Search problems in cryptography

Finding colluders in fingerprinting and finding nearby vectors in lattice sieving

Thijs Laarhoven

November 10, 2015

Thanks

To be completed.

Contents

Introduction

This dissertation consists of two parts. First each part will be described separately, and then common ground in the two parts will be discussed.

Part 1: Finding colluders in fingerprinting

Collusion-resistant fingerprinting is a technique to protect copyrighted digital content against piracy. Fingerprints are added to the content that uniquely link different copies to different users, so that pirates who leak or share their copy with others can be traced and dealt with. Defending against single pirates is relatively easy, but when many pirates *collude* and mix their unique copies to form a pirate copy with a mixed fingerprint, it becomes much harder to track down who was responsible for making this copy. With so-called *collusion-resistant fingerprinting codes*, we can make sure that even such advanced pirate attacks will not allow the traitors to get away with their actions.

Part 1 of this thesis focuses on theoretical improvements to collusion-resistant fingerprinting, that may have a practical impact as well. Can we improve existing schemes even further? Can we design other schemes which work even better against large collusion attacks? And can these schemes be deployed efficiently in practice? Ultimately, the goal of the first part of this thesis is to obtain a better understanding of fingerprinting codes, connecting the fingerprinting problem to various well-studied problems and solutions from other fields of research, and to find improvements to existing schemes that may be useful in practice as well. This is done over the course of seven chapters as follows.

- **Chapter 1** describes preliminaries on fingerprinting, explaining and motivating the problem, describing the mathematical model that we consider, and describing notation that we will use throughout the first part. This chapter also states research questions which we aim to answer in the subsequent chapters.

- **Chapter 2** examines the best known scheme before this work, and whether it can be improved without modifying the decoder. This is based on joint work with Benne de Weger, published at *IH&MMSec 2013* [LdW13].

- **Chapter 3** discusses slightly simplified attack models, analyzing how hard it is to solve these easier problems, and what these results tell us about the general model. These results were published in the proceedings of *IH&MMSec 2014* [Laa14] and in the *IEEE Transactions on Information Forensics and Security* [Laa15b].

- **Chapter 4** looks at practical schemes for the same simplified models, and how these results can be applied in the general fingerprinting problem as well, to obtain a better performance than with the schemes considered in Chapter 2. These results were published in the proceedings of *IH&MMSec 2014* [Laa14] and a journal version is currently under review [Laa15a].

- **Chapter 5** considers a variant of the standard model, where the pirates output their mixed pirate copy in real-time and the tracer may use this information to adjust the scheme and trace the pirates faster. These results appeared at *IH&MMSec 2015* [Laa15c], with parts coming from papers published at *WIFS 2013* [Laa13a] and in the *IEEE Transactions on Information Theory* [LDR+13].

- **Chapter 6** discusses applications of these results outside fingerprinting, showing that these contributions also improve upon previous results in the field of group testing. These results appeared at *Allerton 2013* [Laa13b] and in various other papers mentioned above [Laa14, Laa15b, Laa15c].

- **Chapter 7** finally concludes the first part with an overview of answers to the research questions, the most important results, and open problems that still remain.

Other papers on this topic co-written by the author, which are not covered in this thesis, appeared at *WIFS 2012* [LOD12], *SPIE Electronic Imaging 2014* [ODL14], and in *Designs, Codes and Cryptography* [LdW14][1].

Part 2: Finding nearby vectors in lattice sieving

Lattice-based cryptography is a recent, popular line of research in cryptography, focusing on the use of lattices to design primitives that facilitate secure communication between two or more parties, in the presence of an adversary. Lattice-based cryptography is only secure if the underlying "hard problems" are computationally hard to solve, and one of these problems is the shortest vector problem: find a shortest non-zero vector in a high-dimensional lattice, given a description of this lattice. Currently *sieving* is the fastest known method for solving this problem in high dimensions, and improving sieving algorithms would immediately impact the estimated computational hardness of finding shortest vectors, and thus the estimated security of lattice-based cryptographic primitives.

Lattice sieving algorithms all follow the same principle: given the description of a lattice, we can easily generate many "long" lattice vectors, and given sufficiently many long lattice vectors, we can combine them to find shorter lattice vectors. By first generating a long list of lattice vectors, and then combining them appropriately, we ultimately hope to find a shortest lattice vector in our list as well. The computationally most intensive part of the algorithm consists in finding pairs of vectors which can be combined to obtain shorter lattice vectors. What is commonly done in practice now is looking at all pairs of vectors, and seeing if they can be combined. In many cases two vectors cannot be combined, but going through all pairs guarantees that we will find all good pairs.

Part 2 of this thesis considers more sophisticated and faster methods for finding pairs of vectors which can be combined in a useful way, which ultimately corresponds to faster methods for finding nearby vectors in high-dimensional spaces. The focus is on theoretical

[1] Both [LdW14] and [LDR+13] are the result of previous work done during the author's final Masters project.

improvements for high dimensions, which may be useful in practice as well. Can we find nearby vectors faster than with a naive linear search, using known techniques from the literature? Can we perhaps improve upon the literature on finding nearby vectors, so that we obtain even faster algorithms in high dimensions? And how do different methods compare in practice, when the dimension is not that high? The goal is to obtain a better understanding of sieving and its connection with other areas of research, and to find theoretical improvements to existing sieving algorithms which may be relevant in practice as well. This is done over the course of eight chapters as follows.

- **Chapter 8** describes preliminaries on lattices and lattice sieving, introducing basic sieving algorithms, and describing notation that will be used throughout the second part of this thesis. At the end of this chapter we state the research questions that are studied in the next chapters.

- **Chapter 9** considers the fastest sieving algorithms in high dimensions prior to this work, and studies whether a further modification to one of these methods will lead to even better results. This chapter is based on previously unpublished work.

- **Chapter 10** examines the combination of sieving with an efficient technique from the nearest-neighbor literature, leading to both theoretical and practical improvements in moderate and high dimensions. These results previously appeared in the proceedings of *Crypto 2015* [Laa15d].

- **Chapter 11** looks at a different, recent method from nearest-neighbor literature, and how this leads to further theoretical improvements in high dimensions when combined with sieving. This is based on joint work with Benne de Weger, and these results previously appeared at *Latincrypt 2015* [LdW15].

- **Chapter 12** studies a practical improvement over previous nearest-neighbor methods, and shows how these results apply to sieving on (ideal) lattices. This is based on joint work with Alexandr Andoni, Piotr Indyk, Ilya Razenshteyn, and Ludwig Schmidt, which appeared at *NIPS 2015* [AIL+15], and on joint work with Anja Becker, which is currently under review [BL15a].

- **Chapter 13** studies a new direction in high-density nearest-neighbor searching, leading to further theoretical and practical improvements over previous results. This is based on joint work with Anja Becker, Léo Ducas, and Nicolas Gama, and these results will appear in the proceedings of *SODA 2016* [BDGL16].

- **Chapter 14** investigates the effects of quantum algorithms on the computational complexity of lattice sieving, showing how security parameters should be adjusted to account for quantum attacks. This is based on joint work with Michele Mosca and Joop van de Pol, and these results appeared at *PQCrypto 2013* [LMvdP13] and in *Designs, Codes, and Cryptography* [LMvdP15].

- **Chapter 15** finally concludes with an overview of the results, answers to the research questions, and possible directions for future research.

Other papers on this topic co-written by the author, the contents of which are not covered in this thesis, appeared at *ICPP 2015* [MLB15] and as a preprint [LvdPdW12].

Common themes

Although the two topics of this thesis are very different and at first sight completely unrelated, there are some common themes which appear in both parts of the thesis.

Searching for special elements using circumstantial evidence. The most obvious similarity between these problems is that we are searching a large universe of many elements (legitimate users in fingerprinting, list vectors in lattice sieving) for a small subset of special elements (pirates in fingerprinting, nearby vectors in lattice sieving). In both cases, the proposed methods to solve these problems make use of *circumstantial*, indirect evidence rather than direct, hard evidence to conclude whether an element is special or not (accusation scores in fingerprinting, hash collisions in lattice sieving).

Randomized algorithms and average-case analyses. Closely related to the use of circumstantial evidence is the probabilistic nature of solutions proposed in both parts of the thesis. Many algorithmic problems become much easier if a correct solution only has to be found *with high probability*, rather than guaranteeing that the algorithm always outputs a correct solution, and this idea is also applied in both parts. Rather than focusing on the worst-case costs of these algorithms under worst-case inputs, we focus on average-case analyses, and allow for small errors to be made.

