf{0}\rangle \to \frac{1}{\sqrt{|L|}} \sum_i |i\rangle |L[i]\rangle . \tag{4.5}$$

Here the first register is used to index elements of L and the second register to store them. The first arrow denotes the unitary $\mathbf{H}^{\otimes t}$ that puts the indices into equal superposition, and the second arrow denotes a call to qRAM as introduced in Section 2.6.1. We call the unitary that performs the above $\mathbf{D}$, and for simplicity we ignore the first register that stores the indices. Note that often it is the second register which is ignored, and the indices remembered. This is how quantum search was introduced in Section 2.6.2, where the domain of the function f is the indices of elements in a list, and where f is a function that checks a property of the elements of this list relating to these indices. For notational and expository reasons in this chapter we prefer to work directly with the list elements.

We denote by $|\Psi_L\rangle$ a uniform superposition over the elements of L, i.e. $|\Psi_L\rangle = \frac{1}{\sqrt{|L|}} \sum_{\mathbf{x}\in L} |\mathbf{x}\rangle$. More generally we define the following notation.

Definition 4.5.2. Given any (possibly filtered) list $L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)$ we denote

$$|\Psi_{L_j(\mathbf{x}_1,\ldots,\mathbf{x}_i)}\rangle = \frac{1}{\sqrt{|L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)|}} \sum_{\mathbf{x}\in L_j(\mathbf{x}_1,\ldots,\mathbf{x}_i)} |\mathbf{x}\rangle.$$

The idea of our algorithm for the configuration problem is the following. We have a global *classical* loop over $\mathbf{x}_1 \in L_1$ inside which we run our quantum algorithm to find a $(k-1)$ tuple $(\mathbf{x}_2, \ldots, \mathbf{x}_k)$ that together with $\mathbf{x}_1$ gives a solution to the configuration problem. We expect to have $O(1)$ such $(k-1)$ tuples per $\mathbf{x}_1$.[6] At the end of the algorithm we expect to obtain such a solution by means of amplitude amplification, recall Theorem 2.6.2. In Theorem 4.5.1 we argue that this procedure succeeds in finding a solution with probability in $1 - 2^{-\Omega(d)}$.

Inside the classical loop over $\mathbf{x}_1$ we prepare $(k-1)\bar{d}$ qubits, which we arrange into $k-1$ registers, so that each register will store (a superposition of) input vectors, see Figure 4.3. Each such register is set in uniform superposition over the elements of the input lists: $|\Psi_{L_2}\rangle \otimes |\Psi_{L_3}\rangle \otimes \cdots \otimes |\Psi_{L_k}\rangle$. We apply Grover's algorithm on $|\Psi_{L_2}\rangle$ using $\mathbf{G}(f_{[1],2})$. We have $|L_2(\mathbf{x}_1)|$ 'good' states out of $|L_2|$ states in $|\Psi_{L_2}\rangle$, so by performing $O\left(\sqrt{\frac{|L_2|}{|L_2(\mathbf{x}_1)|}}\right)$ applications of $\mathbf{G}(f_{[1],2})$ we obtain

$$|\Psi_{L_2(\mathbf{x}_1)}\rangle = \frac{1}{\sqrt{|L_2(\mathbf{x}_1)|}} \sum_{\mathbf{x}_2 \in L_2(\mathbf{x}_1)} |\mathbf{x}_2\rangle \,. \tag{4.6}$$

In fact, what we obtain is a state close to (4.6) as there will be some probability of measuring an $\mathbf{x}_2 \notin L_2(\mathbf{x}_1)$. We do not now measure this state, and instead keep it in superposition. For now we drop the expression 'close to' for all the states in this description, and argue about the failure probability in Theorem 4.5.1.

Now consider the state $|\Psi_{L_2(\mathbf{x}_1)}\rangle |\Psi_{L_3}\rangle$ and the function $f_{[2],3}$ that uses the first and second registers as inputs, with a hardcoded $\mathbf{x}_1$. We apply the unitary $\mathbf{G}(f_{[2],3})$ to $|\Psi_{L_2(\mathbf{x}_1)}\rangle |\Psi_{L_3}\rangle$. In other words, for all vectors from L_3, we check if they satisfy the inner product constraints with respect to $\mathbf{x}_1$ and $\mathbf{x}_2$. There are $|L_3(\mathbf{x}_1, \mathbf{x}_2)|$ 'good' states in $|\Psi_{L_3}\rangle$. We therefore perform $O\left(\sqrt{\frac{|L_3|}{|L_3(\mathbf{x}_1,\mathbf{x}_2)|}}\right)$ applications of $\mathbf{G}(f_{[2],3})$ to obtain the state

$$|\Psi_{L_2(\mathbf{x}_1)}\rangle |\Psi_{L_3(\mathbf{x}_1,\mathbf{x}_2)}\rangle = \frac{1}{\sqrt{|L_2(\mathbf{x}_1)|}} \sum_{\mathbf{x}_2 \in L_2(\mathbf{x}_1)} |\mathbf{x}_2\rangle \left(\frac{1}{\sqrt{|L_3(\mathbf{x}_1, \mathbf{x}_2)|}} \sum_{\mathbf{x}_3 \in L_3(\mathbf{x}_1,\mathbf{x}_2)} |\mathbf{x}_3\rangle \right).$$

We continue creating the lists $L_{i+1}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i)$ by filtering the *level 1* list L_{i+1} with respect to $\mathbf{x}_1$ (fixed by the outer classical loop), and with respect to $\mathbf{x}_2, \ldots, \mathbf{x}_i$ (in superposition) using $\mathbf{G}(f_{[i],i+1})$. At level $k-1$ we obtain the state $|\Psi_{L_2(\mathbf{x}_1)}\rangle \otimes |\Psi_{L_3(\mathbf{x}_1,\mathbf{x}_2)}\rangle \otimes \cdots \otimes |\Psi_{L_{k-1}(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle$. For the last list L_k we filter with respect to $\mathbf{x}_1, \ldots, \mathbf{x}_{k-2}$, as for the list L_{k-1}, rather than $\mathbf{x}_1, \ldots, \mathbf{x}_{k-1}$. Finally, for a fixed $\mathbf{x}_1$, the

[6] This follows by multiplying the sizes of the lists $L_i(\mathbf{x}_1, \ldots \mathbf{x}_{i-1})$ for all $2 \leq i \leq k$.

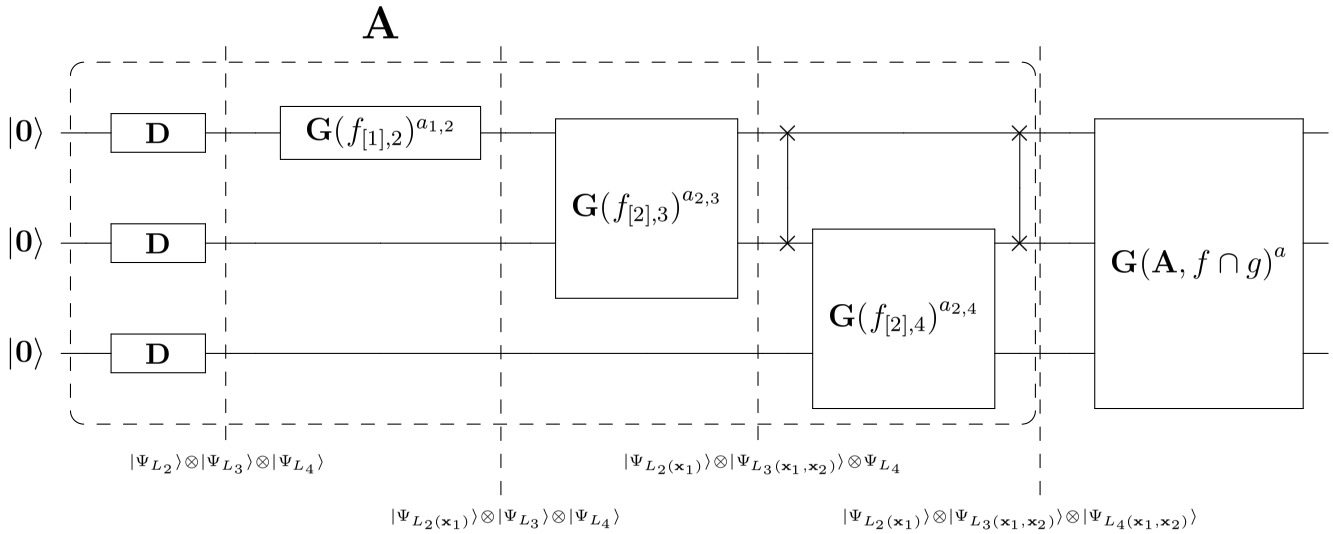


Figure 4.3 A quantum circuit representing the quantum part of Algorithm 8 with $k = 4$, i.e. this circuit is executed inside a classical loop over $\mathbf{x}_1 \in L_1$. The unitary $\mathbf{D}^{\otimes 3}$ creates $|\Psi_{L_2}\rangle \otimes |\Psi_{L_3}\rangle \otimes |\Psi_{L_4}\rangle$. The expressions below the vertical dashed lines represent the (idealised) state of the system after each gate is applied. The various Grover iterations are iterated $a_{i,j} = O\Big(|L_j| \cdot (|L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)|)^{-1}\Big)^{1/2}$ times. We use a swap gate to swap the states of the first and second registers before applying $\mathbf{G}(f_{[2],4})$ to ensure it has the correct inputs. We then swap these states back to their original registers. The composition of the unitaries in the dashed box is $\mathbf{A}$ which is repeated $a = \sqrt{|L_2(\mathbf{x}_1)| \cdot |L_3(\mathbf{x}_1, \mathbf{x}_2)| \cdot |L_4(\mathbf{x}_1, \mathbf{x}_2)|}$ times for amplitude amplification. The final measurement returns a triple $(\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$ which, together with a fixed $\mathbf{x}_1$, forms a solution to the configuration problem with high probability.

'filtered' state we obtain is of the form

$$|\Psi_F\rangle = |\Psi_{L_2(\mathbf{x}_1)}\rangle \otimes |\Psi_{L_3(\mathbf{x}_1,\mathbf{x}_2)}\rangle \otimes \cdots \otimes |\Psi_{L_{k-1}(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle \otimes |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle\,. \tag{4.7}$$

The state is expected to contain $O(1)$ many $(k-1)$ tuples $(\mathbf{x}_2,\ldots,\mathbf{x}_k)$ which together with $\mathbf{x}_1$ give a solution to the configuration problem. To prepare the state $|\Psi_F\rangle$ for a fixed $\mathbf{x}_1$, we need

$$T_{\text{InGrover}} \in O\left(\sqrt{\left(\frac{|L_2|}{|L_2(\mathbf{x}_1)|}\right)} + \cdots + \sqrt{\left(\frac{|L_k|}{|L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})|}\right)}\right) \tag{4.8}$$

unitary operations of the form $\mathbf{G}(f_{[i],j})$. This is what we call the 'inner' Grover procedure.

Recall amplitude amplification, as introduced in Section 2.6.2. Let us denote by $\mathbf{A}$ an algorithm that creates $|\Psi_F\rangle$ from $|\mathbf{0}\rangle \otimes \cdots \otimes |\mathbf{0}\rangle$ in time T_{InGrover} without performing any measurements. In order to obtain a solution tuple $(\mathbf{x}_2,\ldots,\mathbf{x}_k)$ we apply amplitude amplification using the unitary $\mathbf{G}(\mathbf{A}, f \cap g)$.

Here f is a function where a tuple $(\mathbf{x}_2,\ldots,\mathbf{x}_k)$ is a root of f, i.e. $f(\mathbf{x}_2,\ldots,\mathbf{x}_k) = 0$, if and only if it a root of each of $f_{[1],2},\ldots,f_{[k-2],k-1}, f_{[k-2],k}$. That is, it satisfies the filtering conditions that produce $|\Psi_F\rangle$. A tuple $(\mathbf{x}_2,\ldots,\mathbf{x}_k)$ is a root of $f \cap g$ if and only if it is a root of f and a root of g. The function g operates on the final two elements of the tuple, as defined below.

Definition 4.5.3. Let $k \geq 3$. Given some $L_{k-1}, L_k \subset S^{d-1}$, and for some implicit and hardcoded configuration $\mathbf{C} \in \mathbb{R}^{k\times k}$, and $\varepsilon > 0$, define

$$\begin{aligned} g\colon L_{k-1} \times L_k &\to \{0,1\} \\ (\mathbf{x}_{k-1},\mathbf{x}_k) &\mapsto \begin{cases} 0 & |\langle \mathbf{x}_{k-1},\mathbf{x}_k\rangle - \mathbf{C}_{k-1,k}| \leq \varepsilon \\ 1 & \text{else.} \end{cases} \end{aligned} \tag{4.9}$$

Therefore, a tuple satisfying $(f \cap g)(\mathbf{x}_2,\ldots,\mathbf{x}_k) = 0$ forms the solution $(\mathbf{x}_1,\ldots,\mathbf{x}_k)$ to the configuration problem with the fixed $\mathbf{x}_1$. Indeed, in the state $|\Psi_F\rangle$ it is only the last two registers storing $\mathbf{x}_{k-1}$ and $\mathbf{x}_k$ that are left to be checked against the target configuration, which is precisely the purpose of g. Let $|\mathbf{z}\rangle = |\mathbf{x}'_2 \cdots \mathbf{x}'_k\rangle$ be any state that gives a solution of the form $(\mathbf{x}_1,\mathbf{x}'_2,\ldots,\mathbf{x}'_k)$. The state $|\mathbf{z}\rangle$ appears in $|\Psi_F\rangle$ with

amplitude

$$\langle \mathbf{z}|\Psi_F\rangle = \left(|L_2(\mathbf{x}_1)|\cdots|L_{k-1}(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})|\cdot|L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})|\right)^{-1/2}. \tag{4.10}$$

> Note that since we are currently assuming that the Grover iterations $\mathbf{G}(f_{[i],j})$ give us a uniform superposition over only the 'good' states, we can calculate this quantity as
>
> $$\begin{aligned}\langle \mathbf{z}|\Psi_F\rangle &= n_2 \sum_{\mathbf{x}_2\in L_2(\mathbf{x}_1)} \langle \mathbf{x}_2' \,|\, \mathbf{x}_2\rangle \otimes \cdots \otimes n_k \sum_{\mathbf{x}_k\in L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})} \langle \mathbf{x}_k' \,|\, \mathbf{x}_k\rangle \\ &= n_2 \cdot \cdots \cdot n_k,\end{aligned}$$
>
> where $n_i = |L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|^{-1/2}$ for $i \in \{2,\ldots,k-1\}$, and $n_k = |L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})|^{-1/2}$. In practice it will be a little smaller, as dealt with in Theorem 4.5.1.

This value is the inverse of the number of applications of $\mathbf{G}(\mathbf{A}, f\cap g)$ we repeat in order to maximise our probability of obtaining $\mathbf{z}$ when measuring $|\Psi_F\rangle$. The overall complexity of the algorithm for the configuration problem becomes

$$\begin{aligned}T_{\text{BLS}}^{\text{Q}} \in O\Bigg(|L_1|\Bigg(&\sqrt{\left(\frac{|L_2|}{|L_2(\mathbf{x}_1)|}\right)} + \cdots + \sqrt{\left(\frac{|L_k|}{|L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})|}\right)}\Bigg) \\ &\cdot\sqrt{|L_2(\mathbf{x}_1)|\cdot|L_3(\mathbf{x}_1,\mathbf{x}_2)|\cdots|L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})|}\Bigg).\end{aligned} \tag{4.11}$$

The filtered lists in the above expression are assumed to be of expected size greater than or equal to 1. There exist configurations for which intermediate lists have expected size less than 1 (see (4.3)), which should be understood as the reciprocal of the expected number of times we need to construct these lists to obtain 1 element in them. For such a configuration there therefore exist states in the superposition for which a solution does not exist. For states for which a solution does exist (we expect $O(1)$ of these per $\mathbf{x}_1$), we perform $O(\sqrt{|L|})$ Grover iterations during the 'inner' Grover procedure, and during the 'outer' procedure these 'good' elements contribute a $O(1)$ factor to the running time. For such configurations each $|L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|$ in (4.11) and Theorem 4.5.1 (below) should be changed to $\max\{1, |L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|\}$. Alternatively, one can enforce that intermediate lists are of size greater than 1 during the optimisation. We choose this option, and therefore disregard this subtlety for the rest of this work.

Algorithm 8 Quantum algorithm for the Configuration Problem.

Input: $L_1, \ldots, L_k \subset S^{d-1}$, target configuration $\mathbf{C} \in \mathbb{R}^{k \times k}$, $\varepsilon > 0$.
Output: L_{out}, list of k-tuples satisfying configuration $\mathbf{C}$.

1: $L_{\text{out}} \leftarrow \emptyset$
2: **for all** $\mathbf{x}_1 \in L_1$ **do**
3: Prepare the state $|\Psi_{L_2}\rangle \otimes \cdots \otimes |\Psi_{L_k}\rangle$
4: **for all** $i = 2 \ldots k-1$ **do**
5: Apply $\mathbf{G}(f_{[i-1],i})^{a_i}$ for some a_i to obtain $|\Psi_{L_2(\mathbf{x}_1)}\rangle \cdots |\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})}\rangle$.
6: Apply $\mathbf{G}(f_{[k-2],k})^{a_k}$ for some a_k to obtain $|\Psi_{L_2(\mathbf{x}_1)}\rangle \cdots |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle$.
7: Let $\mathbf{A}$ be unitary that implements lines 3–6, i.e.

$$\mathbf{A}\,|\mathbf{0}^{\otimes(k-1)}\rangle = |\Psi_F\rangle\,.$$

8: Apply $\mathbf{G}(\mathbf{A}, f \cap g)^a$ to $|\Psi_F\rangle$ for some a, where g is defined in (4.9).
9: Measure all the registers to obtain a tuple $(\mathbf{x}_2, \ldots, \mathbf{x}_k)$.
10: **if** $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ satisfies $\mathbf{C}$ **then**
11: $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{(\mathbf{x}_1, \ldots, \mathbf{x}_k)\}$.

4.5.1 Intermezzo: on using expected list sizes

Throughout this work we use the expected size of an intermediate list given by (4.3) to determine the correct number of Grover iterations to apply in a given quantum search procedure. This requires some justification, as applying even a constant factor more Grover iterations than the optimal number can, in some cases, form a quantum superposition that is almost an equal superposition (see the example of [BBHT98, Sec. 2]); you have done too much work and are back to where you started! To justify our use of the expected sizes of given intermediate lists we appeal to [HKL18, Lem. 2] and a careful analysis of Grover's algorithm following [BBHT98].

To begin we examine [HKL18, Lem. 2] and recall the two sided Chernoff bound for a binomial distribution $X \sim \text{Bin}(N, p)$. Let $\mu = \mathbb{E}[X]$ and $\delta \in (0, 1)$, then

$$\Pr[X \notin [1-\delta, 1+\delta] \cdot \mu] \leq \exp\left(-\frac{\delta^2}{2+\delta} \cdot \mu\right) + \exp\left(-\frac{\delta^2}{2} \cdot \mu\right) \leq 2 \cdot \exp\left(-\frac{\delta^2}{3} \cdot \mu\right).$$

The lemma argues that any sublist defined by some configuration $\mathbf{C}$, e.g. $L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i) \subset L$, is formed of i.i.d. uniformly sampled elements $\mathbf{x} \leftarrow \mathsf{U}(S^{d-1})$ that satisfy some constant number, less than or equal to k, of approximate inner product constraints. The size of $L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)$ is therefore a binomial distribution with $N = |L| \in 2^{cd+o(d)}$ and p some function of the configuration $\mathbf{C}$ and $\mathbf{x}_1, \ldots, \mathbf{x}_i$. In our applications, $\mathbf{x}_1, \ldots, \mathbf{x}_i$ will satisfy the subconfiguration $\mathbf{C}[1, \ldots, i]$ so p is a function of just $\mathbf{C}$.

We now consider δ in the above Chernoff bound as a function of the dimension d of our lattice. By the concentration result of Theorem 4.4.1, p will be exponentially small in d. As we avoid configurations which imply intermediate lists have expected size less than 1, we focus on the case when $\mu = Np \in 2^{\alpha d + o(d)}$ for $\alpha > 0$. Note that for any $\varepsilon \in (0, \alpha)$ we may set $\delta(d) = 2^{(\varepsilon - \alpha)d/2}$ and have

$$2 \cdot \exp\left(-\frac{\delta^2}{3} \cdot \mu\right) \in \exp\left(-2^{\varepsilon d + o(d)}\right).$$

We have chosen a function δ that tends towards 0 exponentially quickly (in d) such that the Chernoff bound is superexponentially small (in d). To concretise let $t = |L_j(\mathbf{x}_1, \ldots, \mathbf{x}_i)|$, $N = |L|$, and μ be our expected value for t. Henceforth N, μ, δ, and functions of them, implicitly depend on d. The Chernoff bound tells us there is a superexponentially small probability that t deviates from μ by a multiplicative factor not in $[1 - \delta, 1 + \delta]$. This is equivalent to the statement that there is a superexponentially small probability that μ deviates from t by a multiplicative factor not in $[1/(1 + \delta), 1/(1 - \delta)]$.

We now follow the analysis of [BBHT98]. Set θ such that $\sin^2 \theta = t/N$ and also φ such that $\sin^2 \varphi = \mu/N$. If we knew t exactly we would know θ and therefore be able to reuse the original analysis exactly. Given that we do not, we make use of the expected value μ. The optimal number of Grover iterations is $\tilde{m} = (\pi - 2\theta)/4\theta$ and our estimate of this value is $m_\varphi = \lfloor \pi/(4\varphi) \rfloor$. We calculate, for $\lfloor \pi/(4\varphi) \rfloor = \pi/(4\varphi) - \gamma_\varphi$ with $\gamma_\varphi \in [0, 1)$,

$$\begin{aligned} |m_\varphi - \tilde{m}| &= |\pi/(4\varphi) - \gamma_\varphi - \pi/(4\theta) + 1/2| \\ &\leq \frac{\pi}{4} \cdot |1/\varphi - 1/\theta| + 1/2 \end{aligned}$$

Let $t_N = \sqrt{t/N} = \sin\theta$ and note that $t_N \in 2^{-\Omega(d)}$. Note that $1/\varphi$ is maximised when μ is minimised as $\mu = t/(1 + \delta)$, and that in this case $|1/\varphi - 1/\theta| = 1/\varphi - 1/\theta$. Similarly, $1/\varphi$ is minimised for $\mu = t/(1 - \delta)$ and in this case $|1/\varphi - 1/\theta| = 1/\theta - 1/\varphi$. We use the fact $0 \leq x \leq \arcsin x \leq x + x^3$ for $x \in [0, 1]$ to bound these quantites.

First, for $\mu = t/(1+\delta)$ we have

$$\begin{aligned}
1/\varphi - 1/\theta &= 1/\arcsin(t_N/\sqrt{1+\delta}) - 1/\arcsin(t_N) \\
&\leq \sqrt{1+\delta}/t_N - 1/(t_N + t_N^3) \\
&= \frac{(\sqrt{1+\delta}-1)\cdot t_N + \sqrt{1+\delta}\cdot t_N^3}{t_N^2 + t_N^4} \\
&\leq (\sqrt{1+\delta}-1)\cdot t_N^{-1} + \sqrt{1+\delta}\cdot t_N
\end{aligned}$$

Similarly, for $\mu = t/(1-\delta)$ we have

$$\begin{aligned}
1/\theta - 1/\varphi &= 1/\arcsin(t_N) - 1/\arcsin(t_N/(\sqrt{1-\delta})) \\
&\leq \frac{t_N \cdot \left((1-\delta) - (1-\delta)^{3/2}\right) + t_N^3}{t_N^2\cdot(1-\delta) + t_N^4} \\
&\leq \frac{t_N \cdot \left((1-\delta) - (1-\delta)^{3/2}\right) + t_N^3}{t_N^2\cdot(1-\delta)} \\
&= (1-(1-\delta)^{1/2})\cdot t_N^{-1} + (1-\delta)^{-1}\cdot t_N.
\end{aligned}$$

Given that $1-\sqrt{1-\delta} \geq \sqrt{1+\delta} - 1$ and $1/(1-\delta) \geq \sqrt{1+\delta}$ for $\delta \in (0,1)$ the latter bound on $1/\theta - 1/\varphi$ is always greater. Hence we let $E = (1-(1-\delta)^{1/2})\cdot t_N^{-1} + (1-\delta)^{-1}\cdot t_N$. If we let $e = |m_\varphi - \tilde{m}|$ then we have

$$\begin{aligned}
e &\leq \frac{\pi}{4}\cdot E + 1/2 \\
&\leq E + 1/2
\end{aligned}$$

In [BBHT98] e is always upper bounded by $1/2$; we have some extra term depending on how close the expected list size μ is to t. We have $0 \leq |\cos((2m_\varphi+1)\theta)| = |\sin(2\theta e)|$ and note that

$$|\cos((2m_\varphi+1)\theta)|^2 = \cos^2((2m_\varphi+1)\theta) = \sin^2(2\theta e)$$

is exactly the probability that upon measuring the superposition after m_φ Grover iterations we fail and receive a non root.