Divide and conquer. A general theme in various algorithmic applications is *divide and conquer*, where a large problem is partitioned into small subproblems (divide) which are then tackled individually (conquer). In fingerprinting this technique appears as the simple but strong *interleaving attack*, where the pirates mix their copies in divide-and-conquer fashion, but also in the score-based framework of dealing with each user and segment separately, and combining these results to solve the larger problem.[2] In the new lattice sieving techniques, this idea is prominently present in nearest-neighbor searching, as most of these methods are essentially a high-dimensional application of divide-and-conquer: partition the space in regions, and solve the problem of finding nearby vectors in each of these regions separately.

Connecting the dots. Sometimes completely new ideas must be invented to improve upon existing solutions, but in many cases combining existing techniques from different fields already leads to significant improvements. In the first part, solutions for fingerprinting are found by applying tools from most notably information theory and hypothesis testing literature to fingerprinting, and in the second part new solutions are obtained by combining lattice sieving with nearest-neighbor searching and quantum algorithms. We provide feedback to these areas as well, by improving upon state-of-the-art techniques in group testing (Chapter 6) and nearest-neighbor searching (Chapters 12 and 13).

Asymptotic analyses with practical applications. Finally, in both parts we focus on large-parameter asymptotics (large collusion sizes in fingerprinting, high dimensions in sieving), but we always keep in mind what are the practical applications of these techniques. Theoretical improvements which are only better in the asymptotic limit are not very useful, and so we also focus on the practicability of these results. In fingerprinting, we present explicit and efficiently-computable encoders and decoders, and in lattice sieving we support the claimed speed-ups with experimental results.

[2] Certain fingerprinting schemes make even more explicit use of this idea [FT01, LOD12].

PART I

FINDING COLLUDERS IN FINGERPRINTING

CHAPTER 1

Collusion-resistant fingerprinting codes

1.1 — Problem description

Digital content. Over the past few decades, there has been an enormous increase in the use of digital data. Think of music, videos, software; a whole movie can now be encoded by a long (virtual) string of bits, which can then be stored and played on a smartphone. To watch a movie, you do not even have to go to the store anymore to buy a copy; you can just connect to the internet, purchase a digital copy of the content from an authorized reseller, download it to your local machine, and play the file on your computer or TV. This is both convenient for the customer (who does not have to leave his house to buy a copy) and for the distributor of the content (who does not have to produce and sell a physical product (CD, DVD) with the content, and can just send the digital data over the internet). Ideally, these technologies benefit everyone.

Digital piracy. As digital data is much easier to reproduce and copy than say a car, the rise of digital content also raises new issues, such as digital piracy. A customer who has been granted access to watch a movie, e.g. by buying a DVD in the store or obtaining it through official online vendors, can easily distribute this movie to his family, friends, and others as well. The simplest way to do this would just be to send a digital copy of the same content to his friends, so that they can also play the video. With digital rights management (DRM), distributors of the content can make it harder to either copy the raw data or play the same data on two different machines, but even the most advanced technologies cannot prevent that a customer plays the movie on his TV and then records either directly from the TV, or uses an external device (a camera) to capture the movie n another medium. After all, the customer has to be able to watch the movie, so it cannot be prevented that an external source records what the user is seeing. So even with DRM techniques, it seems hard to prevent all types of piracy.

Watermarking. While preventing the recording and redistribution of digital content seems hard, if not impossible, not all is lost for the content owner yet. Each legitimate customer has to be able to watch the same movie, but that does not mean that each user has to watch the *exact same* copy of the movie. More precisely, it may be possible to embed hidden watermarks in the content, which are imperceptible to the eye, but which can be observed by a detector with knowledge of the locations and shapes of these watermarks. For instance, by making some pixels in some of the frames slightly darker on one copy and slightly lighter on another copy, the average user will not be able to tell that he is watching a watermarked copy, but a detector can sense whether this copy is the dark or

the light version. By embedding watermarks in the content that are unique to each user, and recovering a pirate copy of the content which has been distributed online, a tracer can then detect who is responsible for the piracy, and take action against these users.

Collusion attacks. With these watermarking techniques it may become harder for pirates to share content online without being caught, but the cat-and-mouse game does not end here. By obtaining multiple legitimate copies of the same content, or by cooperating with other users, a pirate may compare several differently watermarked copies of the same content, to find parts of the watermark. It may be hard to tell whether something has changed to the content if there is nothing to compare a copy to, but if a user has two different copies, then by simply looking at the raw data he can find parts of the watermark in the content. Pirates who have compared their copies and detected parts of the watermark where their copies differ may try to remove this watermark, or simply mix their copies; if each time the different copies of the content differ, the pirates randomly choose one of their copies to use for the pirate output, then the watermark in the resulting pirate copy will not match any of the pirates' watermarks. Using such collusion attacks, the pirates may not be caught, and they again win the game.

Collusion-resistant fingerprinting. Again, this is not the end of the story, as the distributor of the content can defend against collusion attacks. By carefully choosing which user is assigned which watermarked version of the content in which part of the content, each user may not only be assigned a unique copy of the content, but the fingerprints assigned to the users may also be resistant against such collusion attacks. Then, even a mixed copy of the content formed by a collusion attack may be traced back to one or more of the responsible users. For this we need *collusion-resistant fingerprinting codes*, describing which user receives which watermarked version in each segment, and a *tracing algorithm*, describing how a mixed fingerprint can be traced back to the guilty users. With collusion-resistant fingerprinting schemes, users may be deterred from piracy, and the content distributor may win the cat-and-mouse game after all.

1.2 — The bias-based framework

The above problem of fingerprinting can be modeled by the following two-person game between a distributor $\mathcal{D}$ and an adversary $\mathcal{C}$ (the set of *colluders*). First, there is a universe $\mathcal{U}$ of n users, and the adversary is assigned a random subset of users $\mathcal{C} \subseteq \mathcal{U}$ of size $|\mathcal{C}| = c$. This subset $\mathcal{C}$ is unknown to the distributor, although we commonly assume that the distributor does know (an upper bound on) c. The aim of the game for the distributor $\mathcal{D}$ is ultimately to discover $\mathcal{C}$ by gathering evidence from the pirate output. The two-person game consists of three phases: (1) the distributor uses an *encoder* to generate a fingerprinting code, used for assigning watermarked versions to users; (2) the colluders employ a *collusion channel* to generate the pirate output from their given code words; and (3) the distributor uses a *decoder* to map the pirate output to a set $\mathcal{C}' \subseteq \mathcal{U}$.

1.2.1 – Encoder. First, the distributor generates a fingerprinting code $\mathcal{X}$ of n binary[1] code words of length ℓ. The parameter ℓ is referred to as the *code length*, and the distributor would like ℓ to be as small as possible. For the eventual embedded watermark,

[1] In fingerprinting a common generalization is to assume that the entries of the code words come from an alphabet of size $q \geqslant 2$, but we will restrict our attention to the binary case $q = 2$.

we assume that for each segment of the content there are two different versions, and the watermark of user j is determined by the ℓ entries in the jth code word of $\mathcal{X}$.

A common restriction on the encoding process is to assume that $\mathcal{X}$ is created by first generating a bias vector $\mathbf{P} \in (0,1)^\ell$ (by choosing each entry P_i, for $i = 1, \dots, \ell$, independently from a certain distribution f_P), and then generating code words $\mathbf{X}_j \in \mathcal{X}$ according to $\mathbb{P}(X_{j,i} = 1) = P_i$. This guarantees that watermarks of different users j are independent, and that watermarks in different positions i are independent. Fingerprinting schemes that satisfy this assumption are sometimes called bias-based schemes, and the encoders considered here are also assumed to belong to this category. We denote the set of all possible encoders in our model by $\mathcal{P}^e$.