We expand $2\theta e$ as $2\theta E + \theta$ and use both $\sin(A+B) = \sin A \cos B + \sin B \cos A$ and $0 \leq |\sin x| \leq |x|$ to find

$$\begin{aligned} |\sin(2\theta e)| &= |\sin(2\theta E)\cos\theta + \sin\theta\cos(2\theta E)| \\ &\leq |\sin(2\theta E)| + t_N \\ &\leq |2\theta E| + t_N \\ &= 2\theta E + t_N. \end{aligned}$$

Since $0 \leq \theta = \arcsin(t_N) \leq t_N + t_N^3$ we have

$$\begin{aligned} 2\theta E + t_N &\leq 2 \cdot (t_N + t_N^3) \cdot \left((1 - (1-\delta)^{1/2}) \cdot t_N^{-1} + (1-\delta)^{-1} \cdot t_N \right) + t_N \\ &= 2 \cdot (1 - (1-\delta)^{1/2}) + O(t_N) \end{aligned}$$

Finally, since $0 \leq \sin^2 x \leq |\sin x|$ the probability of failure is bound by the quantity above; $2 \cdot (1 - (1-\delta)^{1/2}) + O(t_N)$. The first summand is of the form $\delta + O(\delta^2)$, which is exponentially small in d. The second summand is also exponentially small in d.

Hence the probability of measuring a non root after applying the number of Grover iterations implied by the expected list size is exponentially small in the dimension of the lattice, except with superexponentially small probability. If we were certain of the value of t we would achieve the same statement but without the 'except with superexponentially small probability' qualifier.

For fixed classical $\mathbf{x}_1$ in Algorithm 8, or $\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}$ in Algorithm 9, we perform at most k Grover searches. We wish to show that each of these Grover searches fails with exponentially small probability. When performing the Grover's search on L_i for $i \geq 2$ in Algorithm 8, or on $L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{j-1})$ for $i \geq j$ in Algorithm 9, we consider fewer than $|L|^k \in 2^{O(d)}$ filtered intermediate lists within the superposition. For each i we have a particular $(p_i, \alpha_i, \varepsilon_i, \delta_i)$ with which we repeat the above analysis.

Let $\hat{\delta} = \max\{\delta_i\}_i$. By the union bound, the probability that any such intermediate list, in any of the k Grover searches, differs from the expected list size by a factor not in $[1-\hat{\delta}, 1+\hat{\delta}]$ is superexponentially small, and therefore the probability of failure for any of the Grover searches is exponentially small, except with superexponentially small probability.

4.5.2 Reprise

The procedure we have just described is summarised in Algorithm 8. If we want to use this algorithm to solve the Approximate k-List problem (Definition 4.4.1), we

additionally require that the number of output solutions is equal to the size of the input lists. Using the results of Theorem 4.4.1, we can express the complexity of Algorithm 8 for the Approximate k-List problem via the determinant of the target configuration $\mathbf{C}$ and its minors.

Theorem 4.5.1. *Given input $L_1, \ldots, L_k \subset S^{d-1}$ and a configuration $\mathbf{C} \in \mathscr{C}$, such that* (4.2) *holds, Algorithm 8 solves the Approximate k-List problem in expected time*

$$T_{\text{k-List}} \in \widetilde{O}\left(\left(\left(\frac{1}{\det(\mathbf{C})}\right)^{\frac{k+1}{2(k-1)}} \cdot \sqrt{\det(\mathbf{C}[1,\ldots,k-1])}\right)^{d/2}\right) \tag{4.12}$$

using

$$M_{\text{k-List}} \in \widetilde{O}\left(\left(\frac{1}{\det(\mathbf{C})}\right)^{\frac{d}{2(k-1)}}\right)$$

classical memory and $\mathrm{poly}(d)$ *quantum memory with success probability in* $1 - 2^{-\Omega(d)}$.

Proof. From (4.2) the input lists $L_1, \ldots, L_k$ should be of sizes

$$|L| \in \widetilde{O}\left(\left(\frac{1}{\det(\mathbf{C})}\right)^{\frac{d}{2(k-1)}}\right)$$

to guarantee a sufficient number of solutions. This determines the requirement for classical memory. Furthermore, since all intermediate lists are stored in the superposition, we require quantum registers of size $\mathrm{poly}(d)$.

We can simplify the expression for $T^{\text{Q}}_{\text{BLS}}$ given in (4.11) by noting that $|L_2(\mathbf{x}_1)| \geq |L_3(\mathbf{x}_1, \mathbf{x}_2)| \geq \ldots \geq |L_{k-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})| = |L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})|$ in expectation. The dominant term in the sum appearing in (4.11) is $\sqrt{\left(\frac{|L_k|}{|L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})|}\right)}$.

We consider the product $\sqrt{|L_2(\mathbf{x}_1)| \cdots |L_{k-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})|}$ which is the amplitude amplification term from (4.11) where the final term $\sqrt{|L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})|}$ has cancelled with the denominator of the dominant term described above. Using Theorem 4.4.2 this product becomes less than

$$|L|^{\frac{k-2}{2}} \left(\sqrt{\det(\mathbf{C}[1,\ldots,k-1])}\right)^{d/2}.$$

in expectation.

We have applied Jensen's inequality (in 'reverse') here. In particular, for some real convex function φ and X a real valued random variable we have $\varphi(\mathbb{E}[X]) \leq \mathbb{E}[\varphi(X)]$. The function $f\colon [0,\infty) \to [0,\infty),\ x \mapsto \sqrt{x}$ is concave, so $-f$ is convex. We therefore have $-f(\mathbb{E}[X]) \leq \mathbb{E}[-f(X)] \Leftrightarrow \mathbb{E}[f(X)] \leq f(\mathbb{E}[X])$.
Therefore taking $L_2(\mathbf{x}_1)$ as an example we have

$$\mathbb{E}\left[\sqrt{|L_2(\mathbf{x}_1)|}\right] \leq \sqrt{\mathbb{E}[|L_2(\mathbf{x}_1)|]}.$$

We arrive at the expression for $T_{\text{k-List}}$ in the theorem statement by simplifying $|L| \cdot |L|^{1/2} \cdot |L|^{\frac{k-2}{2}} \left(\sqrt{\det(\mathbf{C}[1,\ldots,k-1])}\right)^{d/2}$ as

$$\begin{aligned}
|L|^{\frac{k+1}{2}} \left(\sqrt{\det(\mathbf{C}[1,\ldots,k-1])}\right)^{d/2} &\in \widetilde{O}\left(\left(\left(\frac{1}{\det(\mathbf{C})}\right)^{\frac{d}{2(k-1)}}\right)^{\frac{k+1}{2}} \left(\sqrt{\det(\mathbf{C}[1,\ldots,k-1])}\right)^{d/2}\right) \\
&= \widetilde{O}\left(\left(\frac{1}{\det(\mathbf{C})}\right)^{\frac{d(k+1)}{4(k-1)}} \left(\sqrt{\det(\mathbf{C}[1,\ldots,k-1])}\right)^{d/2}\right) \\
&= \widetilde{O}\left(\left(\left(\frac{1}{\det(\mathbf{C})}\right)^{\frac{k+1}{2(k-1)}} \cdot \sqrt{\det(\mathbf{C}[1,\ldots,k-1])}\right)^{d/2}\right)
\end{aligned}$$

The success probability of Algorithm 8 is determined by the probability that measuring $\mathbf{A}\,|\mathbf{0}^{\otimes(k-1)}\rangle$ returns the $(\mathbf{x}_2,\ldots,\mathbf{x}_k)$ part of a solution tuple. For this we consider the precise form of the state $|\Psi_F\rangle$ given in (4.7). This state is obtained by running $k-1$ (sequential) Grover algorithms. While we have been assuming that each application of Grover's algorithm outputs an idealised state in the form of a uniform superposition over elements of the appropriate filtered list, in fact each tensor $|\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})}\rangle$ in this state is of the form

$$\begin{aligned}
|\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})}\rangle = \sqrt{\frac{1-\epsilon_i}{|L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|}} \sum_{\mathbf{x}\in L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})} |\mathbf{x}\rangle + \\
\sqrt{\frac{\epsilon_i}{|L_i \setminus L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})|}} \sum_{\mathbf{x}\in L_i\setminus L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})} |\mathbf{x}\rangle\,,
\end{aligned}$$

where $\epsilon_i \in 2^{-\Omega(d)}$. This follows from the explanation given in the 'Intermezzo' of Section 4.5.1, in particular that using the expected list sizes to parametrise our Grover's searches fails with exponentially small probability.

Therefore, the probability of measuring the $(\mathbf{x}_2, \ldots, \mathbf{x}_k)$ part of a solution tuple is

$$a = \left(\prod_{i=2}^{k-1} \frac{1-\epsilon_i}{|L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})|} \right) \cdot \frac{1-\epsilon_k}{|L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})|}$$

which is in the class $(1 - 2^{-\Omega(d)}) \cdot \left(\left(\prod_{i=2}^{k-1} |L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})| \right) \cdot |L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})| \right)^{-1}$.

> The square root of this quantity is the non idealised amplitude $\langle \mathbf{z} | \Psi_F \rangle$ that we calculated an idealised value for in (4.10). It is smaller than the idealised amplitude by a factor that is exponentially close to 1, which means the optimal number of amplitude amplification iterations differs only by a factor similarly close to 1.

According to Theorem 2.6.2, after performing

$$\left(\left(\prod_{i=2}^{k-1} |L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})| \right) \cdot |L_k(\mathbf{x}_1, \ldots, \mathbf{x}_{k-2})| \right)^{1/2}$$

amplitude amplification iterations, in Step 9 we measure a 'good' $(\mathbf{x}_2, \ldots, \mathbf{x}_k)$ with probability at least $\max\{a, 1-a\} \in 1 - 2^{-\Omega(d)}$. □

4.5.3 Quantum version of the [HKL18] configuration search algorithm

The main difference between the two algorithms for the configuration problem – the algorithm due to Bai–Laarhoven–Stehlé [BLS16] and due to Herold–Kirshanova–Laarhoven [HKL18] – is that the latter constructs intermediate filtered lists, Figure 4.2. We use quantum enumeration to construct and classically store these lists.

For a fixed $\mathbf{x}$, quantum enumeration repeatedly applies Grover's algorithm to a list L, where each successful application returns a random vector from the filtered list $L(\mathbf{x})$ with probability in $1 - 2^{-\Omega(d)}$. The quantum complexity of obtaining one vector by running Grover's algorithm is $O\left(\sqrt{\frac{|L|}{|L(\mathbf{x})|}}\right)$. We check that the returned vector belongs to $L(\mathbf{x})$ by checking its inner product with $\mathbf{x}$. Repeating this process $\widetilde{O}(|L(\mathbf{x})|)$ times we obtain the entire list $L(\mathbf{x})$ stored classically in time $\widetilde{O}(\sqrt{|L| \cdot |L(\mathbf{x})|})$.

The advantage of constructing the lists $L_i(\mathbf{x})$ is that we can now efficiently prepare the state $|\Psi_{L_2(\mathbf{x})}\rangle \otimes \cdots \otimes |\Psi_{L_k(\mathbf{x})}\rangle$ (cf. Line 3 in Algorithm 8) and run amplitude amplification on the states $|\Psi_{L_i(\mathbf{x})}\rangle$ rather than on $|\Psi_{L_i}\rangle$. This may give a speed up

if the complexity of the lines 3–11 of Algorithm 8, which is of order $\widetilde{O}(T^{\mathrm{Q}}_{\mathrm{BLS}}/|L_1|)$, dominates the cost of quantum enumeration, which is of order $\widetilde{O}(\sqrt{|L_i| \cdot |L_i(\mathbf{x})|})$. In general, we can continue creating the 'levels' as in [HKL18] (see Figure 4.2b) using quantum enumeration and at some level switch to the quantum BLS style algorithm. For example, for some level $1 < j \leq k-1$, we apply quantum enumeration to obtain $L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{j-1})$ for all $i \geq j$. Then, rather than looping over $\mathbf{x}_1 \in L_1$ and applying Grover's algorithm to $|\Psi_{L_2}\rangle \otimes \cdots \otimes |\Psi_{L_k}\rangle$ (cf. lines 3–11 of Algorithm 8), we loop over the $(j-1)$ tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}) \in L_1 \times \cdots \times L_{j-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{j-2})$ and apply Grover's algorithm to $|\Psi_{L_j(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle \otimes \cdots \otimes |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle$. Note that since we have these lists stored in memory, we can efficiently create the latter superposition. In this way we obtain a quantum 'hybrid' between the HKL and the BLS algorithms: until some level j, we construct the intermediate filtered lists using quantum enumeration, create superpositions over all the filtered lists at level j for some fixed values $\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}$, and apply Grover's algorithm to find (if it exists) the $(k-j+1)$ tuple $(\mathbf{x}_j, \ldots, \mathbf{x}_k)$ such that $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ is a solution to the configuration problem. Pseudocode for this approach is given in Algorithm 9. Note that on lines 11 and 12 we only write the register altered by the Grover iteration, even though it acts on more registers.

Let us now analyse Algorithm 9. Recall Definition 4.4.4 of the level of a list. To simplify notation we denote $L_i^{(r)} = L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{r-1})$ for all $r \leq i$, so r denotes the level e.g. $L_i^{(1)} = L_i$ and $L_i^{(i)} = L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})$. In the following all O notations are omitted. Each quantum enumeration of $L_i^{(r)}$ from $L_i^{(r-1)}$ for $i \geq r \geq 2$ costs

$$\sqrt{\left|L_i^{(r-1)}\right| \left|L_i^{(r)}\right|}.$$

Let $j \geq 2$ determine the level to which we create the filtered intermediate lists via quantum enumeration, i.e. we create $L_j^{(j)}, \ldots, L_k^{(j)}$. For a given $2 \leq \ell \leq j$ we form $L_\ell^{(\ell)}, \ldots, L_k^{(\ell)}$ once for each tuple $(\mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1})$, that is $\prod_{r=1}^{\ell-1} \left|L_r^{(r)}\right|$ times. Once the lists $L_i^{(j)}$ for $i \geq j$ are constructed we can form the state

$$|\Psi_{L_j^{(j)}}\rangle \otimes |\Psi_{L_{j+1}^{(j)}}\rangle \otimes \cdots \otimes |\Psi_{L_{k-1}^{(j)}}\rangle \otimes |\Psi_{L_k^{(j)}}\rangle$$

and sequential applications of Grover's algorithm then gives

$$|\Psi_{L_j^{(j)}}\rangle \otimes |\Psi_{L_{j+1}^{(j+1)}}\rangle \otimes \cdots \otimes |\Psi_{L_{k-1}^{(k-1)}}\rangle \otimes |\Psi_{L_k^{(k-1)}}\rangle$$

Algorithm 9 Hybrid quantum algorithm for the Configuration Problem.

Input: $L_1, \ldots, L_k \subset S^{d-1}$, target configuration $\mathbf{C} \in \mathbb{R}^{k \times k}$, $\varepsilon > 0$, hybrid parameter $2 \leq j \leq k-1$.

Output: L_{out}, list of k-tuples satisfying configuration $\mathbf{C}$.

1: $L_{\text{out}} \leftarrow \emptyset$
2: **for all** $\mathbf{x}_1 \in L_1$ **do**
3: Use quantum enumeration to construct $L_i(\mathbf{x}_1)$ for $i \geq 2$
4: **for all** $\mathbf{x}_2 \in L_2(\mathbf{x}_1)$ **do**
5: Use quantum enumeration to construct $L_i(\mathbf{x}_1, \mathbf{x}_2)$ for $i \geq 3$
6: $\ddots$ ▷ This represents an arbitrary but constant number of nested loops.
7: **for all** $\mathbf{x}_{j-1} \in L_{j-1}(\mathbf{x}_1, \ldots, \mathbf{x}_{j-2})$ **do**
8: Use quantum enumeration to construct $L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{j-1})$ for $i \geq j$
9: Prepare the state $|\Psi_{L_j(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle \otimes \cdots \otimes |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle$
10: **for all** $i = j+1, \ldots, k-1$ **do**
11: For some a_i do $\mathbf{G}(f_{[i-1],i})^{a_i} |\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle = |\Psi_{L_i(\mathbf{x}_1,\ldots,\mathbf{x}_{i-1})}\rangle$
12: For some a_k do $\mathbf{G}(f_{[k-2],k})^{a_k} |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle = |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle$
13: Let $\mathbf{A}$ be unitary that implements lines 9–12, i.e.

$$\mathbf{A} |\mathbf{0}^{\otimes(k-j+1)}\rangle = |\Psi_{L_j(\mathbf{x}_1,\ldots,\mathbf{x}_{j-1})}\rangle \otimes \cdots \otimes |\Psi_{L_{k-1}(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle \otimes |\Psi_{L_k(\mathbf{x}_1,\ldots,\mathbf{x}_{k-2})}\rangle$$

14: Apply $\mathbf{G}(\mathbf{A}, f \cap g)^a$ for some a to $\mathbf{A} |\mathbf{0}^{\otimes(k-j+1)}\rangle$, where g is defined in (4.9).
15: Measure all the registers to obtain a tuple $(\mathbf{x}_j, \ldots, \mathbf{x}_k)$.
16: **if** $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ satisfies C **then**
17: $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{(\mathbf{x}_1, \ldots, \mathbf{x}_k)\}$.

in time

$$\left(\sqrt{\frac{\left|L_{j+1}^{(j)}\right|}{\left|L_{j+1}^{(j+1)}\right|}} + \cdots + \sqrt{\frac{\left|L_{k-1}^{(j)}\right|}{\left|L_{k-1}^{(k-1)}\right|}} + \sqrt{\frac{\left|L_{k}^{(j)}\right|}{\left|L_{k}^{(k-1)}\right|}} \right)$$

(cf. lines 11–12 in Algorithm 9). On Step 14 the unitary $\mathbf{G}(\mathbf{A}, f \cap g)$ must be executed

$$\sqrt{\left|L_j^{(j)}\right| \cdots \left|L_{k-1}^{(k-1)}\right| \cdot \left|L_k^{(k-1)}\right|}$$

times to maximise the probability that the system returns a 'good' tuple $(\mathbf{x}_j, \ldots, \mathbf{x}_k)$. Such tuples may not exist. For $j \geq 3$, i.e. for *fixed* $\mathbf{x}_1, \mathbf{x}_2$, we expect to have less than 1 such tuple. Therefore in many instances the measurement will return a uniform $(k-j+1)$ tuple, which we classically check against the target configuration $\mathbf{C}$. To simplify the expression for time complexity we introduce two functions of j. The first

captures the maximum cost of any of the *quantum enumerations*,

$$qe(j) = \max_{2\le\ell\le j}\left\{\prod_{r=1}^{\ell-1}\left|L_r^{(r)}\right| \cdot \max_{\ell\le i\le k}\sqrt{\left|L_i^{(\ell-1)}\right| \cdot \left|L_i^{(\ell)}\right|}\right\}.$$

The outer max over ℓ denotes the level being quantumly enumerated with the product denoting the number of times that level is quantumly enumerated. The inner max determines the largest cost to quantumly enumerate a list on that level. The second function of j captures the cost of the *quantum search* component of Algorithm 9, i.e. the Grover's search and amplitude amplification,

$$qs(j) = \prod_{r=1}^{j-1}\left|L_r^{(r)}\right| \cdot \left(\sqrt{\frac{\left|L_{j+1}^{(j)}\right|}{\left|L_{j+1}^{(j+1)}\right|}} + \cdots + \sqrt{\frac{\left|L_{k-1}^{(j)}\right|}{\left|L_{k-1}^{(k-1)}\right|}} + \sqrt{\frac{\left|L_k^{(j)}\right|}{\left|L_k^{(k-1)}\right|}}\right) \cdot \sqrt{\left|L_j^{(j)}\right| \cdot \ldots \left|L_{k-1}^{(k-1)}\right| \cdot \left|L_k^{(k-1)}\right|}.$$

Overall, given on input a level $j \ge 2$, the runtime of Algorithm 9 is

$$T^{\mathrm{Q}}_{\mathrm{Hybrid}}(j) = \max\{qe(j), qs(j)\}. \tag{4.13}$$

The above complexity can be expressed via the determinant and subdeterminants of the target configuration $\mathbf{C}$ using Theorem 4.4.2. An optimal value of j for a given $\mathbf{C}$ can be found using numerical optimisations by looking for j that minimises (4.13).

Numerical optimisations

We performed numerical optimisations for the target configuration $\mathbf{C}$ which minimise the runtime of the two algorithms for the configuration problem given in this section. The upper part of Table 4.2 gives time optimal c for (4.12) and the c' of the corresponding memory requirements for various k, i.e. the time and memory complexities are $2^{\mathsf{c}d+o(d)}$ and $2^{\mathsf{c}'d+o(d)}$ respectively. These constants decrease for $k \ge 3$ and, eventually, those for time become close to the value 0.2989. Looking at (4.11) the expression decreases when the lists $L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})$ under the square root become smaller. When k is large enough, in particular, once $k \ge 6$, there is a target configuration that ensures that $|L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})|$ are of expected size 1 for levels $i \ge 4$. So for $k \ge 6$, under the observation that the maximal value in the sum appearing in (4.11) is attained by the last summand, the runtime of Algorithm 8 becomes $T^{\mathrm{Q}}_{\mathrm{BLS}} = |L_1|^{3/2} \cdot \sqrt{|L_2(\mathbf{x}_1)|\,|L_3(\mathbf{x}_1, \mathbf{x}_2)|}$. The list sizes can be made explicit using (4.3) when a configuration $\mathbf{C}$ is such that

$|L_i(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1})|$ are of expected size 1. Namely, for $k \geq 6$ and for configuration $\mathbf{C}$ that minimises the runtime exponent, (4.11) with the help of (4.3) simplifies to

$$\left(\left(\frac{1}{\det(\mathbf{C})}\right)^{\frac{5}{2(k-1)}} \sqrt{\det(\mathbf{C}[1,2,3])}\right)^{d/2}.$$

We experimentally observe that this expression becomes constant for time optimal configurations $\mathbf{C}$ and large enough k.

k	2	3	4	5	6	...	28	29	30
Quantum version of [BLS16] Algorithm 8									
Time	0.3112	0.3306	0.3289	0.3219	0.3147	...	0.29893	0.29893	0.29893
Space	0.2075	0.1907	0.1796	0.1685	0.1596	...	0.1395	0.1395	0.1395
Quantum Hybrid version of [BLS16, HKL18] Algorithm 9									
Time	0.3112	0.3306	0.3197	0.3088	0.3059	...	0.29893	0.29893	0.29893
Space	0.2075	0.1907	0.1731	0.1638	0.1595	...	0.1395	0.1395	0.1395
Low memory Quantum Hybrid version of [BLS16, HKL18] Algorithm 9									
Time	0.3112	0.3349	0.3215	0.3305	0.3655	...	0.6352	0.6423	0.6490
Space	0.2075	0.1887	0.1724	0.1587	0.1473	...	0.0637	0.0623	0.0609

Table 4.2 Asymptotic complexity exponents base 2 for the approximate k-List problem. The top part gives optimised runtime exponents and the corresponding memory exponents for Algorithm 8. These are the results of the optimisation (minimisation) of the runtime expression given in (4.12). The middle part gives the runtime and memory exponents for Algorithm 9, again optimising for time, with $j = 2$, i.e. when we use quantum enumeration to create the second level lists $L_i(\mathbf{x}_1)$ for $i \geq 2$. The bottom part gives the exponents for Algorithm 9 with $j = 2$ in the memory optimal setting.

The optimal runtime exponents for the hybrid given as Algorithm 9 with $j = 2$ are given in the middle part of Table 4.2. Experimentally, we establish that $j = 2$ is optimal for small values of k and that this algorithm has the same behaviour for large values of k as Algorithm 8. Indeed, for the runtime optimal configuration $\mathbf{C}$ the intermediate filtered lists on the same level increase in size 'from left to right', i.e. $|L_2(\mathbf{x}_1)| \leq |L_3(\mathbf{x}_1)| \leq \cdots \leq |L_k(\mathbf{x}_1)|$. It is also the case for the runtime optimal configuration that $|L_k(\mathbf{x}_1)|$ becomes almost $|L_k|$ (i.e. the target inner product determined by $\mathbf{C}_{1,k}$ is very close to 0), so quantumly enumerating this list brings no advantage over Algorithm 8 where we use the initial list L_k, of essentially the same size, in Grover's algorithm.

4.6 Quantum configuration search for $k = 3$ via triangle finding

In this section we introduce a distinct approach to finding solutions of the configuration problem of Definition 4.4.3 via triangle listing in graphs. We achieve this by embedding solution tuples of the configuration problem as triangles in a graph and repeatedly applying a triangle finding algorithm. Throughout this section we assume that $L_1 = \cdots = L_k = L$. We solve the configuration problem with $k = 3$ and the *balanced* configuration $\mathbf{C}$. This is memory optimal configuration with all off diagonal entries equal to $-1/k$ and the size of L determined by (4.2). In particular

$$\mathbf{C} = \begin{pmatrix} 1 & -1/3 & -1/3 \\ -1/3 & 1 & -1/3 \\ -1/3 & -1/3 & 1 \end{pmatrix}, \quad |L| \in \widetilde{O}\left((3\sqrt{3}/4)^{d/2}\right).$$

Let $G = (V, E)$ be an undirected graph with known vertices and let $f_E \colon V^2 \to \{0, 1\}$ be a function such that $(\mathbf{x}_1, \mathbf{x}_2) \mapsto 0$ if $(\mathbf{x}_1, \mathbf{x}_2) \in E$ and 1 otherwise. A triangle is a set $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\} \subset V$ of pairwise distinct vertices such that $f_E(\mathbf{x}_1, \mathbf{x}_2) = f_E(\mathbf{x}_1, \mathbf{x}_3) = f_E(\mathbf{x}_2, \mathbf{x}_3) = 0$. In the balanced configuration case we define the vertices of V as the elements of the list L and let $(\mathbf{x}_i, \mathbf{x}_j) \in E \Leftrightarrow |\langle \mathbf{x}_i, \mathbf{x}_j \rangle + 1/3| \leq \varepsilon$ for some $\varepsilon > 0$. The function f_E is therefore realised by performing a d dimensional inner product and a comparison. We note that $\langle \mathbf{x}, \mathbf{x} \rangle = 1$, and so by setting $\varepsilon < 1 < 1 + 1/3$ we do not allow loops in our graph, i.e. we never have $f_E(\mathbf{x}, \mathbf{x}) = 0$. Throughout we let $|V| = n$ and $|E| = m$.

4.6.1 The algorithm of Buhrman et al.

We start with the triangle finding algorithm of [BdWD+01]. Given the balanced configuration and $k = 3$ on S^{d-1}, we set

$$n = |L| \in \widetilde{O}\left((3\sqrt{3}/4)^{d/2}\right), \; m = |L| \cdot \mathbb{E}\left[|L(\mathbf{x}_1)|\right] \in \widetilde{O}\left(n^2(8/9)^{d/2}\right) \tag{4.14}$$

by (4.2) and (4.3) respectively.[7] The values of n and m are approximately $2^{0.1887d+o(d)}$ and $2^{0.2925d+o(d)}$ respectively. We expect $\Theta(n)$ triangles to be found since $|L|$ is chosen as in (4.2). The algorithm of [BdWD+01] consists of three steps

[7] As we are in the balanced configuration case, and our input lists are identical, (4.4.2) has no dependence on j, i.e. we may take any $j > 1$ for $\mathbb{E}\left[|L(\mathbf{x}_1)|\right]$.

1. Use Grover's algorithm to find any edge $(\mathbf{x}_1, \mathbf{x}_2) \in E$ among all potential $O(n^2)$ edges.

2. Use Grover's algorithm to find a vertex $\mathbf{x}_3 \in V$ such that $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ is a triangle.

3. Apply amplitude amplification to steps 1–2.

Note that the algorithm returns triangles from the graph according to some distribution, we will argue below that it is uniform. Explicitly step 1 applies the relevant Grover iteration to the equal superposition over all pairs of vertices,

$$\mathbf{G}(f_E)^e \left(\frac{1}{n} \sum_{(\mathbf{x}_1,\mathbf{x}_2)\in V^2} |\mathbf{x}_1\rangle \otimes |\mathbf{x}_2\rangle \right) = \sqrt{\frac{\epsilon}{n^2 - m}} \sum_{(\mathbf{x}_1,\mathbf{x}_2)\notin E} |\mathbf{x}_1\rangle \otimes |\mathbf{x}_2\rangle + \sqrt{\frac{1-\epsilon}{m}} \sum_{(\mathbf{x}_1,\mathbf{x}_2)\in E} |\mathbf{x}_1\rangle \otimes |\mathbf{x}_2\rangle,$$

where $e \in O\left(\sqrt{n^2/m}\right)$. Above $\epsilon \leq m/n^2 \in 2^{-\Omega(d)}$ represents the probability of failure, so as in Theorem 4.5.1 this will only alter the optimal number of amplitude amplification iterations by a similarly small factor, which we disregard going forwards.

We then consider a function that recognises triangles defined as $f_T \colon V^3 \to \{0,1\}$, $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \mapsto 0 \Leftrightarrow f_E(\mathbf{x}_1, \mathbf{x}_2) = f_E(\mathbf{x}_1, \mathbf{x}_3) = f_E(\mathbf{x}_2, \mathbf{x}_3) = 0$ and 1 otherwise. We consider the superposition above tensored with a further uniform superposition over the vertices, namely

$$\frac{1}{\sqrt{m}} \sum_{(\mathbf{x}_1,\mathbf{x}_2)\in E} |\mathbf{x}_1\rangle \otimes |\mathbf{x}_2\rangle \otimes \frac{1}{\sqrt{n}} \sum_{\mathbf{x}_3 \in V} |\mathbf{x}_3\rangle.$$

and apply $t \in O(\sqrt{n})$ iterations of $\mathbf{G}(f_T)$. We call the final state $|\Psi_F\rangle$, specifically

$$|\Psi_F\rangle = \mathbf{G}(f_T)^t \cdot (\mathbf{G}(f_E)^e \otimes \mathbf{I}_{\bar{d}}) \left(\left(\frac{1}{n}\right)^{3/2} \sum_{(\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3)\in V^3} |\mathbf{x}_1\rangle \otimes |\mathbf{x}_2\rangle \otimes |\mathbf{x}_3\rangle \right)$$

Let $\mathbf{A}\,|\mathbf{0}^{\otimes 3}\rangle \to |\Psi_F\rangle$. We apply amplitude amplification $\mathbf{G}(\mathbf{A}, f_T)^a$ for a determined by the probability that measuring $\mathbf{A}\,|\mathbf{0}^{\otimes 3}\rangle$ returns a triangle, calculated below.

We first argue that t is the correct number of iterations for $\mathbf{G}(f_T)$. For any edge in G we expect a subconstant number of triangles. We calculate the number of triangles we expect a given edge to be part of as $\mathbb{E}\left[|L(\mathbf{x}_1, \mathbf{x}_2)|\right] = (\sqrt{3}/2)^{d/2}$ using (4.3) and

setting $j = 3$, that is

$$\begin{aligned}\mathbb{E}\left[|L_3(\mathbf{x}_1, \mathbf{x}_2)|\right] &= |L| \cdot \left(\det(\mathbf{C}[1,2,3])/\det(\mathbf{C}[1,2])\right)^{d/2} \\ &= \left(3\sqrt{3}/4\right)^{d/2} \cdot \left(\frac{16}{27} \cdot \frac{9}{8}\right)^{d/2} = \left(\sqrt{3}/2\right)^{d/2}.\end{aligned}$$

Therefore if a given edge is included in a triangle, we expect it to be included in one triangle, and hence apply $t \in O(\sqrt{n})$ Grover iterations to find the final vertex of the triangle. This will succeed with constant probability greater than $1/2$, and so the asymptotic probability that $\mathbf{A}$ succeeds is the probability that step 1 selects an edge of a triangle. There are $\Theta(n)$ triangles and m edges, so this is $\Theta(n/m)$, we must therefore apply amplitude amplification $a \in \Theta(\sqrt{m/n})$ times.

The cost of the first step is $O(\sqrt{n^2/m})$ and the second step $O(\sqrt{n})$. From (4.14), and that the costs of step 1 and step 2 are additive, we see that $O(\sqrt{n})$ dominates, therefore steps 1–2 cost $O(\sqrt{n})$. By applying the amplitude amplification in step 3 $O(\sqrt{m/n})$ times, the cost of finding a triangle is $O(\sqrt{m})$.[8]

The algorithm finds one of the n triangles uniformly at random. Indeed, step 1 uniformly selects an edge of G. If this edge is part of a triangle we expect it to be part of a single triangle as argued above, so uniformly selecting an edge of a triangle selects a uniform triangle. By the coupon collector problem we must repeat the algorithm $\widetilde{O}(n)$ times to find all the triangles. Therefore the total cost of finding all triangles is $\widetilde{O}(n\sqrt{m}) = \widetilde{O}(|L|^{3/2}|L(\mathbf{x}_1)|^{1/2}) \approx 2^{0.3349d+o(d)}$ using $2^{0.1887d+o(d)}$ memory. This matches the complexity of Algorithm 9 for $k = 3$ in the balanced setting (see Table 4.2).

4.6.2 The algorithm of Le Gall–Nakajima

The [BdWD+01] algorithm finds a triangle in an arbitrary graph in $O(n + \sqrt{nm})$ queries. We measure the *sparsity* of a graph G in terms of c such that $m = n^c$, and say that a graph becomes more sparse as c decreases. When $m \in \Omega(n)$, i.e. $c \geq 1$, the sparsity determines the size of the dominant summand for the [BdWD+01] algorithm.

Another triangle finding algorithm whose complexity depends on the sparsity of the graph is given in [GN17], and for certain sparsity regimes it outperforms the query complexity of the [BdWD+01] algorithm. Finally, the algorithm of [Gal14] finds a triangle in an arbitrary graph using $\widetilde{O}(n^{5/4})$ queries, regardless of sparsity.

[8] Note that this differs from [BdWD+01] as in general either of step 1 or 2 may dominate and we also make use of the existence of $\Theta(n)$ triangles.

The operations counted in the works discussed above are *queries* to an oracle that returns whether an edge exists between two vertices in our graph. The algorithms of [Gal14, GN17] use more complex data structures than [BdWD+01], and converting the query complexity into a time or memory complexity is more challenging. We report only the query complexity in this section. We also assume that a triangle returned by the [GN17] algorithm is (close to) uniform over the triangles in a given graph.

The main result of [GN17] is an interpolation between the query complexities of [BdWD+01, Gal14] as the sparsity of the graph varies.

Theorem 4.6.1 ([GN17, Theorem 1])**.** *There exists a quantum algorithm that solves, with high probability, the triangle finding problem over graphs of n vertices and m edges with query complexity*

$$\begin{cases} O(n+\sqrt{nm}) & \text{if } 0 \leq m \leq n^{7/6} \\ \widetilde{O}(nm^{1/14}) & \text{if } n^{7/6} \leq m \leq n^{7/5} \\ \widetilde{O}(n^{1/6}m^{2/3}) & \text{if } n^{7/5} \leq m \leq n^{3/2} \\ \widetilde{O}(n^{23/30}m^{4/15}) & \text{if } n^{3/2} \leq m \leq n^{13/8} \\ \widetilde{O}(n^{59/60}m^{2/15}) & \text{if } n^{13/8} \leq m \leq n^2. \end{cases}$$

More specifically it is shown that for $c \in (7/6, 2)$ a better complexity can be achieved than shown in [BdWD+01, Gal14]. At the end points the two previous algorithms are recovered; [BdWD+01] for $c \leq 7/6$ and [Gal14] for $c = 2$. We recall that these costs are in the query model, and that for $c > 7/6$ where we do not recover [BdWD+01], we do not convert them into time complexity.

We explore two directions that follow from the above embedding of the configuration problem into a graph. The first is the most naïve, we simply calculate the sparsity regime as per Theorem 4.6.1 that the graph lies in, and calculate the query complexity of finding all triangles. The second splits our list into triples of distinct sublists and considers graphs formed from the union of said triples of sublists. The sublists are parameterised such that the sparsity and the expected number of triangles in these new graphs can be altered.

Naïve triangle finding

With $G = (V, E)$ and n, m as in (4.14), we expect to have

$$m = O\left(n^{2+\delta}\right) = O\left(n^{1.5500}\right), \ \delta = \log(8/9)/\log(3\sqrt{3}/4).$$

Therefore the sparsity is $c = 1.55$ and by Theorem 4.6.1 finding a triangle takes $\widetilde{O}(n^{23/30}m^{4/15}) = \widetilde{O}\left(n^{1.1799}\right)$ queries. We require $\widetilde{O}(n)$ repeats, giving a total query complexity of $\widetilde{O}(n^{2.1799}) = 2^{0.4114d+o(d)}$. This is not competitive with classical algorithms [HK17] or the approach of Section 4.5.

Altering the sparsity

Let n remain as in (4.14) and $\gamma \in (0,1)$ be such that we consider $\Gamma = n^{1-\gamma}$ disjoint sublists of L, $\ell_1, \ldots, \ell_\Gamma$, each with $n' = n^\gamma$ elements. There are $O(n^{3(1-\gamma)})$ triples of these sublists (ℓ_i, ℓ_j, ℓ_k) with $1 \leq i < j < k \leq \Gamma$.

Letting $\Gamma = n^{1-\gamma}$ then the number of triples with indices as stated is

$$\sum_{i=1}^{\Gamma-2}\left(\sum_{j=1}^{\Gamma-i-1} j\right) = \frac{1}{2}\sum_{i=1}^{\Gamma-2}(\Gamma-i+1)(\Gamma-i) = \frac{\Gamma^3}{6} - \frac{\Gamma^2}{2} + \frac{\Gamma}{3}.$$

We define the union of the sublists within a triple as $\ell_{ijk} = \ell_i \cup \ell_j \cup \ell_k$ which has size $O(n')$. Let $G_{ijk} = (\ell_{ijk}, E_{ijk})$ with $(\mathbf{x}_1, \mathbf{x}_2) \in \ell_{ijk} \times \ell_{ijk}$ being an edge $(\mathbf{x}_1, \mathbf{x}_2) \in E_{ijk} \Leftrightarrow |\langle \mathbf{x}_1, \mathbf{x}_2 \rangle + 1/3| \leq \varepsilon$ as before. Using Theorem 4.4.2 each G_{ijk} is expected to have

$$m' = |\ell_{ijk}|\,|\ell_{ijk}(x_1)| \in O\left((n')^2(8/9)^{d/2}\right) = O\left(n^{2\gamma}(8/9)^{d/2}\right)$$

edges. By finding all triangles in all G_{ijk} we find all triangles in G, and as n is chosen to expect $\Theta(n)$ triangles in G, we have sufficiently many solutions for the underlying configuration problem. We expect by Theorem 4.4.2

$$\begin{aligned}|\ell_{ijk}||\ell_{ijk}(\mathbf{x}_1)||\ell_{ijk}(\mathbf{x}_1, \mathbf{x}_2)| &= |\ell_{ijk}|\left(|\ell_{ijk}|(8/9)^{d/2}\right)\left(|\ell_{ijk}|(2/3)^{d/2}\right) \\ &\in O(n^{3\gamma})(16/27)^{d/2} = O(n^{3\gamma-2})\end{aligned}$$

triangles per ℓ_{ijk}. We must at least test each ℓ_{ijk} once, regardless of $O(n^{3\gamma-2})$. The sparsity of ℓ_{ijk} given γ is calculated as

$$m' \in O\left((n')^{2+\beta(\gamma)}\right),\ \beta(\gamma) = \frac{\log(8/9)}{\gamma\log(3\sqrt{3}/4)}. \tag{4.15}$$

In conclusion then, for a given γ the number of ℓ_{ijk} to test is $O(n^{3(1-\gamma)})$, the number of triangles to find per ℓ_{ijk} is $O(n^{3\gamma-2})$ – we always perform at least one triangle finding attempt and assume finding them all takes $\widetilde{O}(n^{3\gamma-2})$ repeats – and when using Theorem 4.6.1 we are in the sparsity regime $c(\gamma) = 2 + \beta(\gamma)$ for $\beta(\gamma)$ defined in (4.15).

Let a, b represent the exponents of n', m' respectively in Theorem 4.6.1 when $m' = (n')^{c(\gamma)}$. We therefore minimise, for $\gamma \in (0,1)$, the exponent of n in

$$O(n^{3(1-\gamma)}) \cdot \widetilde{O}(n^{3\gamma-2}) \cdot \widetilde{O}((n')^a (m')^b).$$

In the final term we calculate

$$(n')^a = n^{\gamma a} \text{ and } (m')^b = (n^{\gamma \cdot c(\gamma)})^b = n^{(2\gamma+\gamma\cdot\beta(\gamma))b}$$

from which we arrive at the following exponent for n,

$$3(1-\gamma) + \max\{0, 3\gamma - 2\} + \gamma a + \left(2\gamma + \frac{\log(8/9)}{\log(3\sqrt{3}/4)}\right) b.$$

The minimal query complexity of $n^{1.7298+o(d)} = 2^{0.326d+o(d)}$ is achieved at $\gamma = \frac{2}{3}$.

Note that the same triangle may be discovered several times in the above algorithm. For example if we define $G_{ij} = (\ell_{ij}, E_{ij})$, with ℓ_{ij} and E_{ij} defined analogously to ℓ_{ijk} and E_{ijk}, then any triangle in G_{ij} will be present in G_{ijk} for all $j < k \leq \Gamma$. Similarly, any triangle in $G_i = (\ell_i, E_i)$ will be present in G_{ijk} for all $i < j < k \leq \Gamma$. However, the cost of these rediscoveries is already accounted for since we find all triangles in each G_{ijk}. Unfortunately, it does not seem that finding an approach that reduces the number of such rediscoveries would improve the asymptotics of the above. Concretely, each G_{ijk} contains $G_i, G_j, G_k, G_{ij}, G_{ik}$, and G_{jk}, each of which contain an expected $O(n^{3\gamma-2})$ triangles.[9] Therefore these potential rediscoveries do not alter the number of triangles to be found in each G_{ijk}.

4.7 Conclusion

Since the publication of this work a new sieving algorithm with heuristic complexity $2^{0.2570d+o(d)}$ has been introduced [CL21]. To obtain this improvement the authors make use of a quantum random walk on Johnson graphs. This time complexity is lower than the quantum version of [BDGL16] described in [Laa16], although it requires different assumptions on quantum memory. The authors provide interesting time memory trade offs for their algorithm to vary the amount of different forms of memory required, which would then allow one to optimise the algorithm if given precise relative costs for these different forms of memory. To achieve the $2^{0.2570d+o(d)}$ time complexity the algorithm

[9] Given that e.g. $|\ell_i| = n^\gamma, |\ell_{ij}| = 2n^\gamma, |\ell_{ijk}| = 3n^\gamma$ the expected numbers of triangles in G_i, G_{ij}, G_{ijk} differ only by subexponential factors.

requires exponential quantum memory, as opposed to the polynomial quantum memory required in this work and previous quantum lattice sieves that apply Grover's algorithm to speed up the search for reducing lattice vectors. The authors also make the distinction between two types of qRAM, which are called QRACM and QRAQM. Throughout this thesis when referring to qRAM we have been implicitly referring to QRACM, where the qRAM operation is used to create arbitrary superpositions of classical registers. In QRAQM the same is true, but for quanutm registers. The relative difficulty of constructing these different forms of qRAM, and indeed qRAM in any form, is an open topic of research [Kup13, AGJO$^+$15].

Future work directions for this chapter could include examining whether the quantum random walks described in [CL21] could be applied to k-sieves, even for fixed small k, e.g. 3. It would also be interesting to understand the extent to which quantum random walks can be applied to the exponential width quanutm sieve given in the full version of this work [KMPM19b, Sec. 7]. Other open problems are listed in Section 4.2.

Chapter 5

Lattice Sieves in (Quantum) Cost Metrics

Shortly after heuristic lattice sieves became an area of research with [NV08, MV10c] the effect of using quantum search to find reducible pairs was examined [LMv13, Laa16]. This is a natural line of inquiry; under the heuristics we consider we have a large list of uniform points on some sphere. We fix one and try to find others that are close under an angular metric, so that we may form reducible pairs. At least in the simple setting where we do not use locality sensitive techniques, none of the list points are a priori more likely to form a reducible pair, and so we have an unstructured list search with a large list, say of size N.

Classically we have a cost of $\Theta(N)$ to find a reducible pair provided there are not many, say some small constant number per fixed point. If reducible pairs were significantly more frequent then we could take a smaller list. Quantumly, using Grover's search algorithm, we have a cost of $\Theta(\sqrt{N})$ to find a reducible pair. Given that we repeat this process for each of the points in the list we have a classical cost of $\Theta(N^2)$ and a quantum cost of $\Theta(N^{3/2})$.

An immediate question is 'how is this cost measured?' In the classical case one computes the angle between the fixed point in the list and all other points. Each such computation can be realised as an inner product on real vectors of some dimension and using some precision, followed by a comparison. Ultimately, when the dimension and precision are fixed, each computation will take $\Theta(1)$ classical gates. In the quantum case we apply the Grover iteration approximately $\sqrt{N}$ times in series, and then measure the resulting quantum state. Each of these Grover iterations will also require $\Theta(1)$ quantum gates.

Replacing $\Theta(1)$ with 1 in the above is called the query model. In our case a classical query means to ask an oracle for the angle between two vectors on the sphere, and a quantum query means to ask an oracle for the quantum state given by applying a Grover iteration to an input quantum state. If we go beyond this query model, and try our best to tear open the $\Theta(1)$ and determine the gate cost of implementing these oracles, then even if we succeed with perfect accuracy we are left with a number of classical gates, and a number of quantum gates.

Perhaps a more useful question is therefore 'how can the reported classical and quantum costs be meaningfully compared?' In this chapter we approach this question in several ways following [JS19]. Here the idea is that a certain amount of classical control is necessary to enact gates in the quantum circuit. By calculating this classical control we can then compare the gate cost of a classical circuit to the classical gate cost required to enact a quantum circuit. In [JS19] the authors introduce the gate metric, which assumes $\Theta(1)$ classical gates are required to enact each quantum gate. Implicit in this metric is the idea that identity wires in the quantum circuit cost nothing to enact – that self correcting quantum memories exist that require no control to maintain the state they contain. Whether such quantum memories are realisable is an active area of research, and so the other cost the authors introduce is the depth width cost, which assumes $\Theta(1)$ classical gates are required to enact any part of the quantum circuit, including identity wires. In particular this metric assumes that some amount of active error correction will be required to ensure the quantum circuit correctly computes its output. In this chapter we further consider a particular style of active error correction that uses surface codes. Surface codes encode logical qubits into a grid of noisy physical qubits, where the size of this grid grows with the size of the circuit. Some fraction of the noisy qubits that encode a logical qubit are measured at regular intervals, and reinitialised based on classical computations over the outcome of these measurements. By adapting the scripts of an in depth study of surface codes given in [GE21] we are able to estimate the classical cost of (a part of) the surface code computation.

We therefore have three different metrics that we can compare to a classical gate cost for a circuit that achieves the same task. Each of the three represents a different physical reality; the gate metric represents a world where self correcting quantum memory can be built, the depth width metric represents one where it cannot, but active error correction has a constant cost that is independent of the size of the circuit, and the surface code metric represents a world where the cost of active error correction grows with the size of the circuit. It should be noted that if a sieve has a complexity of e.g. $2^{cn+o(n)}$ in the quantum query model, none of the above costs alter the leading

constant c. Instead, for dimensions of cryptanalytical relevance we are interested in providing estimates for the cost of quantum sieving that can be meaningfully compared to similar estimates for classical sieves.