1.2.2 – Collusion channel. After generating $\mathcal{X}$, the code words are used to select and embed watermarks in the content, and the content is sent out to all users. The colluders then get together, compare their copies, and use a certain collusion channel or pirate attack $\boldsymbol{\Theta}$ to determine the pirate output $\mathbf{Y} \in \{0,1\}^\ell$. If the pirate attack behaves symmetrically both in the colluders and in the positions i, then the collusion channel can be modeled by a vector $\theta \in [0,1]^{c+1}$, consisting of entries $\theta_z = f_{Y|Z}(1|z) = \mathbb{P}(Y_i = 1|z)$ (for $z = 0, \dots, c$) indicating the probability of outputting a 1 when the pirates received z ones and $c - z$ zeroes. The set of possible pirate attacks θ is sometimes denoted by $\mathcal{P}^c$. A further restriction on θ in fingerprinting is the *marking assumption*, introduced by Boneh and Shaw [BS98], which says that $\theta_0 = 0$ and $\theta_c = 1$, i.e., if the pirates receive only zeros or ones they have to output this symbol. As an example, one might think of symbols corresponding to different decryption keys, in which case it may indeed be impossible to output a symbol (decryption key) which was not among the received symbols (decryption keys) of the colluders.

For concreteness, throughout the first part we will consider various pirate attacks which have been considered in the fingerprinting literature before:

- **Interleaving attack**: The coalition randomly selects one of its members and outputs his symbol. This corresponds to $(\theta_{\text{int}})_z = z/c$ for all z, so that indeed $(\theta_{\text{int}})_0 = 0$ and $(\theta_{\text{int}})_c = 1$. This attack is simple to execute for the pirates and at the same time known to be one of the hardest attacks to deal with, from the tracing point of view.
- **All-1 attack**: The pirates output a 1 whenever they can, i.e., whenever they have at least one 1. This translates to $(\theta_{\text{all1}})_z = 1\{z > 0\}$. As we will see in Chapter 6, this attack is of particular interest due to its relation with group testing.
- **Majority voting**: The colluders output the most common symbol among their received symbols. This means that $(\theta_{\text{maj}})_z = 1\{z > c/2\}$.
- **Minority voting**: The traitors output the symbol which they received the *least* often (but received at least once). For $1 \leqslant z \leqslant c-1$, this corresponds to $(\theta_{\text{min}})_z = 1\{z < c/2\}$, while by the marking assumption we have $(\theta_{\text{min}})_0 = 0$ and $(\theta_{\text{min}})_c = 1$.
- **Coin-flip attack**: If the pirates receive both symbols, they flip a (fair) coin to decide which symbol to output. For $1 \leqslant z \leqslant c-1$, this corresponds to $(\theta_{\text{coin}})_z = \frac{1}{2}$, while due to the marking assumption we have $(\theta_{\text{coin}})_0 = 0$ and $(\theta_{\text{coin}})_c = 1$.

Note that not all pirate attacks can be categorized in this model, as this assumes that the pirate output only depends on the pirate tallies, rather than the exact assignment of symbols to colluders. This symmetry among pirates may be a common assumption, but as noted in e.g. [Sch08] this assumption is not obvious. Among the collusion attacks that

are not captured in the above model, we highlight one in particular:

- **Scapegoat attack**: The coalition always selects the same colluder, and outputs his symbol. Although this makes finding this one colluder easy, it may guarantee that the other colluders are never in any danger of getting caught.

1.2.3 – Decoder. After the pirate output has been generated and distributed, we assume the distributor intercepts it and applies a decoding algorithm to $\boldsymbol{Y}$, $\mathcal{X}$ and $\mathbf{P}$ to compute a set $\mathcal{C}' \subseteq \mathcal{U}$ of accused users. The distributor wins the game if no innocent users are caught and colluders are caught, and loses if this is not the case. This definition is intentionally ambiguous, as it covers two different but closely related definitions of when the distributor wins the game:

- **Catch-all scenario**: The distributor $\mathcal{D}$ only wins the game if the estimated collusion $\mathcal{C}'$ is exactly equal to $\mathcal{C}$. Note that in the standard non-adaptive setting, this game can never be won if the pirates are allowed to use pirate-asymmetric attacks like the scapegoat attack.
- **Catch-one scenario**: The distributor $\mathcal{D}$ wins if $\mathcal{C}'$ contains at least one colluder and contains no innocent users. This game can be won by the distributor with high probability in any setting.

Unless stated otherwise, we will consider the catch-one scenario. Finally, regardless of the model, it is generally impossible to guarantee that the distributor wins the game with probability 1, and we therefore define two error probabilities as follows:

- ε_0: The **false-positive probability** of accidentally accusing one or more innocent users. In other words, this corresponds to the event $\mathcal{C}' \not\subseteq \mathcal{C}$.
- ε_1: The **false-negative probability** of not catching the colluders. In the catch-all setting this means $\mathcal{C} \not\subseteq \mathcal{C}'$ and in the catch-one setting this means $\mathcal{C}' \cap \mathcal{C} = \emptyset$.

In practice, what usually happens is that the parameters $c, n, \varepsilon_0, \varepsilon_1$ are specified, and one is tasked with designing a scheme that works against c colluders hidden among n total users with false-positive and false-negative error probabilities bounded from above by ε_0 and ε_1 respectively, while at the same time minimizing the code length ℓ required to provide these guarantees. Commonly one prefers to set $\varepsilon_0 \ll \varepsilon_1$, as making sure that innocent users are not harmed is more important than making sure that (all) guilty users are caught.

1.2.4 – Adaptivity. Finally, the differences between non-adaptive (static) fingerprinting, adaptive (dynamic) fingerprinting, and sequential fingerprinting can be explained by showing in which order the encoding, collusion and decoding phases take place. Denoting by $\mathrm{enc}_i, \mathrm{coll}_i, \mathrm{dec}_i$ the encoding, collusion and decoding phases corresponding to the ith segment of the content, we can order the phases as follows:

- Non-adaptive: $\mathrm{enc}_{[1,\ldots,\ell]}$; $\mathrm{coll}_{[1,\ldots,\ell]}$; $\mathrm{dec}_{[1,\ldots,\ell]}$.
- Sequential: $\mathrm{enc}_{[1,\ldots,\ell]}$; coll_1; dec_1; coll_2; dec_2; $\ldots$; coll_ℓ; dec_ℓ.
- Adaptive: enc_1; coll_1; dec_1; enc_2; coll_2; dec_2; $\ldots$; enc_ℓ; coll_ℓ; dec_ℓ.

In other words: in the adaptive setting the code can be adjusted and accusations can be made after every symbol; in sequential fingerprinting only users can be accused between rounds, but the code cannot be updated; and in non-adaptive settings the distributor is only allowed to make a final decision at the end of the game.

1.3 — Searching for two colluders

To illustrate the above model, let us consider a very basic example, where we know that there exists a collusion of size $c = 2$. By the marking assumption, we know that if both colluders receive the same symbol they are forced to output this symbol, while if they receive different symbols they are allowed to choose either of them to output.

Encoder. Perhaps the simplest solution one could think of to design a fingerprinting code for this setting, namely letting the code matrix X consist of uniformly random bits (with probability $\frac{1}{2}$ of either a 0 or a 1), turns out to work very well in the case of two colluders. In the bias-based setting this corresponds to fixing $p \equiv \frac{1}{2}$, or equivalently $f_P(p) = \delta(p - \frac{1}{2})$ where δ is the Dirac delta-function.

Decoder. For the decoding procedure, let us use the following method: we assign a score S_j to each user j, where $S_j = |\{i : x_{j,i} = y_i\}|$ counts how often this user had a symbol matching the pirate output. We decide to include user j in the set of accused users $\mathcal{C}'$ if and only if his score S_j exceeds a certain *accusation threshold* η to be specified later.

Performance. For innocent users, note that $\mathbb{P}(x_{j,i} = y_i) = \frac{1}{2}$, and so an innocent user's score S_j will be distributed according to a binomial distribution on ℓ trials with probability of success $\frac{1}{2}$, i.e., $S_j \sim \mathrm{Bin}(\ell, \frac{1}{2})$. For guilty users j_1 and j_2, note that with probability $\frac{1}{2}$ both S_{j_1} and S_{j_2} increase by 1 (if they receive the same symbol, they are forced to output this symbol), and with probability $\frac{1}{2}$ only one of S_{j_1} and S_{j_2} increases by 1. Letting $S_{\mathcal{C}} = S_{j_1} + S_{j_2}$, we see that $S_{\mathcal{C}} - \ell \sim \mathrm{Bin}(\ell, \frac{1}{2})$. Then, assuming that the colluders share the blame equally and randomly select one of their symbols to output, both $2S_{j_1} - \ell$ and $2S_{j_2} - \ell$ follow (dependent!) binomial distributions with parameters ℓ and $\frac{1}{2}$ as well. To summarize:

- For innocent users, we have $S_j \sim \mathrm{Bin}(\ell, \frac{1}{2})$;
- For the colluders j_1, j_2, we have $S_{j_{1,2}} \sim \frac{1}{2}\ell + \frac{1}{2}\mathrm{Bin}(\ell, \frac{1}{2})$.