All of the above metrics require a quantum circuit to be designed, the gate metric and depth width metric because they require the number of gates and the depth and width of the circuit respectively, and the surface code metric because the number of physical qubits used to encode a logical qubit is an increasing function of the depth and width of the circuit. We therefore must design quantum circuits for various lattice sieves to give these costs.

To do this we consider one iteration of Nguyen–Vidick style sieves, in particular we have some list of size N and from it we wish to find N reducible pairs so that we can iterate. To build this quantum circuit we need to specify the unitaries that make up a Grover iteration, and in particular $\mathbf{R}_f$ (recall Section 2.6.2). A natural choice for f would be a function that has the fixed point from our list hardwired into it and takes the inner product of this fixed point with its input. If this inner product is sufficiently large, which implies the angle between the points in the list is sufficiently small, then the input is a root. However, we choose instead to use the `popcount` filter introduced in Section 2.5.4. This filter is used in the sieving implementation that at the time held the record on the Darmstadt SVP challenges [GS10] and has a significantly smaller and simpler quantum circuit. In particular it only requires addition and comparison, rather than floating point multiplication. However, this choice does require us to design a suitable filtered quantum search procedure, that first uses Grover search to find roots of the `popcount` filter before using amplitude amplification techniques to find elements which also satisfy the inner product constraint.

Given this filtered quantum search routine we analyse iterations of various sieves in such a way that exposes the `popcount` parameters along with any sieve parameters. We can then design quantum circuits for these sieve iterations and minimise their complexity under a given metric. This finally allows us our comparison to classical sieving, and we conclude by discussing the remaining barriers to realising quantum sieving with the costs that we estimate.

Contributions

The author of this thesis contributed equally with the other authors to Sections 5.2 and 5.3, equally with Vlad Gheorghiu and John Schanck to Section 5.5, and equally with John Schanck to Sections 5.6 and 5.7. The author of this thesis contributed equally with Martin Albrecht and John Schanck to the software.

Acknowledgements

The authors thank Léo Ducas for helpful discussions regarding ListDecodingSearch.

5.1 Estimating quantum speedups for lattice sieves

The work presented here is an amended and annotated version of what is published as

Albrecht M.R., Gheorghiu V., Postlethwaite E.W., Schanck J.M. (2020) Estimating Quantum Speedups for Lattice Sieves.

In: Moriai S., Wang H. (eds) Advances in Cryptology – ASIACRYPT 2020. ASIACRYPT 2020. Lecture Notes in Computer Science, vol 12492. Springer, Cham.
https://doi.org/10.1007/978-3-030-64834-3_20

The appendices are available at https://eprint.iacr.org/2019/1161.

An implementation is available at https://github.com/jschanck/eprint-2019-1161.

5.2 Introduction

Sieving algorithms for the shortest vector problem (SVP) in a lattice have received a great deal of attention recently [AKS01, ADH+19a, BDGL16, Duc18a, Laa15, NV08]. The attention mostly stems from lattice based cryptography, as many attacks on lattice based cryptographic constructions involve finding short lattice vectors [ADPS16, LP11, MR09].

Lattice based cryptography is thought to be secure against quantum adversaries. None of the known algorithms to solve SVP (to a small approximation factor) do so in subexponential time, but this is not to say that there is no gain to be had given a large quantum computer. Lattice sieve algorithms use near neighbour search (NNS) as a subroutine; near neighbour search algorithms use black box search as a subroutine; and Grover's quantum search algorithm [Gro96] gives a square root improvement to the query complexity of black box search. A black box search that is expected to take $\Theta(N)$ queries on classical hardware will take $\Theta(\sqrt{N})$ queries on quantum hardware using Grover's algorithm.

Previous work has analysed the effect of quantum search on the query complexity of lattice sieves [Laa16, LMv13]. Of course, one must implement the queries efficiently in order to realise the improvement in practice. Recent work has given concrete quantum resource estimates for the black box search problems involved in key recovery attacks on AES [GLRS16, JNRV20] and preimage attacks on SHA-2 and SHA-3 [AMG+16].

In this work, we give explicit quantum circuits that implement the black box search subroutines of several quantum lattice sieves. Our quantum circuits are efficient enough to yield a cost improvement in dimensions of cryptanalytic interest. However, for the most performant sieve that we analyse the cost improvement is small and several barriers stand in the way of achieving it.

Outline and contributions

We start with some preliminaries in Section 5.3. In Section 5.4 we introduce and analyse a filtered quantum search procedure. We present our quantum circuit for `popcount` (recall Section 2.5.4) in Section 5.5. In Section 5.6 we provide a heuristic analysis of the probability that `popcount` successfully identifies pairs of vectors that are close to each other. The `popcount` operation is our primary optimisation target, and is typically less expensive than a full inner product computation. This analysis may be of independent interest; previous work [ADH+19a, Duc18a] relied largely on experimental data for choosing `popcount` parameters. In Section 5.7, we rederive the overall cost of the NNS subroutines of three lattice sieves. Our cost analysis exposes the impact of the `popcount` parameters so that we can numerically optimise these in parallel with the sieve parameters.

We have chosen to profile the Nguyen–Vidick sieve [NV08], the `bgj1` specialisation [ADH+19a] of the Becker–Gama–Joux sieve [BGJ15], and the Becker–Ducas–Gama–Laarhoven sieve [BDGL16]. We have chosen these three sieves as they are, respectively, the earliest and most conceptually simple, the most performant implemented (at the time of publication), and the fastest known asymptotically.

Finally, we optimise the cost of classical and quantum search under various cost metrics to produce Figure 5.2 of Section 5.8. We conclude by discussing barriers to obtaining the reported quantum advantages in NNS, the relationship between SVP and NNS, and future work. Both the data produced, and the source code used to compute it, are available at `https://github.com/jschanck/eprint-2019-1161`. We consider our software a contribution in its own right; it is documented, easily extensible and allows for the inclusion of new nearest neighbour search strategies and cost models.

Interpretation

Quantum computation seems to be more difficult than classical computation. As such, there will likely be some minimal dimension, a crossover point, below which classical sieves outperform quantum ones. Our estimates give non-trivial crossover points for the sieves we consider. Yet, our results do not rule out the relevance of

quantum sieves to lattice cryptanalysis. The crossover points that we estimate are well below the dimensions commonly thought to achieve 128 bits of security against quantum adversaries. However, our initial logical circuit level analysis (Figure 5.2, q: depth-width) is optimistic. It ignores the costs of quantum random access memory and quantum error correction.

To illustrate the potential impact of error correction, we apply a cost model developed by Gidney and Ekerå to our quantum circuits. The Gidney–Ekerå model was developed as part of a recent analysis of Shor's algorithm [GE21]. In the Gidney–Ekerå model, the crossover point for the NNS algorithm underlying the Becker–Ducas–Gama–Laarhoven sieve [BDGL16] is dimension 312. In this dimension, the classical and quantum variants both perform $2^{119.0}$ operations and need at least $2^{78.3}$ bits of (quantum accessible) random access memory. A large cost improvement is obtained asymptotically, but for cryptanalytically relevant dimensions the improvement is tenuous. Between dimensions 352 and 824 our estimate for the quantum cost grows from appoximately 2^{128} to approximately 2^{256}. In dimension 352 this is an improvement of a factor of $2^{1.8}$ over our estimate for the classical cost. In dimension 824 the improvement is by a factor of $2^{14.4}$.

We caution that a memory constraint would significantly reduce the range of cryptanalytically relevant dimensions. For instance, an adversary with no more than 2^{128} bits of quantum accessible classical memory is limited to dimension 544 and below. In these dimensions we estimate a cost improvement of no more than a factor of $2^{13.6}$ at the logical circuit level and no more than $2^{7.1}$ in the Gidney–Ekerå metric.

A depth constraint would also reduce the range of cryptanalytically relevant dimensions. The quantum algorithms that we consider would be more severely affected by a depth constraint than their classical counterparts, due to the poor parallisability of Grover's algorithm.

5.3 Preliminaries

5.3.1 Models of computation

We describe quantum algorithms as circuits using the Clifford+T gate set, but we augment this gate set with a table lookup operation (qRAM), recall Section 2.6.1. We note that quantum access to classical RAM is a powerful resource, and the algorithms we describe below fail to achieve an advantage over their classical counterparts when

qRAM is not available. We discuss qRAM at greater length in Section 5.8. We describe classical algorithms as programs for RAM machines (random access memory machines).

RAM machines

We describe classical algorithms in terms of random access memory machines. For comparability with the Clifford+T gate set, we will work with a limited instruction set, e.g. {NOT, AND, OR, XOR, LOAD, STORE}. For comparability with qRAM, LOAD and STORE act on ℓ bit registers.

Cost

The cost of a RAM program is the number of instructions that it performs. One can similarly define the *gate cost* of a quantum circuit to be the number of gates that it performs. Both metrics are reasonable in isolation, but it is not clear how one should compare the two. Jaques and Schanck recommend that quantum circuits be assigned a cost in the unit of RAM instructions to account for the role that classical computers play in dispatching gates to quantum memories [JS19]. They also recommend that the identity gate be assigned unit cost to account for error correction. The *depth-width cost* of a quantum circuit is the total number of gate operations that it performs when one includes identity gates in the count.

5.3.2 Filtered black box search

In Section 5.4 of this work we design a filtered quantum search operation tailored to our needs. We first define a filtered classical search routine and introduce a lemma from the literature we will make use of. Recall the definitions of a predicate, exhaustive search, Grover's algorithm, and amplitude amplification from Section 2.6.2.

A black box search algorithm finds a root of a predicate without exploiting any structure present in the description of the predicate itself. However, we may still use them in cases where some structure is known, for example 'f has M roots' or 'f is expected to have no more than M roots'. Our analyses often use this kind of knowledge of structure. We will also use the fact that the set of predicates on any given finite set can be viewed as a Boolean algebra. We write $f \cup g$ for the predicate with kernel $\mathsf{Ker}\,(f) \cup \mathsf{Ker}\,(g)$ and $f \cap g$ for the predicate with kernel $\mathsf{Ker}\,(f) \cap \mathsf{Ker}\,(g)$.

If f is expensive to evaluate, we may try to decrease the cost of exhaustive search by applying a search filter. We say that a predicate g is a filter for f if $f \neq g$ and

$|f \cap g| \geq 1$. We say that g recognises f with a false positive rate of

$$\rho_f(g) = 1 - \frac{|f \cap g|}{|g|},$$

and a false negative rate of

$$\eta_f(g) = 1 - \frac{|f \cap g|}{|f|}.$$

A classical filtered search evaluates $g(0), f(0), g(1), f(1), g(2), f(2)$, and so on until a root of $f \cap g$ is found. The evaluation of $f(i)$ is skipped when i is not a root of g, which may reduce the cost of filtered search below that of exhaustive search.

We wish to design a quantum filtered search, which will make use of the following lemma. It is an integral part of [BBHT98, Thm. 3] which allows Grover's algorithm to maintain the same asymptotic query complexity if the number of roots is unknown. We will refer to this paper by BBHT, and the algorithm analysed in [BBHT98, Thm. 3] as the BBHT algorithm.

Lemma 5.3.1 (Lemma 2 of [BBHT98])**.** *Suppose that measuring* $\mathbf{D}|0\rangle$ *would yield a root of* f *with probability* $\sin^2(\theta)$*. Fix a positive integer* m*. Let* j *be chosen uniformly from* $\{0, \ldots, m-1\}$*. The expected probability that measuring* $\mathbf{G}(f)^j\mathbf{D}|0\rangle$ *yields a root of* f *is* $\frac{1}{m}\sum_{j=0}^{m-1}\sin^2((2j+1)\cdot\theta) = \frac{1}{2} - \frac{\sin(4m\theta)}{4m\sin(2\theta)}$*. If* $m > 1/\sin(2\theta)$ *then this quantity is at least* $1/4$*.*

5.3.3 Lattice sieving and near neighbour search on the sphere

In Section 2.5 we introduced lattice sieves, heuristics that are commonly used to analyse them, the measure of several geometric figures on the sphere, and the `popcount` filter. In this paper we consider 2-sieves on full rank lattices $\Lambda \subset \mathbb{R}^d$, and assume the elements of a list L held by a lattice sieve are i.i.d. $\mathbf{v}_i \leftarrow \mathsf{U}(S^{d-1})$. Therefore a pair $(\mathbf{u}, \mathbf{v})$ is reducible if and only if $\theta(\mathbf{u}, \mathbf{v}) < \pi/3$. The `popcount` filter is used as a first approximation to $\theta(\cdot\,,\cdot)$ and by modelling L as a subset of S^{d-1} we may translate some lattice sieves into the language of (angular) near neighbour search on the sphere. To wit, we want to take a list $L \subset S^{d-1}$ and find $|L|$ reducible pairs so that the sieve can be iterated.

The near neighbour search subroutine of a Nguyen–Vidick style sieve that we examine is given below as AllPairSearch. This algorithm represents one iteration of the sieve, to continue the sieve one takes each pair $(\mathbf{u}, \mathbf{v}) \in L'$ and returns $\mathbf{u} - \mathbf{v}$. Progress is made provided $\theta = \pi/3 - \varepsilon$ for any $\varepsilon \in (0, \pi/3)$. The smaller ε the smaller the expected size of L required to iterate will be, so we consider $\varepsilon \to 0$.

Algorithm 10 AllPairSearch.

Require: A list $L = (\mathbf{v}_1, \mathbf{v}_2, \dots \mathbf{v}_N) \subset S^{d-1}$ of N points. Parameter $\theta \in (0, \pi/2)$.
Ensure: A list of pairs $(\mathbf{u}, \mathbf{v}) \in L \times L$ with $\theta(\mathbf{u}, \mathbf{v}) \leq \theta$.

1: **function** AllPairSearch($L; \theta$)
2: $L' \leftarrow \emptyset$
3: **for** $1 \leq i < N$ **do**
4: $L_i \leftarrow (\mathbf{v}_{i+1}, \dots, \mathbf{v}_N)$
5: Search L_i for any number of $\mathbf{u}$ that satisfy $\theta(\mathbf{u}, \mathbf{v}_i) \leq \theta$.
6: For each such $\mathbf{u}$ found, add $(\mathbf{u}, \mathbf{v}_i)$ to L'.
7: If $|L'| \geq N$, **return** L'.
8: **return** L'

Recall the geometric figures on the sphere introduced in Section 2.5.2 and the description of the `popcount` filter in Section 2.5.4, both of which we make frequent use of throughout this work. In particular, in Section 5.6 we calculate the false positive and negative rates of `popcount` under the heuristic that one can fix h and vary $\mathbf{u}$ and $\mathbf{v}$. These calculations, the fact that `popcount` is significantly cheaper than an inner product, and that it is a filter used in many performant sieves [ADH+19a, Duc18a], make `popcount` a good candidate for use as a filter under the techniques of Section 5.4.

5.4 Filtered quantum search

A filter can reduce the cost of a search because a classical computer can branch to avoid evaluating an expensive predicate. A quantum circuit cannot branch inside a Grover search in this way. Nevertheless, a filter can be used to reduce the cost of a quantum search.

The idea is to apply amplitude amplification to a Grover search. The inner Grover search prepares the uniform superposition over roots of the filter, g. The outer amplitude amplification searches for a root of f among the roots of g. We present pseudocode for this strategy in Algorithm 11.

If $|g|$ and $|f \cap g|$ are known, then we can choose the number of iterations of the inner Grover search and the outer amplitude amplification optimally. When these quantities are not known, we can attempt to guess them as in the BBHT algorithm. In our applications, we have some information about $|g|$ and $|f \cap g|$, which we can use to fine tune a BBHT like strategy.

Proposition 5.4.2 gives the cost of Algorithm 11 when we know *1.* a lower bound, Q, on the size of $|f \cap g|$, and *2.* the value of $|g|$ up to relative error γ. In essence, when

a filter with a low false positive rate is used to search a space with few true positives, Algorithm 11 can be tuned such that it finds a root of f with probability at least $1/14$ and at a cost of roughly $\frac{\gamma}{2}\sqrt{N/Q}$ iterations of $\mathbf{G}(g)$.

Algorithm 11 FilteredQuantumSearch.

Require: A predicate f and a filter g defined on $\{0, \ldots, N-1\}$. Integer parameters m_1 and m_2.

Ensure: A root of f or $\perp$.

1: **function** FilteredQuantumSearch($f, g; m_1, m_2$)
2: Sample integers j and k with $0 \leq j < m_1$ and $0 \leq k < m_2$ uniformly at random.
3: Let $\mathbf{A}_j = \mathbf{G}(g)^j\mathbf{D}$.
4: Let $\mathbf{B}_k = \mathbf{G}(\mathbf{A}_j, f \cap g)^k$.
5: Prepare the state $|\psi\rangle = \mathbf{B}_k\mathbf{A}_j|0\rangle$.
6: Let r be the result of measuring $|\psi\rangle$ in the computational basis.
7: **if** $f(r) = 0$ **then**
8: **return** r
9: **return** $\perp$

If we know that the inner Grover search succeeds with probability $x < 1$ then we can compensate with a factor of approximately $\sqrt{1/x}$ more iterations of the outer amplitude amplification. In particular, recall Theorem 2.6.2 and let the $\mathbf{A}$ therein represent the inner Grover mentioned above. Amplitude amplification then requires approximately $\pi/(4\sqrt{x})$ applications of $\mathbf{G}(\mathbf{A}, f)$ to succeed in finding a root with probability at least $\max\{x, 1-x\}$.

We do not know x. However, in our applications, we do know that the value of θ for which $\sin^2(\theta) = |g|/N$ will be fairly small, e.g. $\theta < 1/10$. The following technical lemma shows that, when θ is small, we may assume that $x = 1/5$ with little impact on the overall cost of the search.

Let j and $\mathbf{A}_j$ be as in Algorithm 11. Let $p_\theta(j)$ be the probability that measuring $\mathbf{A}_j|0\rangle$ would yield a root of g. For any $x \in (0,1)$, there is some probability $q_x(m_1)$ that the choice of j is insufficient, i.e. that $p_\theta(j) < x$. We expect to repeat Algorithm 11 a total of $(1 - q_x(m_1))^{-1}$ times to avoid this type of failure.

Lemma 5.4.1. *Fix $\theta \in (0, \pi/2]$ and $x \in [0,1)$. Let $p_\theta, q_x : \mathbb{R} \to \mathbb{R}$ be defined by $p_\theta(j) = \sin^2((2j+1)\cdot\theta)$ and $q_x(m) = \frac{1}{m}|\{j \in \mathbb{Z} : 0 \leq j < m, p_\theta(j) < x\}|$. If $m > \frac{\pi}{4\theta}$, then*

$$q_x(m) < \frac{3\arcsin(\sqrt{x})}{\pi - \arcsin(\sqrt{x})} + \frac{8\theta}{\pi}.$$

Proof. Observe that $q_x(m)$ is exactly the probability that a uniform integer $j \in [m-1]_0$ is such that measuring $\mathbf{A}_j|0\rangle$ returns a root of g with probability less than x, and that

$p_\theta(j) < x$ when $|(2j+1)\theta \bmod \pi| < \arcsin(\sqrt{x})$. Let I_0 be the interval $[0, \arcsin(\sqrt{x}))$. For integers $t \geq 1$ let $I_t = (t\pi - \arcsin(\sqrt{x}), t\pi + \arcsin(\sqrt{x}))$. Then $(I_t)_{t\geq 0}$ are the intervals for which $|(2j+1)\theta \bmod \pi| < \arcsin(\sqrt{x})$. Let $c = c(m)$ be the largest integer for which $[0, (2m-1)\cdot\theta)$ intersects I_c, and note $2m - 1 = 2j + 1$ for $j = m - 1$.

The quantity $mq_x(m)$ counts the number of non negative integers $i < m$ for which $(2i+1)\cdot\theta$ lies in $I_0 \cup I_1 \cup \cdots \cup I_c$. This is no more than $(c+1) + \lfloor (2c+1)\arcsin(\sqrt{x})/(2\theta) \rfloor$.

> Indeed, we have two kinds of interval, I_0 and I_t for integer $t \geq 1$. Given an interval of length ℓ (ultimately of length 2θ) we wish to upper bound the number of such intervals that can fit in intervals of the form $[a, b)$ (for I_0) or (a, b) (for I_t). The intervals of length ℓ may only overlap on their endpoints. Having done this, by adding one we count the number of endpoints that can fit in these intervals, i.e. the number of different $(2i+1)\cdot\theta$.
> In both the $[a, b)$ and (a, b) cases we find that $\lceil (b-a)/\ell \rceil - 1$ intervals can fit, and so $\lceil (b-a)/\ell \rceil \leq \lfloor (b-a)/\ell \rfloor + 1$ endpoints. Setting $a = 0, b = \arcsin(\sqrt{x}), \ell = 2\theta$ gives us $\lfloor \arcsin(\sqrt{x})/2\theta \rfloor + 1$ for the single interval I_0. Setting $a = t - \arcsin(\sqrt{x}), b = t + \arcsin(\sqrt{x}), \ell = 2\theta$ gives us $\lfloor 2\arcsin(\sqrt{x})/2\theta \rfloor + 1$ for the c intervals $I_1, \ldots, I_c$.
> Using the fact that for $\alpha, \beta \in \mathbb{N}$ and $x, y \geq 0$ we have $\alpha\lfloor x \rfloor + \beta\lfloor y \rfloor \leq \lfloor \alpha x + \beta y \rfloor$, from
>
> $$\lfloor \arcsin(\sqrt{x})/2\theta \rfloor + 1 + c \cdot \left(\lfloor 2\arcsin(\sqrt{x})/2\theta \rfloor + 1 \right)$$
>
> we attain the required result.

It follows that $q_x(m) \leq (c+1)/m + (2c+1)\arcsin(\sqrt{x})/2m\theta$. Note that for $c \geq 1$ we have $2m\theta > (2m-1)\theta > c\pi - \arcsin(\sqrt{x})$ and $(c+1)/m < 4\theta/\pi + 1/m$. Hence $q_x(m) < (2c+1)\arcsin(\sqrt{x})/(c\pi - \arcsin(\sqrt{x})) + 4\theta/\pi + 1/m$. Moreover, $q_x(m) > q_x(m-1)$ when $(2m-1)\cdot\theta$ lies in I_c, and $q_x(m) < q_x(m-1)$ otherwise. The upper bound on $q_x(m)$ that we have derived is decreasing as a function of c. Hence the claim holds when $c \geq 1$.

> Note that the m that maximises $q_x(m)$ for a given c, i.e. we enforce $(2m-1)\cdot\theta \in (c\pi - \arcsin(\sqrt{x}), (c+1)\pi - \arcsin(\sqrt{x})]$, is such that $(2m-1)\cdot\theta \in I_c$ but $(2m+1)\cdot\theta \notin I_c$. The counting argument to

upper bound $q_x(m)$ above captures this case. If we take $m' < m$ or $m' > m$ while maintaining the same c then the bound continues to hold. The bound decreases with c so we set $c = 1$ and show the same bound is valid in the $c = 0$ case below.

In the $c = 0$ case we have $q_x(m) \leq \arcsin(\sqrt{x})/2m\theta + 1/m$, and since $m > \pi/4\theta$ we therefore have $q_x(m) < 2\arcsin(\sqrt{x})/\pi + 4\theta/\pi$. This bound is smaller than the one derived above for the $c \geq 1$ case and by observation holds for all m such that $c = 0$.

For m such that $(2m-1)\cdot\theta \in I_0$ we have $q_x(m) = 1$, which our bound satisfies. Indeed $\pi/2 - \theta < (2m-1)\cdot\theta$ follows from $m > \pi/4\theta$ and $(2m-1)\cdot\theta < \arcsin(\sqrt{x})$ since $c = 0$. We therefore have $\pi/2 - \theta < \arcsin(\sqrt{x})$ which implies $2\arcsin(\sqrt{x})/\pi + 4\theta/\pi > 1$. For $m' > m$ such that we maintain $c = 0$ but $(2m'-1)\cdot\theta \notin I_0$ we have $q_x(m') < q_x(m)$, and hence the bound continues to hold.

□

There are situations in which filtering is not effective, e.g. when the false positive rate of g is very high, when evaluting g is not much less expensive than evaluating f, or when f has a very large number of roots. In these cases, other algorithms will outperform Algorithm 11. We remark on these below. Proposition 5.4.2 optimises the choice of m_1 and m_2 in Algorithm 11 for a large class of filters that are typical of our applications.

Proposition 5.4.2. *Suppose that f and g are predicates on a domain of size N and that g is a filter for f. Let $Q \in \mathbb{R}$ be such that $1 \leq Q \leq |f \cap g|$. Let P and γ be real numbers such that $P/\gamma \leq |g| \leq \gamma P$. If $\gamma P/N < 1/200$ and $\gamma\,|f \cap g|\,/P < 1/4$, then there are parameters m_1 and m_2 for Algorithm 11 such that Algorithm 11 finds a root of f with probability at least $1/14$ and has a cost that is dominated by approximately $\frac{\gamma}{2}\sqrt{N/Q}$ times the cost of $\mathbf{G}(g)$ or by approximately $\frac{2}{3}\sqrt{\gamma P/Q}$ times the cost of $\mathbf{R}_{f\cap g}$.*

Proof. Fix $x \in (0,1)$. We will analyse Algorithm 11 with respect to the parameters $m_1 = \left\lceil \frac{\pi}{4}\sqrt{\gamma N/P} \right\rceil$ and $m_2 = \left\lceil \sqrt{\gamma P/3xQ} \right\rceil$. Let θ_g be such that $\sin^2(\theta_g) = |g|\,/N$. Let j and k be chosen as in Algorithm 11. Let $p = p_{\theta_g}(j)$ and $q = q_x(m_1)$ be defined as in Lemma 5.4.1. Note that since $|g|\,/N \leq \gamma P/N < 1/200$ we can use $8\theta_g/\pi < 1/5$ in applying Lemma 5.4.1.

By bounding $\gamma P/N$ above we are bounding the number of elements that pass the filter g, and therefore its false positive rate. The additive $8\theta_g/\pi$ term of Lemma 5.4.1 when applied to the filter g intuitively captures the component of the upper bound on $q_x(m_1)$ determined by the false positive rate of g.

Let $\theta_h(j)$ be such that $\sin^2(\theta_h(j)) = p \cdot |f \cap g| / |g|$. With probability at least $1 - q$ we have $p \geq x$, which implies that $\sin(\theta_h(j)) \geq \sqrt{xQ/\gamma P}$. Since $\gamma |f \cap g| / P < 1/4 \Rightarrow \sin^2(\theta_h(j)) < 1/4$, then $\cos(\theta_h(j)) > \sqrt{3/4}$. Thus $1/\sin(2\theta_h(j)) < \sqrt{\frac{\gamma P}{3xQ}} \leq m_2$. By Lemma 5.3.1 measuring $\mathbf{G}(\mathbf{A}_j, f \cap g)^k \mathbf{A}_j |0\rangle$ yields a root of $f \cap g$ with probability at least $1/4$.

We are applying an 'amplitude amplification' version of Lemma 5.3.1. We have that measuring $\mathbf{A}_j |0\rangle$ returns a root of $f \cap g$ with probability $\sin^2(\theta_h(j))$ since a uniform root of g is measured with probability p and it is also a root of f with probability $|f \cap g| / |g|$. Measuring $\mathbf{G}(\mathbf{A}_j, f \cap g)^k \mathbf{A}_j |0\rangle$ returns a root of $f \cap g$ with probability $\sin^2((2k+1) \cdot \theta_h(j))$, and hence we may apply the lemma by ensuring $m_2 > 1/\sin(2\theta_h(j)) \Leftrightarrow m_2 \cdot \sin(\theta_h(j)) \cdot \cos(\theta_h(j)) > 1/2$.
The condition $\gamma |f \cap g| / P < 1/4$ may seem like both a loose and a strange requirement. Setting the technical factor $\gamma = 1$ for ease, it is equivalent to $|f \cap g| / |g| < 1/4$, and surely the *larger* this ratio, the better? This is true, but if it is sufficiently large then one may as well just search on the filter g, see Remark 5.4.1 below. In fact, given its use above to lower bound $\cos(\theta_h(j))$, the smaller the upper bound on this ratio the better. Indeed, if e.g. $\gamma |f \cap g| / P < 1/8$ then $\cos(\theta_h(j)) > \sqrt{7/8} > \sqrt{3/4}$. In terms of looseness, even if we had $\cos(\theta_h(j)) = 1$ this would allow us to choose $m_2 > (1/2) \cdot \sqrt{\frac{\gamma P}{xQ}}$, only a factor of $\sqrt{3}/2$ smaller than the requirement on m_2 given above.

It follows that Algorithm 11 succeeds with probability at least $(1 - q)/4$.

The algorithm evaluates $\mathbf{G}(g)$ exactly $k \cdot j + 1$ times and evaluates $\mathbf{G}(g)^\dagger$ exactly $k \cdot j$ times. The expected value of $2kj + 1$ is $c_1(x) \cdot \gamma \cdot \sqrt{N/Q}$ where $c_1(x) \approx (\pi/8)/\sqrt{3x}$. Likewise the algorithm evaluates $\mathbf{R}_{f \cap g}$ exactly k times, which is $c_2(x) \cdot \sqrt{\gamma P/Q}$ in expectation where $c_2(x) \approx (1/2)/\sqrt{3x}$.

> We recall that for independent random variables X, Y we have $\mathbb{E}[XY] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$, and that the expectation of the uniform distribution on integers $a, \ldots, b$ is $(b-a)/2$, hence the above.

Taking $x = 1/5$, and applying the upper bound on $q_x(m_1)$ from Lemma 5.4.1, we have $(1 - q_x(m_1))/4 \geq 1/14$, $c_1(x) \approx 1/2$ and $c_2(x) \approx 2/3$. □

Remark 5.4.1. When $\gamma P/N \geq 1/200$ or $\gamma |f \cap g| / P \geq 1/4$ there are better algorithms. If both inequalities hold then classical search finds a root of f quickly. If $\gamma |f \cap g| / P \geq 1/4$ then finding a root of f is not much harder than finding a root of g, so one can search on g directly. If $\gamma P/N \geq 1/200$ then the filter has little effect and one can search on f directly.

Remark 5.4.2. It is helpful to understand when we can ignore the cost of $\mathbf{R}_{f \cap g}$ in Proposition 5.4.2. Roughly speaking, if evaluating f is c times more expensive than evaluating g, then the cost of calls to $\mathbf{G}(g)$ will dominate when $N > c^2 |g|$. In a classical filtered search the cost of evaluating g dominates when $N > c |g|$.

5.5 Circuits for popcount

Consider a program for $\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v})$. This program loads $\mathbf{u}$ and $\mathbf{v}$ from specified memory addresses, computes $h(\mathbf{u})$ and $h(\mathbf{v})$, computes the Hamming weight of $h(\mathbf{u}) \oplus h(\mathbf{v})$, and checks whether it is less than or equal to k. Recall $h(\mathbf{u})$ is defined by n inner products. If the popcount procedure is executed many times for each $\mathbf{u}$, then it may be reasonable to compute $h(\mathbf{u})$ once and store it in memory. Moreover, if $\mathbf{u}$ is fixed for many sequential calls to the procedure, then it may be reasonable to cache $h(\mathbf{u})$ between calls. The algorithms that we consider in Section 5.7 use both of these optimisations.

In this section we describe RAM programs and quantum circuits that compute $\texttt{popcount}_{k,n}(\mathbf{u}, \cdot)$ for a fixed $\mathbf{u}$. These circuits have the value of $h(\mathbf{u})$ hard coded. They load $h(\mathbf{v})$ from memory, compute the Hamming weight of $h(\mathbf{u}) \oplus h(\mathbf{v})$, and check whether the Hamming weight is less than or equal to k. We ignore the initial, one time, cost of computing $h(\mathbf{u})$ and $h(\mathbf{v})$.

5.5.1 Quantum circuit for popcount

Loading $h(\mathbf{v})$ costs a single qRAM gate. Computing $h(\mathbf{u}) \oplus h(\mathbf{v})$ can then be done in-place using a sequence of $\mathbf{X}$ gates that encode $h(\mathbf{u})$. The bulk of the effort is in computing the Hamming weight. For that we use a tree of in-place adders. The final comparison is also computed with an adder, although only one bit of the output is needed. See Figure 5.1 for a full description of the circuit.

We use the Cuccaro–Draper–Kutin–Petrie adder [CDKM04], with 'incoming carry' inputs, to compute the Hamming weight. We argue in favour of this choice of adder in [AGPS19, App. C]. We use the Häner–Roetteler–Svore [HRS17] carry bit circuit for implementing the comparison.

We will later use popcount within filtered quantum searches by defining predicates of the form $g(i) = \texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}_i)$, $i \in \{0, \ldots, N-1\}$. To simplify that later discussion, we cost the entire Grover iteration $\mathbf{G}(g) = \mathbf{D}\mathbf{R}_0\mathbf{D}^\dagger\mathbf{R}_g$ here. In [AGPS19, App. B] we introduce the (possibly multiply controlled) Toffoli gate and discuss the Toffoli count for $\mathbf{G}(g)$, which in turn gives the $\mathbf{T}$ count for $\mathbf{G}(g)$.

The cost of $\mathbf{R}_g$

The $\mathbf{R}_g$ subroutine is computed by running the popcount circuit in Figure 5.1 and then uncomputing the addition tree and $\mathbf{X}$ gates. The circuit uses in-place i bit adders[1] for $i \in \{1, \ldots, \ell-1\}$. The width of the circuit is given in [AGPS19, App. B]. The depth of the circuit is

$$\text{depth} = 2 + d(\text{CARRY}) + \sum_{i=1}^{\ell-1} 2 \cdot d(\text{ADD}_i), \tag{5.1}$$

where $d(\cdot)$ denotes the depth of its argument. The factor of 2 accounts for uncomputation of the ADD_i circuits. The CARRY circuit is only cost once as the carry bit is computed directly into the $|-\rangle$ state during the CARRY circuit itself. The summand 2 accounts for the $\mathbf{X}$ gates used to compute, and later uncompute, $h(\mathbf{u}) \oplus h(\mathbf{v})$.

> The phase kickback mentioned in the caption of Figure 5.1 works as follows. We let $g : \{0, \ldots, N-1\} \to \{0, 1\}$, $i \mapsto \texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}_i)$ map indices of vectors to whether they pass a popcount filter with $\mathbf{u}$ and parameters (k, n) (all left implicit). We consider the unitary $\mathbf{U}_g$

[1] An in-place i bit quantum adder takes two i bit inputs, initialises an ancilla qubit in the $|0\rangle$ state, and returns the addition result in an $i+1$ bit register that includes the new ancilla and overlaps with i bits of the input.

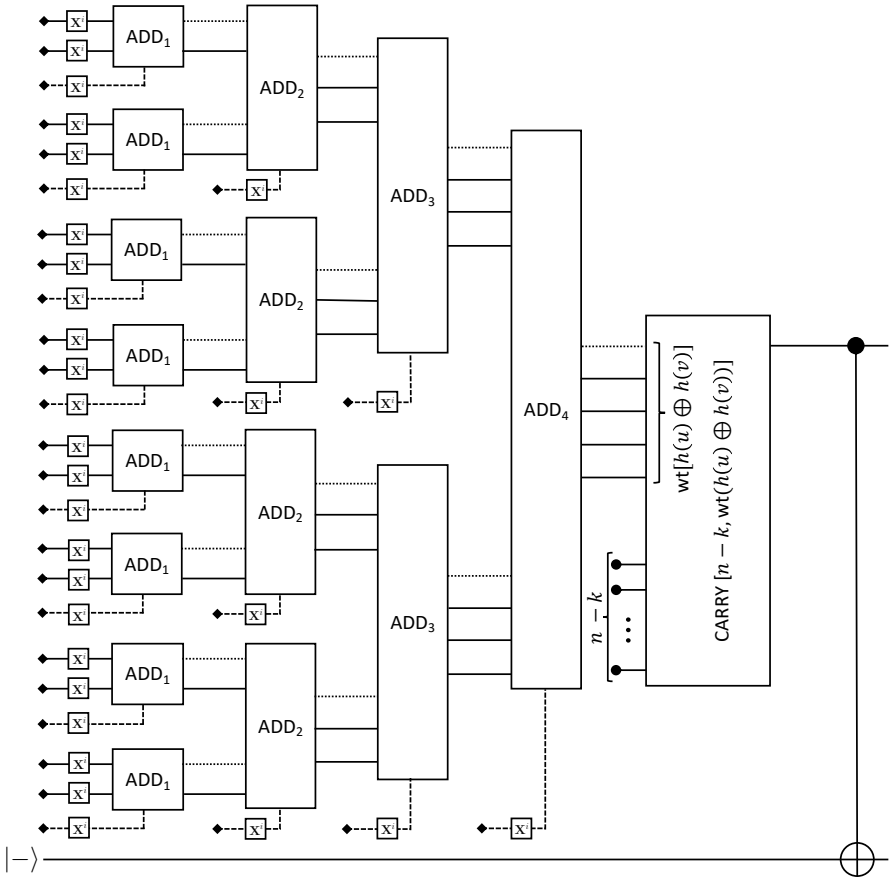


Figure 5.1 A quantum circuit for `popcount`. This circuit computes $h(\mathbf{u}) \oplus h(\mathbf{v})$ for a fixed n bit $h(\mathbf{u})$, computes the Hamming weight of $h(\mathbf{u}) \oplus h(\mathbf{v})$, and checks whether the Hamming weight is less than or equal to k. Here $n = 2^\ell - 1 = 31$. The input qubits are represented as lines ending with a black diamond. The dashed lines represent incoming carry inputs, and the dotted lines represent carry outputs. Not all of the output wires are drawn. For space efficiency, some of the input qubits are fed into the incoming carry qubits of the adders (dashed lines). The $\mathbf{X}^i$ mean that gate $\mathbf{X}$ is applied to input qubit i if bit i of $h(\mathbf{u})$ is 1. The circuit uses a depth $\ell - 1$ binary tree of full bit adders from [CDKM04], where ADD_i denotes an i bit full adder. The output $wt(h(\mathbf{u}) \oplus h(\mathbf{v}))$ from the tree of adders together with the binary representation of the number $n - k$ are finally fed into the input of the CARRY circuit from [HRS17], which computes the carry bit of $n - k + wt(h(\mathbf{u}) \oplus h(\mathbf{v}))$ (the carry bit will be 0 if $wt(h(\mathbf{u}) \oplus h(\mathbf{v})) \leq k$, and 1 otherwise). The final **CNOT** is for illustration only. In actuality, the carry bit is computed directly into an ancilla that is initialised in the $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$ state, so we can obtain the needed phase kickback. The tree of adders and the initial $\mathbf{X}$ gates, but not the CARRY circuit, are run in reverse to clean up scratch space and return the inputs to their initial state. The uncomputation step is not depicted here.

that acts on computational basis states as $\mathbf{U}_g |x\rangle |y\rangle = |x\rangle |y \oplus g(x)\rangle$, for $x \in \{0, \ldots, N-1\}$, $y \in \{0, 1\}$. In general the first register will contain a superposition of these indices. The second register will contain the state $|-\rangle = (1/\sqrt{2}) \cdot (|0\rangle - |1\rangle)$ which has the property that $(1/\sqrt{2}) \cdot (|0 \oplus g(x)\rangle - |1 \oplus g(x)\rangle) = (-1)^{g(x)} \cdot (1/\sqrt{2}) \cdot (|0\rangle - |1\rangle)$. We therefore have that $\mathbf{U}_g |x\rangle |-\rangle = (-1)^{g(x)} |x\rangle |-\rangle$. Recalling Section 2.6.2 this is exactly the requirement for $\mathbf{R}_g$. Since the register containing the state $|-\rangle$ never alters, it is ignored.

The cost of $\mathbf{DR}_0\mathbf{D}^\dagger$

Recall that $\mathbf{D}$ can be any circuit that maps $|0\rangle$ to the uniform distribution on the domain of the search predicate. While there is no serious difficulty in sampling from the uniform distribution on $\{0, \ldots, N-1\}$ for any integer N, when costing the circuit we assume that N is a power of two. In this case $\mathbf{D}$ is simply $\log_2 N$ parallel $\mathbf{H}$ gates. The reflection $\mathbf{R}_0$ is implemented as a multiply controlled Toffoli gate that targets an ancilla initialised in the $|-\rangle$ state. We use Maslov's multiply controlled Toffoli from [Mas16]. The depth and width of $\mathbf{DR}_0\mathbf{D}^\dagger$ are both $O(\log N)$; our software calculates the exact value.

5.5.2 RAM program for `popcount`

Recall that we use a RAM instruction set that consists of simple bit operations and table lookups. A Boolean circuit for `popcount` is schematically similar to Figure 5.1. Let $\ell = \lceil \log_2 n \rceil$. Loading $h(\mathbf{v})$ has cost 1. Computing $h(\mathbf{u}) \oplus h(\mathbf{v})$ takes n XOR instructions and has depth 1. Following [Par09, Table. II], with $c_{FA} = 5$ the number of instructions in a full adder, $(n - \ell - 1)c_{FA} + \ell$ lower bounds the instruction cost of computing the Hamming weight and comparing it with a fixed k. This has depth $(\ell - 1)(\delta_{\text{sum}} + \delta_{\text{carry}}) + 1$. We assume $\delta_{\text{sum}} = \delta_{\text{carry}} = 1$. Thus, the overall instruction count is $6n - 4\ell - 5$ and the overall depth is 2ℓ.

5.5.3 Cost of inner products

The optimal `popcount` parameters will depend on the cost of a computing an inner product in dimension d. The cost of one inner product is amortised over many `popcounts`, and a small change in the `popcount` parameters will quickly suppress the

ratio of inner products to popcounts (see Remark 5.4.2). Hence we only need a rough estimate for the cost of an inner product. We assume 32 bits of precision are sufficient. We then assume schoolbook multiplication is used for scalar products, which costs approximately 32^2 AND instructions. We then assume the cost of a full inner product is approximately $32^2\, d$, i.e. we ignore the cost of the final summation, assuming it is dwarfed by the ANDs.[2]

5.6 The accuracy of popcount

Here we give an analysis of the popcount technique based on some standard simplifying assumptions. We are particularly interested in the probability that a popcount filter identifies a random pair of points as potential neighbours. We are also interested in the probability that a pair of actual neighbours are not identified as potential neighbours, i.e. the false negative rate. Our software computes all of the quantities in this section to high precision.

Let $P_{k,n}(\mathbf{u}, \mathbf{v})$ be the probability that $\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}; h) = 0$ for a uniformly random h (recall $\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}; h) = 0$ if $\mathbf{u}, \mathbf{v}$ pass the filter). In other words, let $h = (h_1, \ldots, h_n)$ be a collection of i.i.d. random variables that are distributed uniformly on the sphere so that $\left\{\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}; h)\right\}_{(\mathbf{u},\mathbf{v})\in S^{d-1}\times S^{d-1}}$ are random variables indexed by $\mathbf{u}, \mathbf{v}$ on the sphere, and define

$$P_{k,n}(\mathbf{u}, \mathbf{v}) = 1 - \mathbb{E}\left[\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}; h)\right].$$

The hyperplane defined by h_i separates $\mathbf{u}$ and $\mathbf{v}$ with probability $\theta(\mathbf{u}, \mathbf{v})/\pi$, and $\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}; h) = 0$ if no more than k of the hyperplanes separate $\mathbf{u}$ and $\mathbf{v}$. Hence,

$$P_{k,n}(\mathbf{u}, \mathbf{v}) = \sum_{i=0}^{k} \binom{n}{i} \cdot \left(\frac{\theta(\mathbf{u}, \mathbf{v})}{\pi}\right)^i \cdot \left(1 - \frac{\theta(\mathbf{u}, \mathbf{v})}{\pi}\right)^{n-i}.$$

Note that $P_{k,n}(\mathbf{u}, \mathbf{v})$ depends only on the angle between $\mathbf{u}$ and $\mathbf{v}$, so it makes sense to define $P_{k,n}(\theta)$. The main heuristic in our analysis of popcount is that $P_{k,n}(\mathbf{u}, \mathbf{v})$ is a good approximation to the probability that $\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}; h) = 0$ for *fixed* h and *varying* $\mathbf{u}$ and $\mathbf{v}$. Under this assumption, all of the quantities in question can be determined by integrating $P_{k,n}(\mathbf{u}, \mathbf{v})$ over different regions of the sphere.

[2] We also tested the effect of assuming 8-bit inner products are sufficient. As expected, this reduces all costs by a factor of two to four and thus does not substantially alter our relative results.

Let $\hat{P}_{k,n}$ denote the event that $\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}; h) = 0$ for uniformly random $\mathbf{u}$, $\mathbf{v}$, and h. Let $\hat{R}_\theta$ be the event that $\theta(\mathbf{u}, \mathbf{v}) \leq \theta$. Recall that $\Pr[\hat{R}_\theta] = C_d(\theta)$, and observe that $\Pr[\hat{R}_\theta]$ is a cumulative distribution with associated density $A_d(\theta) = \frac{\partial}{\partial\theta} C_d(\theta)$. We find, letting $S = S^{d-1}$ for some implicit d,

$$\begin{aligned}\Pr[\hat{P}_{k,n}] &= \int_S \int_S P_{k,n}(\mathbf{u}, \mathbf{v}) \, \mathrm{d}\mu(\mathbf{v}) \, \mathrm{d}\mu(\mathbf{u}) \\ &= \int_S \left(\int_0^\pi P_{k,n}(\theta) \cdot A_d(\theta) \, \mathrm{d}\theta \right) \mathrm{d}\mu(\mathbf{u}) \\ &= \int_0^\pi P_{k,n}(\theta) \cdot A_d(\theta) \, \mathrm{d}\theta. \end{aligned} \tag{5.2}$$

Let $\mathbf{u}, \mathbf{v}$ such that $\theta(\mathbf{u}, \mathbf{v}) \leq \varphi$ be neighbours. The false negative rate is $1 - \Pr[\hat{P}_{k,n} \mid \hat{R}_\varphi]$. The quantity $\Pr[\hat{P}_{k,n} \wedge \hat{R}_\varphi]$ can be calculated by changing the upper limit of integration in (5.2). It follows that

$$1 - \Pr[\hat{P}_{k,n} \mid \hat{R}_\varphi] = 1 - \frac{1}{C_d(\varphi)} \int_0^\varphi P_{k,n}(\theta) \cdot A_d(\theta) \, \mathrm{d}\theta. \tag{5.3}$$

In Section 5.7 we consider $\mathbf{u}$ and $\mathbf{v}$ that are uniformly distributed in a cap of angle $\beta < \pi/2$, rather than the uniformly distributed on the sphere. Let $\hat{B}_{\mathbf{w},\beta}$ be the event that $\mathbf{u}$ and $\mathbf{v}$ are uniformly distributed in a cap of angle β about $\mathbf{w}$. We have

$$\begin{aligned}\Pr[\hat{B}_{\mathbf{w},\beta}] &= \int_S \int_S \mathbb{1}\left\{ \mathbf{w} \in W^{d-1}(\mathbf{u}, \beta, \mathbf{v}, \beta) \right\} \mathrm{d}\mu(\mathbf{v}) \, \mathrm{d}\mu(\mathbf{u}) \\ &= \int_0^{2\beta} W_d(\theta, \beta, \beta) \cdot A_d(\theta) \, \mathrm{d}\theta. \end{aligned} \tag{5.4}$$

In the second line we have used the fact that $\beta < \pi/2$ and $W(\theta, \theta_1, \theta_2)$ is zero when $\theta \geq \theta_1 + \theta_2$. The quantity $\Pr[\hat{R}_\varphi \wedge \hat{B}_{\mathbf{w},\beta}]$ can be computed by changing the upper limit of integration in (5.4) from 2β to $\min\{2\beta, \varphi\}$. We note that $\hat{B}_{\mathbf{w},\beta}$ has no dependence on $\mathbf{w}$ and therefore may also be written $\hat{B}_\beta$. The conditional probability that $\texttt{popcount}_{k,n}(\mathbf{u}, \mathbf{v}; h) = 0$, given that $\mathbf{u}, \mathbf{v}$ are uniformly distributed in a cap B_β, $\Pr[\hat{P}_{k,n} \mid \hat{B}_\beta]$, can be computed using (5.4) and

$$\Pr[\hat{P}_{k,n} \wedge \hat{B}_\beta] = \int_0^{2\beta} P_{k,n}(\theta) \cdot W_d(\theta, \beta, \beta) \cdot A_d(\theta) \, \mathrm{d}\theta. \tag{5.5}$$

The quantity $\Pr[\hat{P}_{k,n} \wedge \hat{R}_\varphi \wedge \hat{B}_\beta]$ can be computed by changing the upper limit of integration in (5.5) from 2β to $\min\{2\beta, \varphi\}$. The false negative rate for `popcount` when restricted to a cap is $1 - \Pr[\hat{P}_{k,n} \mid \hat{R}_\varphi \wedge \hat{B}_\beta]$.