In particular, note that innocent users have an average score of $\frac{1}{2}\ell$ after ℓ segments (with a fluctuation in the scores proportional to $\sqrt{\ell}$), and both colluders have an average score of $\frac{3}{4}\ell$, with small standard deviations as well. So intuitively, it should only be a matter of time before the two colluders will have the highest scores among all users. Taking the threshold η appropriately between $\frac{1}{2}\ell$ and $\frac{3}{4}\ell$ and letting ℓ be sufficiently large (based on the exact parameters n, ε_0 and ε_1), we can guarantee that the colluders and only the colluders are caught with high probability[2].

Hypothesis testing. In the end, what we are actually doing here is trying to distinguish between samples from two different distributions, with as few samples as possible. One distribution could be considered the base case (a user is innocent and has score $S_j \sim \mathrm{Bin}(\ell, \frac{1}{2})$), and the other is the less likely alternative (this user is guilty and $S_j \sim \frac{1}{2}\ell + \frac{1}{2}\mathrm{Bin}(\ell, \frac{1}{2})$), and so we need to decide whether the base case holds or whether the assumption that this user is innocent is false, based on the evidence. This is closely related to hypothesis testing, distinguishing between a null hypothesis H_0 (innocent) and an alternative hypothesis H_1 (guilty), and it is not surprising that various solutions from the fingerprinting literature heavily rely on the theory behind hypothesis testing.

[2]Note that a folklore result from fingerprinting is that with such a "binary alphabet" (symbols assigned to users only take two possible values), it is impossible to trace any collusion of size $c \geqslant 2$ deterministically, i.e., with error probabilities $\varepsilon_0 = \varepsilon_1 = 0$ [BS98, Theorem IV.2].

1.4 — Searching for more colluders

In 2003, Tardos [Tar03] proposed a method for collusion-resistant traitor tracing using the bias-based framework above, improving upon earlier literature of e.g. Boneh and Shaw [BS98] by improving the required code length to capture c colluders from $\ell = O(c^4 \log n)$ to only $\ell = O(c^2 \log n)$. In the same paper he also showed that this asymptotic scaling behavior is optimal; any scheme designed against c colluders and n total users, with fixed error probabilities $\varepsilon_0, \varepsilon_1$, requires the use of a code of length at least $\ell = \Omega(c^2 \log n)$. So not only did he show how to solve the arbitrary-c setting much more efficiently than with previous solutions, he also showed that this solution is optimal up to the leading constant and lower order terms. In the field of collusion-resistant fingerprinting, this was a groundbreaking discovery.

Encoder. Tardos' scheme was also the first scheme for arbitrary collusion sizes which fits the bias-based framework outlined above. The encoder and decoder are implemented as follows. For the encoder, up to small additive order terms (which quickly vanish for large collusion sizes), Tardos proposed to use the following density and distribution functions f_P and F_P:

$$f_P(p) = \frac{1}{\pi\sqrt{p(1-p)}}, \qquad F_P(p) = \frac{2}{\pi}\arcsin\sqrt{p}. \qquad (p \in [0,1]) \tag{1.1}$$

This distribution function is known in the literature as the *arcsine distribution*, which is not so surprising given the formula for F_P. Informally speaking, this choice for the encoder turns out to have just the right balance between extreme and non-extreme values of p; we will often have $p \approx 0, 1$ very close to 0 or 1, which means the colluders will often all have the same symbol, but this distribution also guarantees that with non-negligible probability we will have $p \approx \frac{1}{2}$.

Decoder. The decoder of Tardos' scheme depends on an appropriately chosen *score function* g, as well as an accusation threshold η. The accusation procedure, or decoding method, works as follows:

- For each i, j, compute $S_{j,i} = g(x_{j,i}, y_i, p_i)$.
- For each j, accuse user j if $\sum_{i=1}^{\ell} S_{j,i} > \eta$.

Without specifying g and η this description is still very general, and it covers (almost) all known variants of Tardos' original scheme. The choice of g, as well as the method to determine η (and ℓ), are what separates one scheme from another. Originally, Tardos proposed to use the following *asymmetric* score function g:

$$g(x_{j,i}, y_i, p_i) = \begin{cases} +\sqrt{(1-p_i)/p_i}, & \text{if } x_{j,i} = 1, y_i = 1, \\ -\sqrt{p_i/(1-p_i)}, & \text{if } x_{j,i} = 0, y_i = 1, \\ 0, & \text{if } y_i = 0. \end{cases} \tag{1.2}$$

Note that the name comes from the asymmetry between positions i with $y_i = 0$ and $y_i = 1$; all positions with $y_i = 0$ are disregarded for the tracing process. Surprisingly it took another five years before it was discovered that a symmetrized version of this score function [ŠKC08] works twice as well in distinguishing between innocent and guilty users, and may lead to up to four times shorter code lengths due to the quadratic dependence

on this distinguishability:

$$g(x_{j,i}, y_i, p_i) = \begin{cases} +\sqrt{(1-p_i)/p_i}, & \text{if } x_{j,i} = 1, y_i = 1, \\ -\sqrt{(1-p_i)/p_i}, & \text{if } x_{j,i} = 1, y_i = 0, \\ -\sqrt{p_i/(1-p_i)}, & \text{if } x_{j,i} = 0, y_i = 1, \\ +\sqrt{p_i/(1-p_i)}, & \text{if } x_{j,i} = 0, y_i = 0. \end{cases} \tag{1.3}$$

We refer to this score function as the *symmetric* score function, and we refer to the variant of the Tardos scheme using this score function the *symmetric Tardos scheme*.

Performance. Originally, Tardos used the asymmetric score function described above, and a slightly modified cumulative bias distribution function $F_p^{(\delta)}$ of the following form:

$$F_p^{(\delta)}(p) = \frac{2\arcsin\sqrt{p} - 2\arcsin\sqrt{\delta}}{\pi - 4\arcsin\sqrt{\delta}}. \qquad (p \in [\delta, 1-\delta]) \tag{1.4}$$

The parameter $0 < \delta \ll 1$ is often called the *cutoff* parameter, and Tardos used a parameter $\delta = O(1/c)$ to make a certain proof strategy work[3]. He then showed that setting $\eta = O(c \log n)$ and $\ell = O(c^2 \log n)$ leads to constant overall error probabilities, i.e., guaranteeing that no innocent users are accused with high probability, and at least one guilty user is caught with high probability. As Tardos' main goal was to show achievability of a code length $\ell = O(c^2 \log n)$, the parameters were chosen rather arbitrarily (satisfying given bounds), leading to a provable (asymptotic) code length of $\ell \sim 100c^2 \ln n$.

Improvements. As the scaling in c and n was known to be optimal due to Tardos' matching lower bound on ℓ, later work focused on bringing down the leading constant. Using the symmetric score function, and choosing the parameters accurately, this eventually led to an asymptotic code length of $\ell \sim \frac{1}{2}\pi^2 c^2 \ln n \approx 4.93c^2 \ln n$ [LdW14], thus improving upon Tardos' initial result by more than a factor 20. Still, work on lower bounds had also progressed since then, and it was shown that a tight lower bound on ℓ for large c and n is given by $\ell \geqslant 2c^2 \ln n$. This means that there is still a gap of a factor ≈ 2.5 between the upper and lower bounds, and potentially more can be gained by improving the analysis or using a different scheme.

Larger alphabets. Many generalizations of the fingerprinting model were considered in the past, one of which is using q different watermarked versions per content segment rather than 2 as in e.g. [BŠ11, FT01, HM14, LOD12, Sim14, ŠKSC09, ŠKSC11, ŠO15]. For $q > 2$ there are various different models in the literature concerning what the capabilities of the tracer are, and perhaps closest to the above model is the *restricted digit model*: given any number of q-ary symbols, the colluders are forced to output one of them. Under this assumption, codes were initially found which were roughly a factor $\log q$ shorter than binary codes [ŠKC08], while [BŠ11, HM14] showed that the optimal lower bound on ℓ becomes $\ell \geqslant (2c^2 \ln q)/(q-1)$, i.e., decreases by a factor $O(q/\log q)$ as q increases. In this work we will focus on the case of $q = 2$, although most of the results in the following chapters can be generalized to higher alphabets as well.

[3] There are reasons to believe [Ško14] that without such a cutoff the error probability ε_0 of this scheme will be too large, and that this is therefore not just an artifact of the proof strategy, although a proof of this claim was never formally published.

1.5 — Research questions and outline

With the above previous work in mind, we can formulate various research questions which are ultimately aimed at closing the gap between the best known upper and lower bounds. Below we present the relevant research questions, and how they will be answered in each chapter.

Q1. Can the symmetric Tardos scheme be further improved?

The results of [LdW14] showed that given that the bias distribution function is chosen as the arcsine distribution, the best asymptotic code length that can be achieved with the symmetric score function is $\ell \sim \frac{1}{2}\pi^2 c^2 \ln n$. As this is potentially suboptimal, the question remains: can different choices of the distribution function lead to better results? This question is addressed in Chapter 2.

Q2. How difficult is fingerprinting for fixed pirate strategies?

While the main fingerprinting game of course considers the scenario where the pirate strategy is unknown, sometimes making a simplifying assumption in the model can help to understand the problem better, and to gain more insight into the harder, more general problem. So if defending against arbitrary pirate attacks is (too) hard to solve directly, can we perhaps analyze how hard it is to defend against specific, known pirate strategies? This question is addressed in Chapters 3 and 4.

Q3. Can the insight for fixed pirate strategies be used for arbitrary attacks?

Then, following up on the previous question, we may ask: did we learn anything from defending against specific attacks? Can these insights help us choose a better decoder for the general fingerprinting game? This question is mostly addressed in Chapter 4, where we indeed present an improved decoder for the general fingerprinting game, inspired by the results from Chapters 3 and 4.

Q4. How much does adaptivity help in tracing collusions?

As described at the end of the bias-based framework, besides the commonly considered non-adaptive setting there are some cases where the distributor is more powerful, and may be able to trace colluders faster with an adaptive solution. Some previous works (e.g. [FT01, LDR+13]) have shown how to do better in these settings, but the question remains if these solutions are optimal, and how much exactly can be gained from adapting the code or the decoder to pirate feedback. We address this question in Chapter 5.

Q5. Do results in fingerprinting have applications in different fields?

Finally, we broaden our perspective and consider whether the insights or techniques in fingerprinting can be used in different fields of research as well. While very recently, applications of fingerprinting have also been found in differential privacy [BUV14, DTTZ14, SU15, Ull13], we will focus on a scenario closer to fingerprinting, namely group testing. We will address this question in Chapter 6, where results regarding specific pirate strategies will turn out to be particularly useful for this area.

CHAPTER 2

Limitations of symmetric decoding

2.1 — Overview

Context. After the invention of the Tardos scheme [Tar03], which was shown to have an order-optimal code length of $\ell = \Theta(c^2 \log n)$ for large c, various follow-up works focused on e.g. (i) finding improvements to Tardos' original scheme, to reduce the leading constant for large c [BT08, IŠO14, LdW14, OŠD15, OŠD13, ŠKC08, ŠVCT08, Ško15]; (ii) finding improvements that reduce the required code length for small c [CNFS05, NHWI07, NFH$^+$09, Nui09, Nui10, Nui12, Sch03, Sch04]; (iii) finding tight lower bounds on the code length required by any fingerprinting scheme [AT09, Ami10, AB06, ABD08, BK04, BŠ11, BŠ12, HM09a, HM09b, HM10, HM12a, HM12b, HM14, Mou08, SBM05, Tar10]; and (iv) finding further improvements to Tardos' scheme that reduce the costs of tracing the traitors in practice [BS12, CXFF09, DHPG13, FGC08, FPFGC09, FPF09a, FPF09b, FGC12, FD14, Kur13, MF11b, MF12, ODL14, PFF09, SŠ11, SŠ12, SŠ14, Sim14].

Although in many cases these approaches complemented each other, on some occasions it seemed a piece of the puzzle was still missing to tie the results together. For instance, Nuida–Hagiwara–Watanabe–Imai [NHWI07] focused on the case of small c and showed how the distribution function (encoder) can be optimized for the original Tardos scheme to obtain the best performance, while Škorić–Katzenbeisser–Celik [ŠKC08] focused mostly on large-c asymptotics and showed how the score function (decoder) of Tardos' original scheme can be symmetrized to obtain better asymptotic code lengths. Later Nuida–Fujitsu–Hagiwara–Kitagawa–Watanabe–Ogawa–Imai [NFH$^+$09] updated their result to incorporate the symmetrized score function of Škorić–Katzenbeisser–Celik, but there still seemed to be a gap between the two results: with optimized encoders, it was shown that an asymptotic code length of $\ell \approx 5.35c^2 \ln n$ was achievable [NFH$^+$09, Theorem 2]; while with a possibly suboptimal truncated arcsine encoder, an asymptotic code length of $\ell \approx 4.93c^2 \ln n$ seemed optimal [HM09a, HM09b, ŠKC08], which was later proven rigorously by Laarhoven and De Weger [LdW14]. If the encoders proposed by [NFH$^+$09] were indeed optimal for *arbitrary* c, would they not also be optimal for *large* c? As the authors of [NFH$^+$09] only stated that their asymptotic code length of $\ell \approx 5.35c^2 \ln n$ was *sufficient*, rather than *necessary*, the most probable explanation was that their analysis for large c was simply not tight. Using the optimized encoders of [NFH$^+$09], the asymptotic code length should be at least as good as when using truncated arcsine distributions ($\ell \approx 4.93c^2 \ln n$), and possibly even better. In early 2013, it

*This chapter is based on results from [LdW13].

was still an open problem what the actual asymptotic code length would be when using the optimal encoding scheme from [NFH$^+$09].

Results. In this chapter we revisit the work of Nuida–Fujitsu–Hagiwara–Kitagawa–Watanabe–Ogawa–Imai on optimal distribution functions (encoders) for the symmetric Tardos scheme [NFH$^+$09], and we take another look at the large-c asymptotics of these distributions. The main result of this chapter is the following, which basically ties up a loose end regarding the symmetric Tardos scheme.

Theorem 2.1. *For $c \to \infty$, the optimal distribution functions for the symmetric Tardos scheme of [NFH$^+$09] converge to the arcsine distribution.*

Due to previous work on the symmetric Tardos scheme with the (truncated) arcsine distribution [LdW14, ŠKC08], the next corollary follows immediately from Theorem 2.1.

Corollary 2.2. *In the symmetric Tardos scheme, the arcsine distribution encoder is asymptotically optimal, and the optimal code length for large c and n is $\ell = \left(\frac{1}{2}\pi^2 + o(1)\right) c^2 \ln n$.*

Since by 2013, Huang and Moulin [HM09a,HM09b,HM10,HM12a,HM12b] and Amiri and Tardos [AT09, Ami10] had already proved that the optimal asymptotic code length for the general fingerprinting game is $\ell \sim 2c^2 \ln n$, this immediately establishes that the symmetric Tardos scheme is sub-optimal for large c. In other words, in the bias-based fingerprinting framework, assigning scores to users based on the symmetric score function of Škorić–Katzenbeisser–Celik inevitably leads to code lengths which are larger than theoretically necessary. This motivates why the following chapters will focus on using different score functions in the bias-based fingerprinting framework, as these may not be limited by the asymptotic lower bound of $\ell \gtrsim \frac{1}{2}\pi^2 c^2 \ln n$.

Besides the theoretical results described above, focused on large-c asymptotics, we also describe a new class of encoders for the symmetric Tardos scheme, which aim to capture the best properties of the optimal distribution functions of Nuida–Fujitsu–Hagiwara–Kitagawa–Watanabe–Ogawa–Imai using a much simpler description. These distributions may be described as *discrete arcsine distributions*, as they essentially correspond to the arcsine distribution limited to a finite, well-chosen support. Samples from these distributions are significantly easier to generate (e.g. on resource-constrained devices) than samples from the optimal distributions, while the loss in performance compared to using the optimal encoders is small.