5.7 Tuning popcount for NNS

We now use the circuit sizes from Section 5.5 and the probabilities from Section 5.6 to optimise popcount for use in NNS algorithms. Our analysis is with respect to points sampled independently from the uniform distribution on the sphere. We further restrict our attention to *list-size preserving* parameterisations, which take an input list of size N and return an output list of (expected) size N.

We use the notation for events introduced in Section 5.6. In particular, we write $\hat{R}_\theta$ for the event that a uniformly random pair of vectors are neighbours, i.e. that they lie at angle less than or equal to θ of one another; $\hat{P}_{k,n}$ for the event that popcount identifies a uniformly random pair of vectors as potential neighbours; $\hat{B}_\beta$ for the event that a uniformly random pair of vectors lie in a uniformly random cap of angle β; and $\hat{B}_{\mathbf{w},\beta}$ for the same event except we highlight the cap is centred on $\mathbf{w}$. Throughout this section we use $\texttt{popcount}_{k,n}(\mathbf{u}, \cdot)$, for various fixed $\mathbf{u}$, as a filter for the search predicate $\theta(\mathbf{u}, \cdot) \leq \theta$. We write $\eta(k, n)$ for the false negative rate of popcount. We assume that $\theta(\mathbf{u}, \mathbf{v}) \leq \theta$ is computed using an inner product test. Throughout this section, c_1 represents the instruction cost of the inner product test from Section 5.5.3, $c_2(k, n)$ the instruction cost of popcount from Section 5.5.2, q_1 the quantum cost of the reflection $\mathbf{R}_{f \cap g}$, and $q_2(k, n)$ the quantum cost of $\mathbf{G}(g)$ from Section 5.5.1. We note that c_1 and q_1 have a dependence on d that we suppress. We write $q_0(m)$ for the number of $\mathbf{G}(g)$ iterations that are applied during a quantum search on a set of size m.

Our goal is to minimise the cost of list-size preserving NNS algorithms as a function of the input list size, the popcount parameters k and n, and the other NNS parameters. In a list of N points there are $\binom{N}{2}$ ordered pairs. We expect $\binom{N}{2} \cdot \Pr[\hat{R}_\theta] = \binom{N}{2} \cdot C_d(\theta)$ of these to be neighbours, and we expect a $1 - \eta(k, n)$ fraction of neighbours to be detected by popcount. List-size preserving parmaterisations that use a popcount filter must therefore take an input list of size at least

$$\ell(k, n) = \frac{2}{1 - \eta(k, n)} \cdot \frac{1}{C_d(\theta)}. \tag{5.6}$$

The optimised costs reported in Figure 5.2 typically use popcount parameters for which $\ell(k, n) \in (2/C_d(\pi/3), 4/C_d(\pi/3))$. Here we assume that list-size preserving parameterisations take $N = \ell(k, n)$. Note that $\eta(k, n) = 1 - \Pr[\hat{P}_{k,n} \mid \hat{R}_\theta]$ when the search is over a set of points uniformly distributed on the sphere, and $\eta(k, n) = 1 - \Pr[\hat{P}_{k,n} \mid \hat{R}_\theta \wedge \hat{B}_\beta]$ when the search is over a set of points uniformly distributed in a cap of angle β (left implicit).

In each of the quantum analyses, we apply Proposition 5.4.2 with $\gamma = 1$, $P = |g|$ and $Q = 1$ to estimate $q_0(m)$. We assume that filtered quantum search succeeds with probability 1 instead of probability at least 1/14, as guaranteed by Proposition 5.4.2. In practice, one will not know $|g|$ and one will therefore take $\gamma > 1$. Our use of $\gamma = 1$ is a systematic underestimate of the true cost of the search. There may be searches where our lower bound of $Q = 1$ on $|f \cap g|$ is too pessimistic. However, the probability of success in filtered quantum search decreases quadratically with $Q/|f \cap g|$ if $Q > |f \cap g|$. In Sections 5.7.1 and 5.7.3 we expect $|f \cap g| \approx 2$ so the effect of taking $Q = 1$ is negligible. In Section 5.7.2, where Q may be larger, an optimistic analysis using the expected value of Q makes negligible savings in dimension 512 and small savings in dimension 1024. This analysis does not decrement Q when a neighbour is found in, then removed from, a search space and ignores the quadratic decrease in success probability.

5.7.1 AllPairSearch

As a warmup, we optimise AllPairSearch. Asymptotically its complexity is $2^{(0.415\ldots+o(1))d}$ classically and $2^{(0.311\ldots+o(1))d}$ quantumly. We describe implementations of Line 5 of Algorithm 10 based on filtered search and filtered quantum search, and optimise `popcount` relative to these implementations.

Filtered search

Suppose that Line 5 applies $\texttt{popcount}_{k,n}(\mathbf{v}_i, \cdot)$ to each of $\mathbf{v}_{i+1}$ through $\mathbf{v}_N$ and then applies an inner product test to each vector that passes. With an input list of size $N = \ell(k, n)$, we expect this implementation to test all $\binom{N}{2}$ pairs before finding N neighbouring pairs. Moreover, we expect the `popcount` filter to identify $\binom{N}{2} \cdot \Pr[\hat{P}_{k,n}]$ potential neighbours, and to perform an equal number of inner product tests. The optimal parameters are obtained by minimising

$$\Big(c_1 \cdot \Pr[\hat{P}_{k,n}] + c_2(k, n)\Big) \cdot \binom{\ell(k, n)}{2}. \tag{5.7}$$

Filtered quantum search

Suppose that Line 5 is implemented using the search routine Algorithm 11. Specifically, we take the predicate f to be $\theta(\mathbf{v}_i, \cdot) \leq \theta$ with domain L_i. We take the filter g to be $\texttt{popcount}_{k,n}(\mathbf{v}_i, \cdot)$. Each call to the search routine returns at most one neighbour of $\mathbf{v}_i$. To find all detectable neighbours of $\mathbf{v}_i$ in L_i we must repeat the search $|f \cap g|$ times.

This is expected to be $|L_i| \cdot \Pr[\hat{P}_{k,n} \wedge \hat{R}_\theta]$. Known neighbours of $\mathbf{v}_i$ can be removed from L_i to avoid a coupon collector scenario. We consider an implementation in which searches are repeated until a search fails to find a neighbour of $\mathbf{v}_i$.

We expect to call the search subroutine $|L_i| \cdot \Pr[\hat{P}_{k,n} \wedge \hat{R}_\theta] + 1$ times in iteration i. Proposition 5.4.2 with $P = |L_i| \cdot \Pr[\hat{P}_{k,n}]$, $Q = 1$, and $\gamma = 1$ gives $q_0(|L_i|) = \frac{1}{2}\sqrt{|L_i|}$ iterations of $\mathbf{G}(g)$. As i ranges from 1 to $N-1$ the quantity $|L_i|$ takes each value in $\{1, \ldots, N-1\}$. Our proposed implementation therefore performs an expected

$$\sum_{j=1}^{N-1} \frac{1}{2}\sqrt{j}\left(j \cdot \Pr[\hat{P}_{k,n} \wedge \hat{R}_\theta] + 1\right) \\ = \Pr[\hat{P}_{k,n} \wedge \hat{R}_\theta]\left(\frac{1}{5}N^{5/2} + \frac{1}{4}N^{3/2}\right) + \frac{1}{3}N^{3/2} + O(\sqrt{N}) \quad (5.8)$$

applications of $\mathbf{G}(g)$; the expansion is obtained by the Euler–Maclaurin formula. When $N = \ell(k,n)$ we expect $N \cdot \Pr[\hat{P}_{k,n} \wedge \hat{R}_\theta] = 2 + O(1/N)$. The right hand side of (5.8) is then $\frac{11}{15}N^{3/2} + O(\sqrt{N})$.

Proposition 5.4.2 also provides an estimate for the rate at which reflections about the true positives, $\mathbf{R}_{f\cap g}$ are performed. With P and Q as above, we find that $\mathbf{R}_{f\cap g}$ is performed at roughly $p(k,n) = \sqrt{\Pr[\hat{P}_{k,n}]}$ the rate of calls to $\mathbf{G}(g)$. The optimal `popcount` parameters (up to some small error due to the $O(\sqrt{N})$ term in (5.8)) are obtained by minimising the total cost

$$\frac{11}{15}\left(q_1 p(k,n) + q_2(k,n)\right) \cdot \ell(k,n)^{3/2}. \quad (5.9)$$

5.7.2 RandomBucketSearch

One can improve AllPairSearch by *bucketing* the search space such that vectors in the same bucket are more likely to be neighbours [Laa15]. For example, one could pick a hemisphere H and divide the list into $L_1 = L \cap H$ and $L_2 = L \backslash L_1$. These lists would be approximately half the size of the original and the combined cost of AllPairSearch within L_1 and then within L_2 would be half the cost of an AllPairSearch within L. However, this strategy would fail to detect the expected θ/π fraction of neighbours that lie in opposite hemispheres.

Becker, Gama, and Joux [BGJ15] present a very efficient generalisation of this strategy. They propose bucketing the input list into subsets of the form $\{\mathbf{v} \in L : \texttt{popcount}_{k,n}(\mathbf{0}, \mathbf{v}; h) = 0\}$ with varying choices of h. This bucketing strategy is applied

recursively until the buckets are of a minimum size. Neighbouring pairs are then found by applying AllPairSearch.

A variant of the Becker–Gama–Joux algorithm that uses buckets of the form $L \cap C^{d-1}(f, \theta_1)$, with randomly chosen f and fixed θ_1, was proposed and implemented in [ADH+19a]. This variant is sometimes called `bgj1`. Here we call it RandomBucketSearch. This algorithm has asymptotic complexity $2^{(0.349\ldots+o(1))d}$ classically [ADH+19a] and $2^{(0.301\ldots+o(1))d}$ quantumly.[3] This is worse than the Becker–Gama–Joux algorithm, but RandomBucketSearch is conceptually simple and still provides an enormous improvement over AllPairSearch. Pseudocode is presented in Algorithm 12.

Algorithm 12 RandomBucketSearch.

Require: A list $L = (\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_N) \subset S^{d-1}$ of N points. Parameters $\theta, \theta_1 \in (0, \pi/2)$ and $t \in \mathbb{Z}_+$.

Ensure: A list of pairs $(\mathbf{u}, \mathbf{v}) \in L \times L$ with $\theta(\mathbf{u}, \mathbf{v}) \leq \theta$.

1: **function** RandomBucketSearch($L; \theta, \theta_1, t$)
2: $L' \leftarrow \emptyset$
3: **for** $1 \leq i \leq t$ **do**
4: Sample f uniformly on S^{d-1}
5: $L_f \leftarrow L \cap C^{d-1}(f, \theta_1)$
6: **for** j such that $\mathbf{v}_j \in L_f$ **do**
7: $L_{f,j} \leftarrow \{\mathbf{v}_k \in L_f : j < k \leq N\}$
8: Search $L_{f,j}$ for any number of $\mathbf{u}$ that satisfy $\theta(\mathbf{v}_j, u) \leq \theta$
9: For each such u found, add $(\mathbf{v}_j, \mathbf{u})$ to L'.
10: If $|L'| \geq N$, **return** L'.
11: **return** L'

Description of Algorithm 12

The algorithm takes as input a list of N points uniformly distributed on the sphere. A random bucket centre f is drawn uniformly from S^{d-1} in each of the t iterations of the outer loop. The choice of f defines a bucket in Line 5, $L_f = L \cap C^{d-1}(f, \theta_1)$, which is of expected size $N \cdot C_d(\theta_1)$. For each $\mathbf{v}_j \in L_f$, the inner loop searches a set $L_{f,j} \subset L_f$ for neighbours of $\mathbf{v}_j$. The quantity $|L_{f,j}|$ takes each value in $\{1, \ldots, |L_f| - 1\}$ as $\mathbf{v}_j$ ranges over L_f. The inner loop is identical to the loop in AllPairSearch apart from indexing and the fact that elements of L_f are known to be in the cap $C^{d-1}(f, \theta_1)$.

[3] The asymptotic quantum complexity is calculated, similarly to the classical complexity [ADH+19a], using the asymptotic value of $W_d(\theta, \theta_1, \theta_1)$ given in [BDGL16]. Let $N = 1/C_d(\pi/3)$ and $t(\theta_1) = 1/W_d(\pi/3, \theta_1, \theta_1)$. The exponent $0.3013\ldots$ is obtained by minimising $t(\theta_1)\left(N + (NC_d(\theta_1))^{3/2}\right)$ with respect to θ_1.

A bucket L_f is expected to contain $\binom{N}{2} \cdot \Pr[\hat{R}_\theta \wedge \hat{B}_{f,\theta_1}]$ neighbouring pairs. Only a $1-\eta(k,n)$ fraction of these are expected to be identified by the popcount filter. When $\theta_1 > \theta$ it is reasonable to assume that $\Pr[\hat{R}_\theta \wedge \hat{B}_{f,\theta_1}] \approx C_d(\theta) \cdot W_d(\theta, \theta_1, \theta_1)$. We use this approximation. The expected number of neighbouring pairs in L_f that are detected by the popcount filter is therefore approximately $\binom{N}{2} \cdot (1-\eta(k,n)) \cdot C_d(\theta) \cdot W_d(\theta, \theta_1, \theta_1)$. When $N = \ell(k,n)$ this is $N \cdot W_d(\theta, \theta_1, \theta_1)$. If all detectable neighbours are found by the search routine then the algorithm is list-size preserving when $N = \ell(k,n)$ and $t = 1/W_d(\theta, \theta_1, \theta_1)$.

We can now derive optimal popcount parameters for various implementations of Line 8.

Filtered search

Suppose that Line 8 of Algorithm 12 applies $\texttt{popcount}_{k,n}(\mathbf{v}_j, \cdot)$ to each element of $L_{f,j}$ and then applies an inner product test to each vector that passes. This implementation applies popcount tests to all $\binom{|L_f|}{2} \approx \binom{N \cdot C_d(\theta_1)}{2}$ pairs of elements in L_f and finds all of the neighbouring pairs that pass. In the process it applies inner product tests to a $p(\theta_1, k, n) = \Pr[\hat{P}_{k,n} \mid \hat{B}_{f,\theta_1}]$ fraction of pairs. The cost of populating buckets in one iteration of Line 5 is $c_1 \cdot \ell(k,n)$. The cost of all searches on Line 8 is $(c_1 \cdot p(\theta_1, k, n) + c_2(k,n)) \cdot \binom{NC_d(\theta_1)}{2}$. With the list-size preserving parameters N and t given above, the optimal θ_1, k, and n can be obtained by minimising the total cost

$$\frac{c_1 \cdot \ell(k,n) + (c_1 \cdot p(\theta_1, k, n) + c_2(k,n)) \cdot \binom{\ell(k,n) \cdot C_d(\theta_1)}{2}}{W_d(\theta, \theta_1, \theta_1)}. \tag{5.10}$$

Filtered quantum search

Suppose that Line 8 is implemented using the search routine Algorithm 11. We take the predicate f to be $\theta(\mathbf{v}_j, \cdot) \leq \theta$ with domain $L_{f,j}$. We take the filter g to be $\texttt{popcount}_{k,n}(\mathbf{v}_j, \cdot)$. Each call to the search routine returns at most one neighbour of $\mathbf{v}_j$. To find all detectable neighbours of $\mathbf{v}_j$ in $L_{f,j}$ we must repeat the search several times. Known neighbours of $\mathbf{v}_j$ can be removed from $L_{f,j}$ to avoid a coupon collector scenario. Proposition 5.4.2 with $P = |L_{f,j}| \cdot \Pr[\hat{P}_{k,n} \mid \hat{B}_{f,\theta_1}]$, $Q = 1$, and $\gamma = 1$ gives us that the number of $\mathbf{G}(g)$ iterations in a search on a set of size $|L_{f,j}|$ is $q_0(|L_{f,j}|) = \frac{1}{2}\sqrt{|L_{f,j}|}$.

We consider an implementation of Line 8 in which searches are repeated until a search fails to find a neighbour of $\mathbf{v}_j$. With $N = \ell(k,n)$, the set L_f is of expected size $\ell(k,n) \cdot C_d(\theta_1)$ and contains an expected $\ell(k,n) \cdot W_d(\theta, \theta_1, \theta_1)$ neighbouring pairs detectable by popcount. The set $L_{f,j}$ is expected to contain a proportional fraction

of these pairs. As such, we expect to call the search subroutine $|L_{f,j}| \cdot r(\theta_1, k, n) + 1$ times in iteration j where

$$r(\theta_1, k, n) = \frac{N \cdot W_d(\theta, \theta_1, \theta_1)}{\binom{|L_f|}{2}} \approx \frac{2\, W_d(\theta, \theta_1, \theta_1)}{\ell(k,n) \cdot C_d(\theta_1)^2}.$$

The inner loop makes an expected

$$\sum_{j=1}^{|L_f|-1} \frac{1}{2}\sqrt{j}\,(j \cdot r(\theta_1, k, n) + 1)$$

applications of $\mathbf{G}(g)$. This admits an asymptotic expansion similar to that of (5.8). If we assume that $|L_f|$ takes its expected value of $\ell(k,n) \cdot C_d(\theta_1)$, then the inner loop makes

$$q_3(\theta_1, k, n) \cdot (\ell(k,n) \cdot C_d(\theta_1))^{3/2}$$

applications of $\mathbf{G}(g)$, where

$$q_3(\theta_1, k, n) = \frac{2\, W_d(\theta, \theta_1, \theta_1)}{5\, C_d(\theta_1)} + \frac{1}{3}.$$

Proposition 5.4.2 also provides an estimate for the rate at which reflections about the true positives, $\mathbf{R}_{f \cap g}$ are performed. With P and Q as above, we find that $\mathbf{R}_{f\cap g}$ is applied at roughly $p(\theta_1, k, n) = \sqrt{\Pr[\hat{P}_{k,n} \mid \hat{B}_{f,\theta_1}]}$ the rate of $\mathbf{G}(g)$ iterations. The total cost of searching for neighbouring pairs in L_f is therefore

$$s(\theta_1, k, n) = (q_1 \cdot p(\theta_1, k, n) + q_2(k,n)) \cdot q_3(\theta_1, k, n) \cdot \left(\ell(k,n) \cdot C_d(\theta_1)\right)^{3/2}. \tag{5.11}$$

Populating L_f has a cost of $c_1 \cdot \ell(k,n)$. With the list-size preserving t given above, the optimal parameters θ_1, k, and n can be obtained by minimising the total cost

$$\frac{c_1 \cdot \ell(k,n) + s(\theta_1, k, n)}{W_d(\theta, \theta_1, \theta_1)}. \tag{5.12}$$

5.7.3 ListDecodingSearch

The optimal choice of θ_1 in RandomBucketSearch balances the cost of $N \cdot t$ cap membership tests against the cost of all calls to the search subroutine. It can be seen that reducing the cost of populating the buckets would allow us to choose a smaller θ_1, which would reduce the cost of searching within each bucket.

Algorithm 13, ListDecodingSearch, is due to Becker, Ducas, Gama, and Laarhoven [BDGL16]. Its complexity is $2^{(0.292...+o(1))d}$ classically and $2^{(0.265...+o(1))d}$ quantumly [Laa16, LMv13]. Like RandomBucketSearch, it computes a large number of list-cap intersections. However, these list-cap intersections involve a structured list – the list-cap intersections in RandomBucketSearch involve the inherently unstructured input list.

Algorithm 13 ListDecodingSearch.

Require: A list $L = (\mathbf{v}_1, \mathbf{v}_2, \dots \mathbf{v}_N) \subset S^{d-1}$ of N. Parameters $\theta, \theta_1, \theta_2 \in (0, \pi/2)$ and $t \in \mathbb{Z}_+$.

Ensure: A list of pairs $(\mathbf{u}, \mathbf{v}) \in L \times L$ with $\theta(\mathbf{u}, \mathbf{v}) \leq \theta$.

function ListDecodingSearch($L; \theta, \theta_1, \theta_2, t$)
  Sample a random product code F of size t
  Initialise an empty list L_f for each $f \in F$
  for $1 \leq i \leq N$ **do**
    $F_i \leftarrow F \cap C^{d-1}(\mathbf{v}_i, \theta_2)$
    Add $\mathbf{v}_i$ to L_f for each f in F_i
  for $1 \leq j < N$ **do**
    $F_j \leftarrow F \cap C^{d-1}(\mathbf{v}_j, \theta_1)$
    for $f \in F_j$ **do**
      $L_{f,j} \leftarrow \{\mathbf{v}_k \in L_f : j < k \leq N\}$
    $L_{F,j} \leftarrow \coprod_{f \in F_j} L_{f,j}$ (disjoint union)
    Search $L_{F,j}$ for any number of $\mathbf{u}$ that satisfy $\theta(\mathbf{v}_j, \mathbf{u}) \leq \theta$
    For each such $\mathbf{u}$ found, add $(\mathbf{v}_j, \mathbf{u})$ to L'.
    If $|L'| \geq N$, **return** L'.
  return L'

Description of Algorithm 13

The algorithm first samples a t point *random product code* F. See [BDGL16] for background on random product codes. In our analysis, we treat F as a list of uniformly random points on S^{d-1}. A formal statement is given as [BDGL16, Theorem 5.1], showing that such a heuristic is essentially true, up to a subexponential loss on the probability of finding the intend pairs.

The first loop populates t buckets that have as centres the points f of F. Bucket L_f stores elements of L that lie in the cap of angle θ_2 about f. Each bucket is of expected size $N \cdot C_d(\theta_2)$.

The second loop iterates over $\mathbf{v}_j \in L$ and searches for neighbours of $\mathbf{v}_j$ in the disjoint union of buckets with centres within an angle θ_1 of $\mathbf{v}_j$. The set F_j constructed on Line 8 contains an expected $t \cdot C_d(\theta_1)$ bucket centres. The disjoint union of certain elements from the corresponding buckets, denoted $L_{F,j}$, is of expected size $(N-j) \cdot C_d(\theta_2) \cdot t \cdot C_d(\theta_1)$. We note that by simplifying and assuming the expected size of $L_{F,j}$ is $N \cdot C_d(\theta_2) \cdot t \cdot C_d(\theta_1)$ the costs given below are never wrong by more than a factor of two.

Suppose that $\mathbf{w}$ is a neighbour of $\mathbf{v}_j$, so $\theta(\mathbf{v}_j, \mathbf{w}) \leq \theta$. The measure of the wedge formed by a cap of angle θ_1 about $\mathbf{v}_j$ and a cap of angle θ_2 about $\mathbf{w}$ is at least $W_d(\theta, \theta_1, \theta_2)$. Assuming that the points of a random product code are indistinguishable from points sampled uniformly on the sphere, we expect at least $t \cdot W_d(\theta, \theta_1, \theta_2)$ of the $f \in F_j$ to contain $\mathbf{w}$.

The second loop is executed N times. Iteration j searches $L_{F,j}$ for neighbours of $\mathbf{v}_j$. With $N = \ell(k, n)$ there are expected to be N detectable neighbouring pairs in L. With $t = 1/W_d(\theta, \theta_1, \theta_2)$ we expect that each neighbouring pair is of the form $(\mathbf{v}_j, \mathbf{w})$ with $\mathbf{w} \in L_{F,j}$.

The angles θ_1, θ_2 relate to the spherical cap parameters α, β respectively in [BDGL16], and are such that $\theta_1 \geq \theta_2$. Optimal time complexity is achieved when $\theta_1 = \theta_2$, the setting we use in our estimates.