Outline. The remainder of this chapter is organized as follows. First, in Section 2.2 we recall the results of [NFH$^+$09] regarding optimal bias distributions for the symmetric Tardos scheme. In Section 2.3 we state and prove the main technical contributions. In Section 2.4 we consider a new class of bias distributions for the symmetric Tardos scheme, based on the results from Section 2.3. Finally, in Section 2.5 we heuristically compare the code lengths in the symmetric Tardos scheme for various different distribution functions, such as those proposed in Section 2.4.

2.2 — Bias distributions in the symmetric Tardos scheme

Recall that in bias-based traitor tracing schemes, the fingerprinting code $\mathcal{X} \subset \{0,1\}^\ell$ is generated by first generating a bias vector $\mathbf{p} \in [0,1]^\ell$, and then generating entries

of each fingerprint according to $X_{j,i} \sim \text{Bernoulli}(p_i)$. The bias vector $\mathbf{p}$ is generated by drawing each entry p_i from some distribution F_P, and by the symmetric Tardos scheme we mean a bias-based scheme using the score function introduced by Škorić–Katzenbeisser–Celik [ŠKC08].

2.2.1 – Continuous (truncated) arcsine distributions. A common choice for F_P in the symmetric Tardos scheme is the arcsine distribution with appropriate cutoffs. More precisely, one first computes a cutoff parameter $\delta_c > 0$, and then generates biases from the following cumulative distribution function $F_P^{(c)}$:

$$F_P^{(c)}(p) = \frac{2\arcsin\sqrt{p} - 2\arcsin\sqrt{\delta_c}}{\pi - 4\arcsin\sqrt{\delta_c}}. \qquad (\delta_c \leqslant p \leqslant 1-\delta_c) \tag{2.1}$$

For $c = 10$, the function $F_P^{(10)}$ is shown in Figure 2.1, where $\delta_{10} \approx 0.003$ is chosen according to [LdW14, Theorem 7]. Note that for small values of c, the parameter δ_c has to be sufficiently large to prove that innocent users are not accidentally accused, while for large c the cutoff δ_c needs to tend to 0 to guarantee that colluders are caught even if c is large. As δ_c tends to 0 as a function of c, it is clear that for large c, the distribution functions $F_P^{(c)}$ converge to the well-known arcsine distribution F_P^*, defined on $[0,1]$ as:

$$F_P^*(p) = \frac{2}{\pi}\arcsin\sqrt{p}. \qquad (0 \leqslant p \leqslant 1) \tag{2.2}$$

With these continuous arcsine distribution functions (with cutoffs), it was previously shown that an asymptotic code length of $\ell \sim \frac{1}{2}\pi^2 c^2 \ln n$ is optimal [LdW14].

2.2.2 – Discrete Gauss–Legendre distributions. If the colluders aim to minimize the expected total collusion score (which is the main parameter of interest in the security proofs of e.g. [BT08, LdW14, ŠKC08, ŠVCT08, Tar03]), the optimal distributions to be used by the tracer are in fact discrete distributions with a finite support, and are related to Gauss–Legendre quadratures in numerical analysis [NHWI07, NFH+09]. To define these distributions, we first introduce *Legendre polynomials*. For integer $m \geqslant 1$, the m'th Legendre polynomial is given by

$$P_m(x) = \frac{1}{2^m m!}\left(\frac{d}{dx}\right)^m (x^2-1)^m. \tag{2.3}$$

This polynomial has m simple roots on $(-1,1)$, which we will denote by $-1 < x_{1,m} < x_{2,m} < \ldots < x_{m,m} < 1$. Now, the optimal distribution functions, for arbitrary c, can be expressed in terms of these roots $x_{k,m}$ as follows. Here, 'optimal' means that these distribution functions maximize the expected coalition score per segment, which for large c and n is indeed the main goal of the tracer in the symmetric Tardos scheme [ŠKC08].

Lemma 2.3. *[NHWI07, Theorem 3] Let $\hat{c} = \lceil c/2 \rceil$. Then the optimal distribution function to fight against c colluders, maximizing the expected collusion score, is*

$$F_P^{(c)}(p) = \frac{1}{N_{\hat{c}}} \sum_{k=1}^{\hat{c}} w_{k,\hat{c}}\, \mathbb{1}\{p \geqslant p_{k,\hat{c}}\}. \tag{2.4}$$

Here $N_{\hat{c}}$ *is a normalizing constant,* $\mathbb{1}$ *is the indicator function, and the points* $p_{k,\hat{c}}$ *and weights* $w_{k,\hat{c}}$ *are defined as*

$$p_{k,\hat{c}} = \frac{x_{k,\hat{c}} + 1}{2}, \qquad w_{k,\hat{c}} = \frac{2}{(1 - x_{k,\hat{c}}^2)^{3/2} \cdot P'_{\hat{c}}\,(x_{k,\hat{c}})^2}. \tag{2.5}$$

The Gauss–Legendre distribution $F_P^{(10)}$ of Lemma 2.3, designed to resist $c = 10$ colluders and with a finite support of size $\hat{c} = 5$, is shown in Figure 2.1. For small collusion sizes, this distribution function leads to much shorter codes than those obtained using the arcsine distributions with cutoffs. For large c, the code length parameter increases, and [NFH+09] showed that an asymptotic code length of $\ell \approx 5.35c^2 \ln n$ is sufficient for large c when using these distribution functions $F_P^{(c)}$. For details on this asymptotic result and on how Lemma 2.3 was derived, the reader is referred to [NHWI07, NFH+09].

2.3 — Discrete Gauss–Legendre distributions

Focusing on Nuida–Fujitsu–Hagiwara–Kitagawa–Watanabe–Ogawa–Imai's optimized discrete Gauss–Legendre distributions, we will prove that letting c tend to infinity in the family of discrete Gauss–Legendre distributions leads exactly to the arcsine distribution (cf. Theorem 2.1). This will be done by proving the following main result.

Theorem 2.4. *Let the parameters* $p_{k,\hat{c}}$, $w_{k,\hat{c}}$, *and* $N_{\hat{c}}$ *be as defined as in Lemma 2.3. Let* $\alpha > 0$, *and let* k *satisfy* $\alpha c < k < (1 - \alpha)c$ *as* $c \to \infty$. *Then, for large* c *we have*

$$p_{k,\hat{c}} = \sin^2\left(\frac{\pi k}{2\hat{c}}\right) + o(1), \qquad w_{k,\hat{c}} = \frac{\pi}{\hat{c}} + o\left(\frac{1}{c}\right), \qquad N_{\hat{c}} = \pi - o(1). \tag{2.6}$$

Proof. The proof consists of three parts, corresponding to the three expressions in (2.6).

$p_{k,\hat{c}}$: Let $\theta_{k,\hat{c}} = \arccos(x_{k,\hat{c}})$. From [AS72, Eq. (22.16.6)] we have

$$\theta_{k,\hat{c}} = \left(\frac{4(\hat{c} - k) + 3}{4\hat{c} + 2}\right)\pi + o(1) = \pi - \frac{\pi k}{\hat{c}} + o(1). \tag{2.7}$$

Note that to apply [AS72, Eq. (22.16.6)] we used the fact that $\alpha c < k < (1 - \alpha)c$, as for other values of k this result may not hold.[1] Now, since $\cos(\pi - \phi) = 2\sin^2(\frac{\phi}{2}) - 1$ for all $\phi \in \mathbb{R}$, we obtain

$$x_{k,\hat{c}} = \cos\left(\pi - \frac{\pi k}{\hat{c}} + o(1)\right) = 2\sin^2\left(\frac{\pi k}{2\hat{c}}\right) - 1 + o(1). \tag{2.8}$$

By definition of $p_{k,\hat{c}} = \frac{1}{2}(x_{k,\hat{c}} + 1)$, the given expression for $p_{k,\hat{c}}$ follows.

$w_{k,\hat{c}}$: Combining [Sze75, Equations (15.3.1) and (15.3.10)] for $\alpha = \beta = 0$, and using the fact that $2\sin(\frac{\phi}{2})\cos(\frac{\phi}{2}) = \sin\phi$ for arbitrary $\phi \in \mathbb{R}$, we get

$$\frac{2}{(1 - x_{k,\hat{c}}^2)P'_{\hat{c}}(x_{k,\hat{c}})^2} = \frac{\pi}{\hat{c}}\,\sin(\theta_{k,\hat{c}}) + o\left(\frac{1}{c}\right). \tag{2.9}$$

[1] This subtle condition on k, which is not stated in [AS72] but does appear in the references of [AS72], was previously overlooked by Marshak [Mar90], leading to an incorrect comparable result on the asymptotics of Gauss–Legendre quadratures.

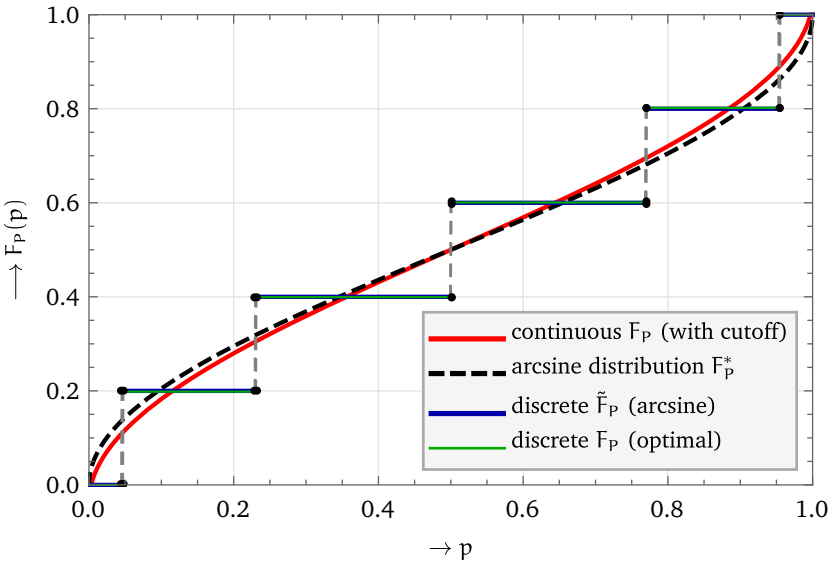


Figure 2.1: The continuous arcsine distribution function $F_P^{(10)}$ with cutoff $\delta_{10} \approx 0.003$ (red) and the discrete Gauss–Legendre distribution function $F_P^{(10)}$ (green), both corresponding to the case $c = 10$. The dashed curve shows the arcsine distribution function. The discrete arcsine distribution (blue) is described in Section 2.4.

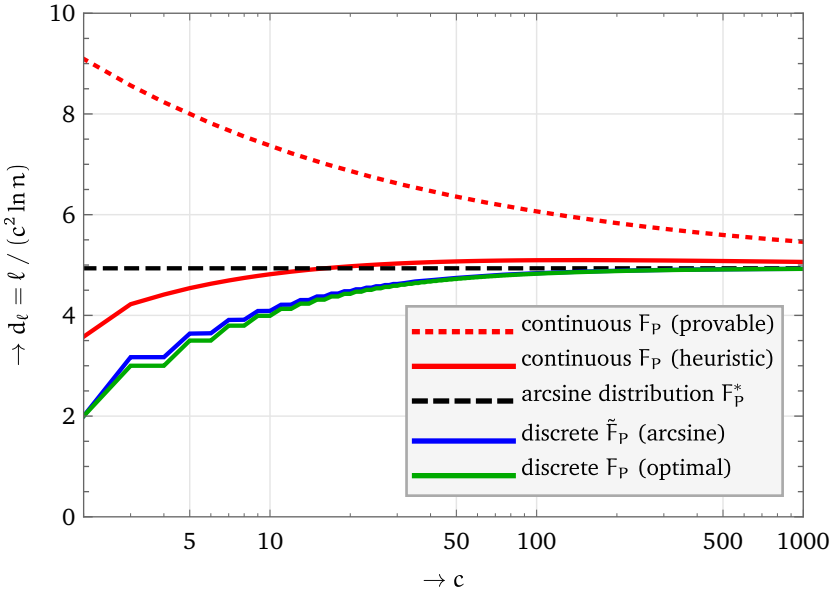


Figure 2.2: Estimates of the code length parameter d_ℓ for several distribution functions F. The dashed line shows the asymptotic optimal value $d_\ell = \frac{1}{2}\pi^2$ corresponding to the arcsine distribution. Note that the discrete distributions are the same for $2c$ and $2c - 1$ colluders, which explains the stepwise behavior of these curves.

Dividing both sides by $\sqrt{1-x_{k,\hat{c}}^2} = \sin\theta_{k,\hat{c}}$, we obtain the expression for $w_{k,\hat{c}}$ of (2.6).

$N_{\hat{c}}$: First, note that the exact expression for the normalizing constant is

$$N_{\hat{c}} = \sum_{k=1}^{\hat{c}} w_{k,\hat{c}}. \tag{2.10}$$

Now, the Gauss–Legendre quadrature rule [AS72, Eq. (25.4.29)] states that for any analytic function f, there exist constants $A_{\hat{c}} > 0$ and $\xi \in (-1,1)$ such that

$$\int_{-1}^{1} f(x)\,dx = \sum_{k=1}^{\hat{c}} \frac{2f(x_{k,\hat{c}})}{(1-x_{k,\hat{c}}^2)P'_{\hat{c}}(x_{k,\hat{c}})^2} + A_{\hat{c}} f^{(2\hat{c})}(\xi). \tag{2.11}$$

For polynomials of degree at most $2\hat{c}$, the remainder term vanishes, and the sum of weighted function values is equal to the integral on the left hand side. This explains why Gauss–Legendre quadratures are used in numerical analysis, as complicated integrals can then be approximated by evaluating the function at a small number of points, and taking a weighted linear combination of these function values. Note that the term inside the summation looks very similar to the definition of $w_{k,\hat{c}}$ of (2.5). Now, let $f(x) = (1-x^2)^{-1/2}$, which is analytic on $(-1,1)$. Then we have $f^{(2\hat{c})}(x) > 0$ for all $x \in (-1,1)$, so in particular $f^{(2\hat{c})}(\xi) > 0$. It follows that

$$\pi = \int_{-1}^{1} \frac{dx}{\sqrt{1-x^2}} = \sum_{k=1}^{\hat{c}} w_{k,\hat{c}} + A_{\hat{c}} f^{(2\hat{c})}(x) > \sum_{k=1}^{\hat{c}} w_{k,\hat{c}} = N_{\hat{c}}. \tag{2.12}$$

On the other hand, from the given expressions for $w_{k,\hat{c}}$ and $p_{k,\hat{c}}$, and the fact that $w_{k,\hat{c}} > 0$ for all k, it follows that

$$N_{\hat{c}} > \sum_{k=o(\hat{c})}^{\hat{c}-o(\hat{c})} w_{k,\hat{c}} = (\hat{c}-o(\hat{c}))\left(\frac{\pi}{\hat{c}} + o\left(\frac{1}{\hat{c}}\right)\right) = \pi - o(1). \tag{2.13}$$

So $\pi - o(1) < N_{\hat{c}} < \pi$, which implies the result in (2.6). □

Note that except for the points $p_{k,\hat{c}}$ near 0 and 1, corresponding to $k = o(\hat{c})$ or $k = \hat{c} - o(\hat{c})$, the leading terms of the weights of these points are all equal. Since these points in the 'middle' carry $1 - o(1)$ of the weight of the distribution, the points near 0 and 1 have a negligible total weight, and do not significantly alter the shape of the limiting distribution. For the points in the middle, note that $p_{k,\hat{c}}$ converges to the expected value of the corresponding order statistic of the arcsine distribution, i.e., the value y corresponding to $F_P^*(y) = \frac{k}{\hat{c}}$ is exactly $y = (F_P^*)^{-1}(\frac{k}{\hat{c}}) = \sin^2(\frac{\pi k}{2\hat{c}}) = p_{k,\hat{c}} + o(1)$. Since asymptotically all these points have the same weight, and the set of points $\{p_{k,\hat{c}}\}_{k=1}^{\hat{c}}$ is dense in $(0,1)$ when c tends to infinity, these results imply that $F_P^{(c)}(p) \to F_P^*(p)$ for each $p \in (0,1)$.