We have omitted the list decoding mechanism by which list-cap intersections are computed. In our analysis we assume that the cost of a list-cap intersection such as $F_i = F \cap C^{d-1}(\mathbf{v}_i, \theta_2)$ is proportional to $|F_i|$, but independent of $|F|$, i.e. we are in the 'efficient list-decodability regime' of [BDGL16, Section 5.1] and may take their parameter $m = \log d$. In particular, we assume that in the cost of $O(\log(d) \cdot |F_i|)$ inner products and $|F|^{O(1/\log(d))}$ other operations, as stated in [BDGL16, Lemma 5.1], the first cost dominates. In [BDGL16] these costs relate to $O(m \cdot M \cdot \mathcal{C}_n(\alpha))$ and $O(nB + mB \log B)$ respectively. We therefore assume the cost of forming $F_i = F \cap C^{d-1}(\mathbf{v}_i, \theta_2)$ is $\log(d) \cdot |F_i|$ inner product tests.

Filtered search

Suppose that the implementation of Line 12 of Algorithm 13 applies $\texttt{popcount}_{k,n}(\mathbf{v}_j, \cdot)$ to each element of $L_{F,j}$ and then applies an inner product test to each vector that passes. This implementation applies `popcount` tests to all $N \cdot C_d(\theta_2) \cdot t \cdot C_d(\theta_1)$ elements of $L_{F,j}$ and finds all of the neighbours of $\mathbf{v}_j$ that pass. Note that $\mathbf{w} \in L_{F,j}$ implies that

there exists some $f \in F$ such that both $\mathbf{v}_j$ and $\mathbf{w}$ lie in a cap of angle θ_1 around f. Inner product tests are applied to a $p(\theta_1, k, n) \geq \Pr[\hat{P}_{k,n} \mid \hat{B}_{f,\theta_1}]$ fraction of all pairs.[4]

The cost of preparing all t buckets in the first loop is $c_1 \cdot N \cdot t \cdot C_d(\theta_2)$. The cost of constructing the search spaces in the second loop is $c_1 \cdot N \cdot t \cdot C_d(\theta_1)$. Each search has a cost of $|L_{F,j}|$ popcount tests and $|L_{F,j}| \cdot p(\theta_1, k, n)$ inner product tests. With the list-size preserving parameterisation given above, the optimal θ_1, θ_2, k, and n can be obtained by minimising the total cost

$$\frac{\ell(k,n)}{W_d(\theta,\theta_1,\theta_2)}\Big(c_1 \cdot C_d(\theta_1) + c_1 \cdot C_d(\theta_2) + \Big(c_1 \cdot p(\theta_1,k,n) + c_2(k,n)\Big) \cdot \ell(k,n) \cdot C_d(\theta_1) \cdot C_d(\theta_2)\Big). \quad (5.13)$$

Filtered quantum search

Suppose that Line 12 is implemented using Algorithm 11. We take the predicate f to be $\theta(\mathbf{v}_j, \cdot) \leq \theta$ with domain $L_{F,j}$. We take the filter g to be $\texttt{popcount}_{k,n}(\mathbf{v}_j, \cdot)$. Each call to the search routine returns at most one neighbour of $\mathbf{v}_j$. Known neighbours of $\mathbf{v}_j$ can be removed from $L_{F,j}$ to avoid a coupon collector scenario. Proposition 5.4.2 with $P = |L_{F,j}| \cdot \Pr[\hat{P}_{k,n} \mid \hat{B}_{f,\theta_1}]$, $Q = 1$, and $\gamma = 1$ gives us that the number of $\mathbf{G}(g)$ iterations in a search on a set of size $|L_{F,j}|$ is $q_0(|L_{F,j}|) \approx \frac{1}{2}\sqrt{|L_{F,j}|}$.

Assuming that computing $F_j = F \cap C(\mathbf{v}_j, \theta_1)$ has a cost of $c_1 |F_j|$, the N iterations of Lines 5 and 8 have a total cost of

$$c_1 \cdot N \cdot t \cdot (C_d(\theta_1) + C_d(\theta_2)) \quad (5.14)$$

Each search applies an expected

$$q_0(|L_{F,j}|) \approx \frac{1}{2}\sqrt{N \cdot C_d(\theta_1) \cdot t \cdot C_d(\theta_2)}$$

applications of $\mathbf{G}(g)$. Reflections about the true positives, $\mathbf{R}_{f \cap g}$, are performed at roughly $p(\theta_1, k, n) = \sqrt{\Pr[\hat{P}_{k,n} \mid B_{f,\theta_1}]}$ the rate of $\mathbf{G}(g)$ iterations. We consider an implementation of Line 8 in which searches are repeated until a search fails to find a neighbour of $\mathbf{v}_j$. With the list-size preserving parameters given above, we expect to perform two filtered quantum searches per iteration of the second loop. The optimal

[4] The inequality is because $\mathbf{w}$ may be contained in multiple buckets, $L_{f,j}$.

parameters can be obtained by minimising the total cost

$$\ell(k,n)\left(c_1\frac{C_d(\theta_1)+C_d(\theta_2)}{W_d(\theta,\theta_1,\theta_2)}+(q_1p(\theta_1,k,n)+q_2(k,n))\sqrt{\frac{\ell(k,n)C_d(\theta_1)C_d(\theta_2)}{W_d(\theta,\theta_1,\theta_2)}}\right).$$

5.8 Cost estimates

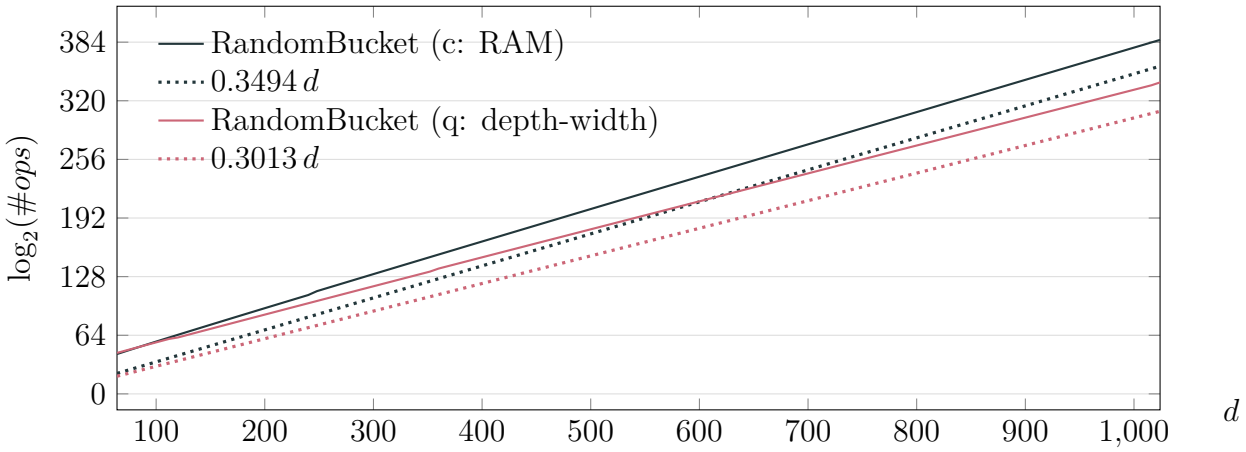
Our software numerically optimises the cost functions in Sections 5.7.1, 5.7.2 and 5.7.3 with respect to several classical and quantum cost metrics. The classical cost metrics that we consider are: *c (unit cost)*, which assigns unit cost to `popcount`; *c (RAM)*, which uses the classical circuits of Section 5.5. The quantum cost metrics that we consider are: *q (unit cost)*, which assigns unit cost to a Grover iteration; *q (depth-width)*, which assigns unit cost to every gate (including the identity) in the quantum circuits of Section 5.5; *q (gates)*, which assigns unit cost only to the non identity gates; *q (T count)*, which assigns unit cost only to T gates; and *q (GE19)*, which is described in Section 5.8.1.

We stress that our software and Figure 5.2 give *estimates* for the cost of each algorithm. A similar figure for AllPairSearch can be found in [AGPS19, App. D]. These estimates are neither upper bounds nor lower bounds. As we mention above, we have systematically omitted and underestimated some costs. For instance, we have omitted the list decoding mechanism in our costing of Algorithm 13. We have approximated other costs. For instance, the cost that we assign to an inner product in Section 5.5.3. We have also not explored the entire optimisation space. We only consider values of the popcount parameter n that are one less than a power of two. Moreover, following the discussion of `popcount` parameters in Section 2.5.4, given that we want $k/n \approx \theta/\pi$ and we have $\theta = \pi/3$, we set $k = \lfloor n/3 \rfloor$.

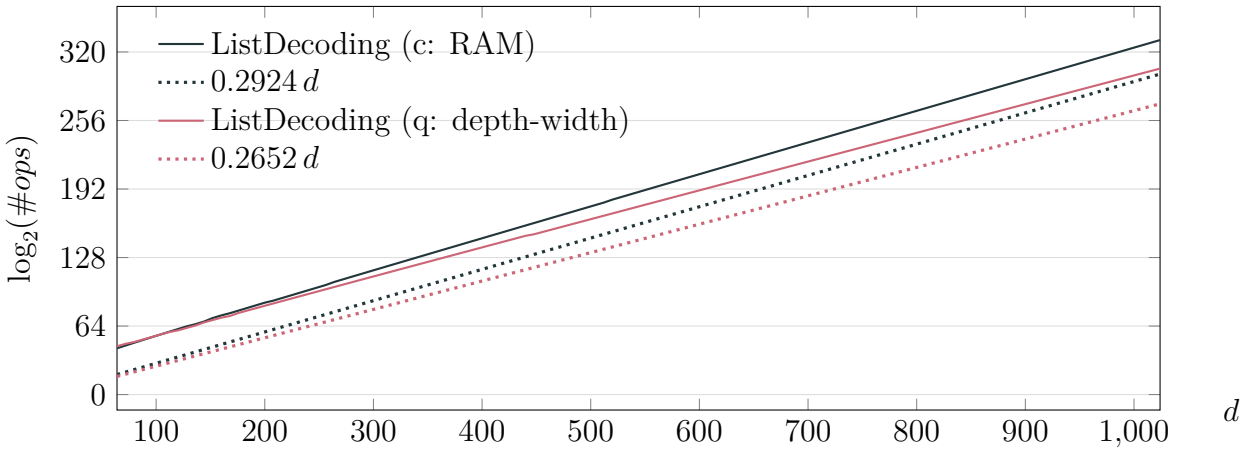
While we have omitted and approximated some costs, we have tried to ensure that these omissions and approximations will ultimately lead our software to *underestimate* of the total cost of the algorithm. For instance, if our inner product cost is accurate, our optimisation procedure ensures that we satisfy Remark 5.4.2 and can ignore costs relating to $\mathbf{R}_{f\cap g}$.

Our results are presented in Figure 5.2. We also plot the leading term of the asymptotic complexity of the respective algorithms as these are routinely referred to in the literature. The source code, and raw data for all considered cost metrics, is available at `https://github.com/jschanck/eprint-2019-1161`.

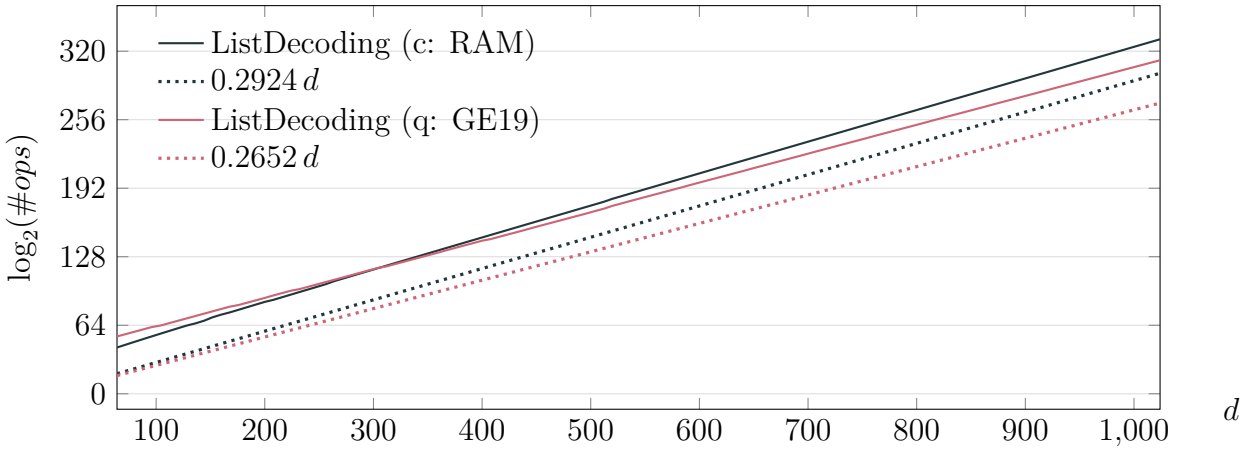
Figure 5.2 Quantum (‘q’) and classical (‘c’) resource estimates for NNS search.



RandomBucketSearch. Comparing c: (RAM) with q: (depth-width), and the leading terms of the asymptotic complexities.



ListDecodingSearch. Comparing c: (RAM) with q: (depth-width), and the leading terms of the asymptotic complexities.



ListDecodingSearch. Comparing c: (RAM) with q: (GE19), and the leading terms of the asymptotic complexities.

5.8.1 Barriers to a quantum advantage

As expected, our results in Figure 5.2 indicate that quantum search provides a substantial savings over classical search asymptotically. Our plots fully contain the range of costs from 2^{128} to 2^{256} that are commonly thought to be cryptanalytically interesting. Modest cost improvements are attained in this range.

The range of parameters in which a sieve could conceivably be run, however, is much narrower. If one assumes a memory density of one petabyte per gram (2^{53} bits per gram), a 2^{140} bit memory would have a mass comparable with that of the Moon. Supposing that a 2-sieve stores $1/C_d(\pi/3)$ vectors, and that each vector is $\log_2(d)$ bits, an adversary with a 2^{140} bit memory could only run a sieve in dimension 608 or lower. The potential cost improvement in dimension 608 is smaller than the potential cost improvement in, say, dimension 1000. The potential cost improvement that can be actualised is likely smaller still.

We expect that our cost estimates are underestimates. However, the quantum advantage could grow, shrink, or even be eliminated if our underestimates do not affect quantum and classical costs equally. In this section, we list several reasons to think that the advantage might shrink or disappear.

Error correction overhead

By using the depth-width metric for quantum circuits, we assume that dispatching a logical gate to a logical qubit costs one RAM instruction. In practice, however, the cost depends on the error correcting code that is used for logical qubits. This cost may be significant.

Gidney and Ekerå have estimated the resources required to factor a 2048 bit RSA modulus using Shor's algorithm on a surface code based quantum computer [GE21]. Under a plausible assumption on the physical qubit error rate, they calculate that a factoring circuit with $2^{12.6}$ logical qubits and depth 2^{31} requires a distance $\delta = 27$ surface code. Each logical qubit is encoded in $2\,\delta^2 = 1458$ physical qubits, and the error tracking routine applies at least $\delta^2 = 729$ bit instructions, per logical qubit per layer of logical circuit depth, to read its input.

In general, a circuit of depth D and width W requires a distance $\delta = \Theta(\log(DW))$ surface code. To perform a single logical gate, classical control hardware dispatches several instructions to each of the $\Theta(\log^2(DW))$ physical qubits. The classical control hardware also performs a non trivial error tracking routine between logical gates, which

takes measurement results from half of the physical qubits as input.[5] Consequently, the cost of surface code computation grows like $\Omega(DW \log^2(DW))$.

We have adapted scripts provided by Gidney and Ekerå to estimate δ for our circuits. The last plot of Figure 5.2 shows the cost of ListDecodingSearch when every logical gate (including the identity) is assigned a cost of δ^2. For ListDecodingSearch the cost in the Gidney–Ekerå metric grows from 2^{128} to 2^{256} between dimensions 352 and 824, and we calculate a 2^{128} bit memory is sufficient to run in dimension 544. We find that the advantage of quantum search over classical search is a factor of $2^{1.8}$ in dimension 352, a factor of $2^{7.1}$ in dimension 544, and a factor of $2^{14.4}$ in dimension 824. Compare this with the naïve estimate for the advantage, $2^{0.292d-0.265d}$, which is a factor of $2^{9.5}$ in dimension 352, a factor of $2^{14.7}$ in dimension 544, and a factor of $2^{22.5}$ in dimension 824.

One should also note that error correction for the surface code sets a natural clock speed, which Gidney and Ekerå estimate at one cycle per microsecond. Gidney and Ekerå estimate that their factoring circuit, the cost of which is dominated by a single modular exponentiation, would take 7.44 hours to run. This additional overhead in terms of time is not refelected in the instruction count.

On the positive side, the cost estimate used in Figure 5.2 is specific to the surface code architecture. Significant improvements may be possible. Gottesman has shown that an overhead of $\Theta(1)$ physical qubits per logical qubit is theoretically possible [Got14]. Whether this technique offers lower overhead than the surface code in practice is yet to be seen.

Dependence on qRAM

Quantum accessible classical memories are used in many quantum algorithms. For example, they are used in black box search algorithms [Gro96], in collision finding algorithms [BHT97], and in some algorithms for the dihedral hidden subgroup problem [Kup13]. The use of qRAM is not without controversy [Ber09, GR04]. Previous work on quantum lattice sieve algorithms [Laa16, LMv13] has noted that constructing practical qRAM seems challenging.

Morally, looking up an ℓ bit value in a table with 2^n entries should have a cost that grows at least with $n + \ell$. Recent results [AGJO$^+$15, BGB$^+$18, MGM20] indicate that realistic implementations of qRAM have costs that grow much more quickly than this. When ancillary qubits are kept to a minimum, the best known Clifford+T

[5] For a thorough introduction to how logical gates are performed on the surface code see [FMMC12], and for more advanced techniques see e.g. [HFDM12].

implementation of a qRAM has a **T** count of $4 \cdot (2^n - 1)$ [BGB+18]. While it is conceivable that a qRAM could be constructed at lower cost on a different architecture, as has been suggested in [GLM08], a unit cost qRAM gate should be seen as a powerful, and potentially unrealistic, resource.

One can argue that classical RAMs also have a large cost. This is not to say that classical and quantum RAMs have the same cost. A qRAM can be used to construct an arbitrary superposition over the elements of a memory. This process relies on quantum interference and necessarily takes as long as a worst case memory access time. This is in contrast with classical RAM, where careful programming and attention to a computer's caches can mask the fact that accessing an N bit memory laid out in a 3-dimensional space necessarily takes $\Omega(N^{1/3})$ time.

If the cost of a qRAM gate is equivalent to $\Theta(N^{1/3})$ Clifford+T gates, then the asymptotic cost of quantum AllPair search is $2^{(0.380\ldots+o(1))d}$, the asymptotic cost of quantum RandomBucket search is $2^{(0.336\ldots+o(1))d}$, and the asymptotic cost of quantum ListDecoding search is $2^{(0.284\ldots+o(1))d}$. If memory is constrained to two dimensions, and qRAM costs $\Theta(N^{1/2})$ Clifford+T gates, the quantum asymptotics match the classical RAM asymptotics.

Quantum sampling routines

We have assumed that **D** in Section 5.5.1 (the uniform sampling subroutine in Grover's algorithm) is implemented using parallel **H** gates. This is the smallest possible circuit that might implement **D**, and may be a significant underestimate. In Line 12 of Algorithm 13 we must construct a superposition (ideally uniform) over $\{k : \mathbf{v}_k \in L_{F,j}\}$. The set $L_{F,j}$ is presented as a disjoint union of smaller sets. Copying the elements of these smaller sets to a flat array would be more expensive than our estimate for the cost of search. While we do not expect the cost of sampling near uniformly from $L_{F,j}$ to be large, it could easily exceed the cost of `popcount`.

5.8.2 Relevance to SVP

The NNS algorithms that we have analysed are closely related to lattice sieves for SVP. While the asymptotic cost of NNS algorithms are often used as a proxy for the asymptotic cost of solving SVP, we caution the reader against making this comparison in a non asymptotic setting. On the one hand, our estimates might lead one to underestimate the cost of solving SVP:

- the costs given in Figure 5.2 represent one iteration of NNS within a sieve, while sieve algorithms make poly(d) iterations;
- the costs given in Figure 5.2 do not account for all of the subroutines within each NNS algorithm.

On the other hand, our estimates might lead one to overestimate the cost of solving SVP:

- it is a mistake to conflate the cost of NNS in dimension d with the cost of SVP in dimension d. The 'dimensions for free' technique of [Duc18a] can be used to solve SVP in dimension d by calling an NNS routine polynomially many times in dimension $d' < d$. Our analysis seamlessly applies to dimension d';
- there are heuristics that exploit structure present in applications to SVP not captured in our general setting, e.g. the vector space structure allowing both $\pm\mathbf{u}$ to be tested for the cost of $\mathbf{u}$, and keeping the vectors sorted by length.

5.9 Conclusion

Since the publication of this work the software we provide, and in particular the gate count estimates for our sieving routines, have been incorporated into the analysis of several finalists and alternative finalists of the NIST post quantum cryptography standardisation process. In particular [SAB+20, DKR+20, NAB+20] use the classical gate count for the [BDGL16] sieve to provide a secondary security analysis outside of the 'CoreSVP' model [ADPS16, Sec. 6.1]. In the CoreSVP model BKZ-β reduction is assumed to require a single β dimensional SVP oracle call, and this oracle call is assumed to cost $2^{0.292d}$ in the classical case, using the best classical complexity achieved by [BDGL16]. In the call for proposals NIST suggest that submissions are compared to AES [NIS17, Sec. 4.A.5]. In particular, for the lowest security level they require that submissions be as hard as key search on AES-128. They estimate that this procedure costs 2^{143} classical gates, and the gate counts given in this chapter allow a careful estimation of BKZ based attacks on lattice submissions, see e.g. [SAB+20, Sec. 5.2].

The classical and quantum gate counts given in this work have also been used alongside the simulation based LWE estimator,[6] an implementation of [DDGR20], in [HKP+21]. The authors of this work describe post quantum continuous group key

[6] https://github.com/lducas/leaky-LWE-Estimator/

agreement protocols that give out many LWE samples and are parametrised to meet NIST Level I security.

The sieving techniques considered here are not exhaustive. While it would be relatively easy to adapt our software to other 2-sieves, like the cross polytope sieve [BL16], future work might consider k-sieves such as [BLS16, KMPM19a].

Future work might also address the barriers to a quantum advantage discussed in Section 5.8.1. Two additional barriers are worth mentioning here. First, as Grover search does not parallelise well, one might consider depth restrictions for classical and quantum circuits. Second, our estimates might be refined by including some of the classical subroutines, present in both the classical and quantum variants of the same sieve, that we have ignored, e.g. the cost of sampling lattice vectors or the cost of list-decoding in Algorithm 13. Any cost increase will reduce the range of cryptanalytically relevant dimensions, giving fewer dimensions to overcome quantum overheads.

Finally, our estimates should be checked against experiments. Our analysis of Algorithm 12 recommends a database of size $N(d) \approx 2/C_d(\pi/3)$, while the largest sieving experiments to date [ADH+19a] runs Algorithm 12 with a database of size $N'(d) = 3.2 \cdot 2^{0.2075d}$ up to dimension $d = 127$. There is a factor of 8 gap between $N'(127)$ and $N(127)$. A factor of two can be explained by the fact that [ADH+19a] treats each database entry $\mathbf{u}$ as $\pm\mathbf{u}$. It is possible that the remaining factor of four can be explained by the other heuristics used in [ADH+19a]. As d increases, $N(d)$ and $N'(d)$ continue to diverge, so future work could attempt to determine more accurately the required list size.

Bibliography

[ABF+20] Martin R. Albrecht, Shi Bai, Pierre-Alain Fouque, Paul Kirchner, Damien Stehlé, and Weiqiang Wen. Faster enumeration-based lattice reduction: Root hermite factor $k^{1/(2k)}$ time $k^{k/8+o(k)}$. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 186–212. Springer, Heidelberg, August 2020.

[ABLR21] Martin R. Albrecht, Shi Bai, Jianwei Li, and Joe Rowell. Lattice reduction with approximate enumeration oracles. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, LNCS, pages 732–759, Cham, 2021. Springer.

[ACD+18] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the LWE, NTRU schemes! In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 351–367. Springer, Heidelberg, September 2018.

[ACF+15] Martin R. Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Algebraic algorithms for lwe problems. *ACM Commun. Comput. Algebra*, 49(2):62, August 2015.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, Heidelberg, August 2009.

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *29th ACM STOC*, pages 284–293. ACM Press, May 1997.

[ADH+19a] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 717–746. Springer, Heidelberg, May 2019.

[ADH+19b] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. Cryptology ePrint Archive, Report 2019/089, 2019. `https://eprint.iacr.org/2019/089`.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete Gaussian sampling: Extended abstract. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 733–742. ACM Press, June 2015.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011, Part I*, volume 6755 of *LNCS*, pages 403–415. Springer, Heidelberg, July 2011.

[AGJO+15] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum ram. *New Journal of Physics*, 17(12):123010, 2015.

[AGPS19] Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. Cryptology ePrint Archive, Report 2019/1161, 2019. `https://eprint.iacr.org/2019/1161`.

[AGPS20] Martin R. Albrecht, Vlad Gheorghiu, Eamonn W. Postlethwaite, and John M. Schanck. Estimating quantum speedups for lattice sieves. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 583–613. Springer, Heidelberg, December 2020.

[AGVW17] Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 297–322. Springer, Heidelberg, December 2017.

[AH21] Martin R. Albrecht and Nadia Heninger. On bounded distance decoding with predicate: Breaking the "lattice barrier" for the hidden number problem. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 528–558, Cham, 2021. Springer.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *33rd ACM STOC*, pages 601–610. ACM Press, July 2001.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of Mathematics*, 160(2):781–793, 2004.

[Alb17] Martin R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 103–129. Springer, Heidelberg, April / May 2017.

[ALSD21] Divesh Aggarwal, Zeyong Li, and Noah Stephens-Davidowitz. A $2^{n/2}$-time algorithm for $\sqrt{n}$-svp and $\sqrt{n}$-hermite svp, and an improved time-approximation tradeoff for (h)svp. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, LNCS, pages 467–497, Cham, 2021. Springer.

[AMG+16] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John M. Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In Roberto Avanzi and Howard M. Heys, editors, *SAC 2016*, volume 10532 of *LNCS*, pages 317–337. Springer, Heidelberg, August 2016.

[AMMR13] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, 2013.

[ANS18] Yoshinori Aono, Phong Q. Nguyen, and Yixin Shen. Quantum lattice enumeration and tweaking discrete pruning. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 405–434. Springer, Heidelberg, December 2018.

[ASD18] Divesh Aggarwal and Noah Stephens-Davidowitz. Just Take the Average! An Embarrassingly Simple 2^n-Time Algorithm for SVP (and CVP). In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms (SOSA 2018)*, volume 61 of *OpenAccess Series in Informatics (OASIcs)*, pages 12:1–12:19, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[AWHT16] Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 789–819. Springer, Heidelberg, May 2016.

[Bab86] L. Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[Bak18] Joanne Baker. Forgotten heroes of the enigma story. `https://www.nature.com/articles/d41586-018-06149-y`, 2018. Accessed: 2021-08-04.

[BBG+13] Robert Beals, Stephen Brierley, Oliver Gray, Aram W. Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. Efficient distributed quantum computing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 469(2153):20120686, 2013.

[BBG+16] Johannes Buchmann, Niklas Büscher, Florian Göpfert, Stefan Katzenbeisser, Juliane Krämer, Daniele Micciancio, Sander Siim, Christine van Vredendaal, and Michael Walter. Creating cryptographic challenges using multi-party computation: The lwe challenge. In *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*, AsiaPKC '16, pages 11–20, New York, NY, USA, 2016. Association for Computing Machinery.

[BBHT98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4–5):493–505, 1998.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016.

[BdWD+01] Harry Buhrman, Ronald de Wolf, Christoph Dürr, Mark Heiligman, Peter H"yer, Frédéric Magniez, and Miklos Santha. Quantum algorithms for element distinctness. In *Proceedings of the 16th Annual Conference on Computational Complexity*, CCC '01, page 131, USA, 2001. IEEE Computer Society.

[Ben82] Paul Benioff. Quantum mechanical hamiltonian models of turing machines. *Journal of Statistical Physics*, 29(3):515–546, 1982.

[Ber09] Dan J. Bernstein. Cost analysis of hash collisions: Will quantum computers make sharcs obsolete? *Workshop Record of SHARCS'09: Special-purpose Hardware for Attacking Cryptographic Systems*, 2009.

[BG14a] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 28–47. Springer, Heidelberg, February 2014.

[BG14b] Shi Bai and Steven D. Galbraith. Lattice decoding attacks on binary LWE. In Willy Susilo and Yi Mu, editors, *ACISP 14*, volume 8544 of *LNCS*, pages 322–337. Springer, Heidelberg, July 2014.

[BGB+18] Ryan Babbush, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear T complexity. *Phys. Rev. X*, 8:041015, Oct 2018.

[BGJ14] Anja Becker, Nicolas Gama, and Antoine Joux. A sieve algorithm based on overlattices. *LMS Journal of Computation and Mathematics*, 17(A):49–70, 2014.

[BGJ15] Anja Becker, Nicolas Gama, and Antoine Joux. Speeding-up lattice sieving without increasing the memory, using sub-quadratic nearest neighbor search. Cryptology ePrint Archive, Report 2015/522, 2015. `https://eprint.iacr.org/2015/522`.

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum computation and information (Washington, DC, 2000)*, volume 305 of *Contemp. Math.*, pages 53–74. Amer. Math. Soc., Providence, RI, 2002.

[BHT97] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. *SIGACT News*, 28(2):14–19, June 1997.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, July 2003.

[BL06] Johannes Buchmann and Christoph Ludwig. Practical lattice basis sampling reduction. In Florian Hess, Sebastian Pauli, and Michael Pohst, editors, *Algorithmic Number Theory*, pages 222–237, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BL16] Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 3–23. Springer, Heidelberg, April 2016.

[Bli29] H. F. Blichfeldt. The minimum value of quadratic forms, and the closest packing of spheres. *Mathematische Annalen*, 101(1):605–608, 1929.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013.

[BLS16] Shi Bai, Thijs Laarhoven, and Damien Stehlé. Tuple lattice sieving. *LMS Journal of Computation and Mathematics*, 19(A):146–162, 2016.

[BMW19] Shi Bai, Shaun Miller, and Weiqiang Wen. A refined analysis of the cost for solving LWE via uSVP. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi, editors, *AFRICACRYPT 19*, volume 11627 of *LNCS*, pages 181–205. Springer, Heidelberg, July 2019.

[BSW16] Shi Bai, Damien Stehlé, and Weiqiang Wen. Improved reduction from the bounded distance decoding problem to the unique shortest vector problem in lattices. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016*, volume 55 of *LIPIcs*, pages 76:1–76:12. Schloss Dagstuhl, July 2016.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[Cas97] J.W.S. Cassels. *An Introduction to the Geometry of Numbers*. Springer-Verlag Berlin Heidelberg, 1st edition, 1997.

[CCL18] Yanlin Chen, Kai-Min Chung, and Ching-Yi Lai. Space-efficient classical and quantum algorithms for the shortest vector problem. *Quantum Info. Comput.*, 18(3–4):283–305, March 2018.

[CDKM04] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit, 2004. arXiv:quant-ph/0410184.

[CDW17] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short stickelberger class relations and application to ideal-SVP. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 324–348. Springer, Heidelberg, April / May 2017.

[Cha02] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *34th ACM STOC*, pages 380–388. ACM Press, May 2002.

[Che13] Yuanmi Chen. *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe*. PhD thesis, Paris 7, 2013. Thèse de doctorat dirigée par Nguyen, Phong-Quang Informatique Paris 7 2013.

[CHKP12] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *Journal of Cryptology*, 25(4):601–639, October 2012.

[CK04] Henry Cohn and Abhinav Kumar. The densest lattice in twenty-four dimensions. *Electron. Res. Announc. Amer. Math. Soc.*, 10:58–67, 2004.

[CL21] André Chailloux and Johanna Loyer. Lattice sieving via quantum random walks. Cryptology ePrint Archive, Report 2021/570, 2021. `https://ia.cr/2021/570`.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2011.

[CS99] John Conway and Neil J.A. Sloane. *Sphere Packings, Lattices and Groups*. Springer, New York, NY, 3rd edition, 1999.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358. Springer, Heidelberg, August 2020.

[Deu85] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[Die82] D. Dieks. Communication by epr devices. *Physics Letters A*, 92(6):271–272, 1982.

[DKR+20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

[DRSD14] Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. On the closest vector problem with a distance guarantee. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 98–109, 2014.

[DSvW21] Léo Ducas, Marc Stevens, and Wessel P. J. van Woerden. Advanced lattice sieving on gpus, with tensor cores. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 249–279, Cham, 2021. Springer.

[dt21a] The FPLLL development team. fplll, a lattice reduction library, Version: 5.4.1. Available at `https://github.com/fplll/fplll`, 2021.

[dt21b] The FPLLL development team. fpylll, a Python wraper for the fplll lattice reduction library, Version: 0.5.6. Available at `https://github.com/fplll/fpylll`, 2021.

[Duc18a] Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 125–145. Springer, Heidelberg, April / May 2018.

[Duc18b] Léo Ducas. Shortest Vector from Lattice Sieving: a Few Dimensions for Free (talk). `https://eurocrypt.iacr.org/2018/Slides/Monday/TrackB/01-01.pdf`, April 2018. Accessed: 2021-03-10.

[FBB+15] Robert Fitzpatrick, Christian H. Bischof, Johannes Buchmann, Özgür Dagdelen, Florian Göpfert, Artur Mariano, and Bo-Yin Yang. Tuning GaussSieve for speed. In Diego F. Aranha and Alfred Menezes, editors, *LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 288–305. Springer, Heidelberg, September 2015.

[Fey82] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1982.

[FK15] Masaharu Fukase and Kenji Kashiwabara. An accelerated algorithm for solving svp based on statistical analysis. *Journal of Information Processing*, 23(1):67–80, 2015.

[FMMC12] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012.

[FP85] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–463, May 1985.

[Gal12] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[Gal14] Francois Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 216–225, 2014.

[GE21] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, April 2021.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 112–131. Springer, Heidelberg, August 1997.

[GJMS17] Qian Guo, Thomas Johansson, Erik Mårtensson, and Paul Stankovski. Coded-BKW with sieving. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 323–346. Springer, Heidelberg, December 2017.

[GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008.

[GLRS16] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying grover's algorithm to AES: Quantum resource estimates. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 29–43. Springer, Heidelberg, 2016.

[GM03] Daniel Goldstein and Andrew Mayer. On the equidistribution of hecke points. *Forum Math.*, 15(2):165–189, 2003.

[GMSS99] O. Goldreich, D. Micciancio, S. Safra, and J. P. Seifert. Approximating shortest lattice vectors is not harder than approximating closet lattice vectors. *Inf. Process. Lett.*, 71(2):55—61, July 1999.

[GN08a] Nicolas Gama and Phong Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 207–216. ACM Press, May 2008.

[GN08b] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 31–51. Springer, Heidelberg, April 2008.

[GN17] François Gall and Shogo Nakajima. Quantum algorithm for triangle finding in sparse graphs. *Algorithmica*, 79(3):941–959, November 2017.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 257–278. Springer, Heidelberg, May / June 2010.

[Got14] Daniel Gottesman. Fault-tolerant quantum computation with constant overhead. *Quantum Info. Comput.*, 14(15–16):1338—1372, November 2014.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[GR04] Lov Grover and Terry Rudolph. How significant are the known collision and element distinctness quantum algorithms. *Quantum Info. Comput.*, 4:201–206, May 2004.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th ACM STOC*, pages 212–219. ACM Press, May 1996.

[GS10] Nicolas Gama and Michael Schneider. Svp challenge. `https://www.latticechallenge.org/svp-challenge/`, 2010. Accessed: 2021-03-10.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.

[GY15] Florian Göpfert and Ashwinkumar Yakkundimath. Tu darmstadt learning with errors challenge. `https://www.latticechallenge.org/lwe_challenge/challenge.php`, 2015. Accessed: 2021-03-10.

[HA28] D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik*. Springer-Verlag Berlin Heidelberg, 1928.

[Han06] Juncheol Han. The general linear group over a ring. *Bulletin of the Korean Mathematical Society*, 43(3):619–626, 08 2006.

[Hen02] Martin Henk. Successive minima and lattice points. *Rend. Circ. Mat. Palermo, Suppl.*, 70:377–384, 2002.

[HFDM12] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, December 2012.

[HK17] Gottfried Herold and Elena Kirshanova. Improved algorithms for the approximate k-list problem in euclidean norm. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 16–40. Springer, Heidelberg, March 2017.

[HKL18] Gottfried Herold, Elena Kirshanova, and Thijs Laarhoven. Speed-ups and time-memory trade-offs for tuple lattice sieving. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 407–436. Springer, Heidelberg, March 2018.

[HKP+21] Keitaro Hashimoto, Shuichi Katsumata, Eamonn Postlethwaite, Thomas Prest, and Bas Westerbaan. A concrete treatment of efficient continuous group key agreement via multi-recipient pkes. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1441–1462, November 2021.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 447–464. Springer, Heidelberg, August 2011.

[HR07] Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 469–477. ACM Press, June 2007.

[HRS17] Thomas Häner, Martin Roetteler, and Krysta M. Svore. Factoring using 2n + 2 qubits with toffoli based modular multiplication. *Quantum Info. Comput.*, 17(7-8):673–684, June 2017.

[HS07] Guillaume Hanrot and Damien Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 170–186. Springer, Heidelberg, August 2007.

[JJP18] Matthew Jenssen, Felix Joos, and Will Perkins. On kissing numbers and spherical codes in high dimensions. *Advances in Mathematics*, 335:307–321, 2018.

[JNRV20] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. Implementing grover oracles for quantum key search on AES and LowMC. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 280–310. Springer, Heidelberg, May 2020.

[JS19] Samuel Jaques and John M. Schanck. Quantum cryptanalysis in the RAM model: Claw-finding attacks on SIKE. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 32–61. Springer, Heidelberg, August 2019.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *15th ACM STOC*, pages 193–206. ACM Press, April 1983.

[Kan87] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[Kir16] Paul Kirchner. Re: Sieving vs. enumeration. `https://groups.google.com/forum/#!msg/cryptanalytic-algorithms/BoSRL0uHIjM/wAkZQlwRAgAJ`„ May 2016. Accessed: 2021-03-10.

[KL78] G.A. Kabatiansky and V.I. Levenshtein. On bounds for packings on a sphere and in space. *Probl. Peredachi Inf.*, 14(1):3–25, 1978.

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In David B. Shmoys, editor, *11th SODA*, pages 937–941. ACM-SIAM, January 2000.

[KLM07] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 2007.

[KMPM19a] Elena Kirshanova, Erik Mårtensson, Eamonn W. Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 521–551. Springer, Heidelberg, December 2019.

[KMPM19b] Elena Kirshanova, Erik Mårtensson, Eamonn W. Postlethwaite, and Subhayan Roy Moulik. Quantum algorithms for the approximate k-list problem and their application to lattice sieving. Cryptology ePrint Archive, Report 2019/1016, 2019. `https://eprint.iacr.org/2019/1016`.

[Knu97] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., USA, 1997.

[Koe14] Wolfram Koepf. *Hypergeometric Summation: An Algorithmic Approach to Summation and Special Function Identities*. Springer-Verlag London, 2nd edition, 2014.

[Kri19] Kris Kwiatkowski and Luke Valenta. The tls post-quantum experiment. `https://blog.cloudflare.com/the-tls-post-quantum-experiment/`, 2019. Accessed: 2021-08-03.

[Kup13] Greg Kuperberg. Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. In Simone Severini and Fernando Brandao, editors, *8th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2013)*, volume 22 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20–34, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Laa15] Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2015.

[Laa16] Thijs Laarhoven. *Search problems in cryptography: from fingerprinting to lattice sieving*. PhD thesis, Mathematics and Computer Science, February 2016. Proefschrift.

[Laa17] Thijs Laarhoven. Hypercube LSH for Approximate near Neighbors. In Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:20, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[LdW15] Thijs Laarhoven and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 101–118. Springer, Heidelberg, August 2015.

[LL93] Arjen K. Lenstra and Hendrik W. Jr. Lenstra. *The Development of the Number Field Sieve*. Springer-Verlag Berlin Heidelberg, 1993.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LLS90] J. C. Lagarias, H. W. Lenstra, and C. P. Schnorr. Korkin-zolotarev bases and successive minima of a lattice and its reciprocal lattice. *Combinatorica*, 10(4):333–348, 1990.

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 577–594. Springer, Heidelberg, August 2009.

[LM18] Thijs Laarhoven and Artur Mariano. Progressive lattice sieving. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 292–311. Springer, Heidelberg, 2018.

[LMv13] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Solving the shortest vector problem in lattices faster using quantum search. In Philippe Gaborit, editor, *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*, pages 83–101. Springer, Heidelberg, June 2013.

[LN13] Mingjie Liu and Phong Q. Nguyen. Solving BDD by enumeration: An update. In Ed Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 293–309. Springer, Heidelberg, February / March 2013.

[LN20] Jianwei Li and Phong Q. Nguyen. A complete analysis of the BKZ lattice reduction algorithm. Cryptology ePrint Archive, Report 2020/1237, 2020. `https://eprint.iacr.org/2020/1237`.

[Lov86] László Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity*. Society for Industrial and Applied Mathematics, 1986.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 319–339. Springer, Heidelberg, February 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, June 2015.

[Map] Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario. Standard Worksheet Interface, Maple 2016.0, February 17 2016.

[Mar03] Jacques Martinet. *Perfect Lattices in Euclidean Spaces*. Springer-Verlag Berlin Heidelberg, 2003.

[Mas16] Dmitri Maslov. Advantages of using relative-phase toffoli gates with an application to multiple control toffoli optimization. *Physical Review A*, 93(2):022311, 2016.

[Mat16] Matt Braithwaite. Experimenting with post-quantum cryptography. `https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html`, 2016. Accessed: 2021-08-03.

[MGM20] Olivia Di Matteo, Vlad Gheorghiu, and Michele Mosca. Fault-tolerant resource estimation of quantum random-access memories. *IEEE Transactions on Quantum Engineering*, 1:1–13, 2020.

[Mic11] Daniele Micciancio. *The Geometry of Lattice Cryptography*, pages 185–210. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[Min91] Hermann Minkowski. Ueber die positiven quadratischen formen und über kettenbruchähnliche algorithmen. *Journal für die reine und angewandte Mathematik*, 107:278–297, 1891.

[Min96] H. Minkowski. *Geometrie der Zahlen*. Teubner, Leipzig-Berlin, 1st edition, 1896.

[Mon18] Ashley Montanaro. Quantum-walk speedup of backtracking algorithms. *Theory of Computing*, 14(15):1–24, 2018.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

[MR09] Daniele Micciancio and Oded Regev. *Lattice-based Cryptography*, pages 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Mus08] Oleg R. Musin. The kissing number in four dimensions. *Annals of Mathematics*, 168(1):1–32, 2008.

[MV10a] Manfred G. Madritsch and Brigitte Vallée. Modelling the LLL algorithm by sandpiles. In Alejandro López-Ortiz, editor, *LATIN 2010*, volume 6034 of *LNCS*, pages 267–281. Springer, Heidelberg, April 2010.

[MV10b] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In Leonard J. Schulman, editor, *42nd ACM STOC*, pages 351–358. ACM Press, June 2010.

[MV10c] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In Moses Charika, editor, *21st SODA*, pages 1468–1480. ACM-SIAM, January 2010.

[MW15] Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In Piotr Indyk, editor, *26th SODA*, pages 276–294. ACM-SIAM, January 2015.

[NAB+20] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

[NC11] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011.

[Ngu10] Phong Q. Nguyen. *The LLL Algorithm: Survey and Applications*, chapter Hermite's Constant and Lattice Algorithms, pages 19–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[NIS17] NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`, 2017. Accessed: 2021-08-03.

[NS06] Phong Q. Nguyen and Damien Stehlé. Lll on the average. In Florian Hess, Sebastian Pauli, and Michael Pohst, editors, *Algorithmic Number Theory*, pages 238–256, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[NS09] Phong Q. Nguyen and Damien Stehlé. Low-dimensional lattice basis reduction revisited. *ACM Trans. Algorithms*, 5(4), November 2009.

[NV08] Phong Q. Nguyen and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *J. Mathematical Cryptology*, 2(2):181–207, 2008.

[OtR85] A.M. Odlyzko and H.J.J. te Riele. Disproof of the mertens conjecture. *Journal für die reine und angewandte Mathematik*, 1985(357):138–160, 1985.

[PAA+17] Thomas Poppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, and Douglas Stebila. NewHope. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[Par09] Behrooz Parhami. Efficient hamming weight comparators for binary vectors based on accumulative and up/down parallel counters. *IEEE Trans. on Circuits and Systems*, 56-II(2):167–171, 2009.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 333–342. ACM Press, May / June 2009.

[PHS19] Alice Pellet-Mary, Guillaume Hanrot, and Damien Stehlé. Approx-SVP in ideal lattices with pre-processing. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 685–716. Springer, Heidelberg, May 2019.

[PV21] Eamonn W. Postlethwaite and Fernando Virdia. On the success probability of solving unique SVP via BKZ. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 68–98. Springer, Heidelberg, May 2021.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Heidelberg, August 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Reg09a] Oded Regev. Lecture notes: Lattices in computer science. `http://www.cims.nyu.edu/~regev/teaching/lattices_fall_2009/index.html`, 2009. Accessed: 19-07-2021.

[Reg09b] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), September 2009.

[Reg10] Oded Regev. *On the Complexity of Lattice Problems with Polynomial Approximation Factors*, pages 475–496. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[SAB+20] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

[Sch03] Claus Peter Schnorr. Lattice reduction by random sampling and birthday methods. In Helmut Alt and Michel Habib, editors, *STACS 2003*, pages 145–156, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[SE91] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In L. Budach, editor, *Fundamentals of Computation Theory*, pages 68–85, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[SE94] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1):181–199, Aug 1994.

[Sem01] Igor Semaev. A 3-dimensional lattice reduction algorithm. In *Revised Papers from the International Conference on Cryptography and Lattices*, CaLC '01, pages 181–193, Berlin, Heidelberg, 2001. Springer-Verlag.

[Sho94] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[Sie89] C.L. Siegel. *Lectures on the Geometry of Numbers*. Springer-Verlag Berlin Heidelberg, 1st edition, 1989.

[SS77] R. Solovay and V. Strassen. A fast monte-carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1977.

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 617–635. Springer, Heidelberg, December 2009.

[Ste10] Damien Stehlé. *The LLL Algorithm: Survey and Applications*, chapter Floating-Point LLL: Theoretical and Practical Aspects, pages 179–213. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Tam76] P. P. Tammela. The hermite-minkowski domain of reduction of positive definite quadratic forms in six variables. *Journal of Soviet Mathematics*, 6(6):677–688, 1976.

[Tea17] The CADO-NFS Development Team. CADO-NFS, an implementation of the number field sieve algorithm, 2017. Release 2.3.0.

[TKH18] Tadanori Teruya, Kenji Kashiwabara, and Goichiro Hanaoka. Fast lattice basis reduction suitable for massive parallelization and its application to the shortest vector problem. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 437–460. Springer, Heidelberg, March 2018.

[Tur37] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 01 1937.

[Tur18] Dermot Turing. *X, Y & Z: The Real Story of How Enigma Was Broken*. The History Press, 2018.

[Vir21] Fernando Virdia. *Post-Quantum Cryptography: Cryptanalysis and Implementation*. PhD thesis, Royal Holloway, University of London, Egham, Surrey, UK, March 2021.

[Vou11] Panagiotis Voulgaris. Gauss sieve – alpha release. `https://cseweb.ucsd.edu/~pvoulgar/impl.html`, 2011. Accessed: 2021-06-28.

[Wal14] Michael Walter. Sage implementation of Chen and Nguyen's BKZ simulator. `https://pub.ist.ac.at/~mwalter/src/sim_bkz.sage`, 2014. Accessed: 2021-03-24.

[WCW19] Jinming Wen, Xiao-Wen Chang, and Jian Weng. Improved upper bounds on the hermite and kz constants. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1742–1746, 2019.

[Why05] William Whyte. *Ntru*, pages 427–430. Springer US, Boston, MA, 2005.

[WZ82] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.

[YD17] Yang Yu and Léo Ducas. Second order statistical behavior of LLL and BKZ. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017*, volume 10719 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2017.

[Zim20] Paul Zimmermann. Factorization of rsa-250. `https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;dc42ccd1.2002`, 2020. Accessed: 2021-08-03.