Theorem 2.5. *In the symmetric Tardos scheme, the arcsine distribution is asymptotically optimal.*

Škorić–Katzenbeisser–Celik [ŠKC08, Section 6] previously showed that when using the arcsine distribution in the symmetric Tardos scheme, due to the Central Limit Theorem the optimal code length inevitably converges to $\ell \to \frac{1}{2}\pi^2 c^2 \ln n$. Therefore the following corollary is immediate.

Corollary 2.6. *In the symmetric Tardos scheme, the arcsine distribution encoder and a code length of* $\ell = \left(\frac{1}{2}\pi^2 + o(1)\right) c^2 \ln n$ *are asymptotically optimal.*

2.4 — Discrete arcsine distributions

In addition to the previous theoretical results, we present a new, simple class of distribution functions, which are obtained by discarding appropriate order terms in the proof of Theorem 2.4. Compared to the Gauss–Legendre distributions of Lemma 2.3, these distributions are much simpler to state and implement, but still seem to achieve a performance comparable to the optimal distributions. This suggests that these distributions may form a good alternative for the optimal distributions in e.g. resource-constrained environments like smart cards.

As in the proof of Theorem 2.4, according to [AS72, Eq. (22.16.6)] the parameters of the optimal distribution function $F_P^{(c)}$ satisfy

$$p_{k,\hat{c}} \approx \sin^2\left(\frac{4k-1}{8\hat{c}+4}\pi\right), \qquad w_{k,\hat{c}} \approx \frac{\pi}{\hat{c}}, \qquad N_{\hat{c}} \approx \pi. \tag{2.14}$$

To get the exact values of these parameters for large c requires quite some effort, involving e.g. the computation of roots of Legendre polynomials, which are $\hat{c}$'th derivatives of degree-$2\hat{c}$ polynomials. A practical solution may be to store all values $p_{k,\hat{c}}, w_{k,\hat{c}}, N_{\hat{c}}$ in memory, but (i) the range of possible values of c can be large, (ii) in certain applications the amount of memory may be limited, and (iii) one inevitably has to store approximations of these values in memory, introducing slight deviations from the optimal distribution functions $F_P^{(c)}$. If we cannot avoid approximating $F_P^{(c)}$ anyway, and memory is limited, we may just as well use a much simpler[2] approximation to $F_P^{(c)}$, by taking:

$$p'_{k,\hat{c}} = \sin^2\left(\frac{4k-1}{8\hat{c}+4}\pi\right), \qquad w'_{k,\hat{c}} = \frac{\pi}{\hat{c}}, \qquad N'_{\hat{c}} = \pi. \tag{2.15}$$

This translates to the following class of discrete distribution functions $\tilde{F}_P$, where as before $\hat{c} = \lceil c/2 \rceil$ is the size of the support of $\tilde{F}_P^{(c)}$:

$$\tilde{F}_P^{(c)}(p) = \frac{1}{\hat{c}} \sum_{k=1}^{\hat{c}} \mathbb{1}\left\{p \geqslant \sin^2\left(\frac{4k-1}{8c+4}\pi\right)\right\}. \tag{2.16}$$

Generating a bias p from this distribution is equivalent to drawing r uniformly at random from $\{\frac{3\pi}{8\hat{c}+4}, \frac{7\pi}{8\hat{c}+4}, \ldots, \frac{\pi}{2} - \frac{3\pi}{8\hat{c}+4}\}$, and setting $p = \sin^2(r)$. Note that if we were to draw r uniformly at random from the complete interval $[0, \frac{\pi}{2}]$, this would correspond to

[2] Although the given expressions are trivial approximations of (2.14), it is important to note that one should not use the expressions from Theorem 2.4 without the order terms: for small c, the resulting distribution would be much further off from the optimal distribution $F_P^{(c)}$ than with the expressions given in the text.

the arcsine distribution, while drawing r uniformly at random from $[\arcsin(\sqrt{\delta_c}), \frac{\pi}{2} - \arcsin(\sqrt{\delta_c})]$ corresponds to the arcsine distribution with cutoff δ_c. These distributions may therefore appropriately be called discrete arcsine distributions, and of course for large c these distributions also converge to the arcsine distribution Γ_P^*.

Remark 2.1. Interestingly, the parameters

$$p''_{k,\hat{c}} = \sin^2\left(\frac{4k-2}{8\hat{c}}\,\pi\right), \qquad w''_{k,\hat{c}} = \frac{\pi}{\hat{c}}, \qquad N''_{\hat{c}} = \pi, \tag{2.17}$$

correspond to the parameters of the so-called *Chebyshev–Gauss quadratures* [AS72, Eq. (25.4.38)] as opposed to Gauss–Legendre quadratures, considered previously. These quadratures are commonly used to approximate integrals of the form

$$\int_{-1}^{1} \frac{g(x)}{\sqrt{1-x^2}}\,dx \approx \sum_{k=1}^{c} w''_{k,c}\, g(x''_{k,c}), \tag{2.18}$$

where $x''_{k,c} = 2p''_{k,c} - 1$ and g is a sufficiently smooth function. Note the resemblance between (2.18) and (2.12), in case $g(x) \equiv 1$ is the constant function with value 1. The distribution functions generated by these weights and points are very similar to the discrete arcsine distributions described above, with the main difference being the "cutoff", which is about a third smaller (i.e., $\frac{2\pi}{8c}$ compared to $\frac{3\pi}{8c+4}$).

2.5 — Estimating code lengths

Let us finally try to give a qualitative comparison of the classes of discrete and continuous distribution functions considered in this chapter, in terms of code lengths. Since the (tails of) distributions of user scores are hard to estimate, and known proof methods are not tight, we will only give a heuristic comparison of the code lengths. We will assume that the scores of users are exactly Gaussian, so that we can get a reasonable estimate for the optimal code length constant as $d_\ell = \ell/(c^2 \ln n) \approx 2c^2/\mu_1^2$, where $\mu_1 = \tilde{\mu}/c$ is the expected average colluder score per position [ŠKC08, Corollary 2]. In the case of the discrete distributions of [NFH+09], μ_1 does not depend on the pirate strategy, and we can compute its value exactly. For the arcsine distribution with cutoffs and the discrete arcsine distribution, μ_1 does depend on the pirate strategy, but by considering the attack that minimizes μ_1 we can obtain reasonable estimates for d_ℓ.

Figure 2.2 shows the resulting estimates of d_ℓ, as well as the provably sufficient values for d_ℓ of [LdW14]. Note that the heuristic estimates for the continuous distributions are based on using the arcsine distributions with cutoffs optimized for the proof technique of [LdW14], and so these results may not be tight. As illustrated in Figures 2.1 and 2.2, the discrete arcsine distributions approximate the optimal Gauss–Legendre distributions very well, and seem to achieve a comparable performance for most values of c.

CHAPTER 3

Non-adaptive fingerprinting capacities

3.1 — Overview

Context. As described in Chapter 2, the binary symmetric Tardos scheme [ŠVCT08] is bound by an asymptotic code length of at least $\ell \approx 4.93c^2 \ln n$, while previous work showed that constructions must exist which attain an asymptotic code length of $\ell \sim 2c^2 \ln n$. As a result, for large collusion sizes using the symmetric score function is sub-optimal, and to achieve a better performance in this regime one inevitably has to use different score functions or alternative decoding schemes.

Since by 2013, more than a decade had passed since the invention of the Tardos scheme, and still no simple "capacity-achieving" decoder had been invented[1], it made sense to consider a slightly simplified problem, and see if answers to these easier questions would lead to further insights or solutions for the harder, more general problem. In particular, if finding solutions for the model of unknown pirate attacks is hard, why not start with the easier problem of defending against known pirate strategies, and see what this teaches us about the general fingerprinting game between the colluders and the tracer? Can we find optimal solutions in case the tracer knows exactly how the pirates will mix their copies? And what do 'optimal solutions' actually correspond to for the case of known attacks?

Known attacks. In this chapter we will attempt to answer the latter question: What exactly does it mean for a strategy-dependent tracing scheme to be optimal? More specifically, what is the optimal asymptotic code length that such a scheme should achieve, to claim optimality? The other questions are postponed to the following chapters. For concreteness, we will consider explicit pirate strategies θ previously considered in e.g. [BS12, CXFF09, FPF09a, FD14, HM12b, Laa13a, MF11a, MF12, OŠD15, OŠD13] (and described in the preliminaries in Chapter 1), and derive explicit asymptotics of the required code lengths to defend against these attacks. The results in this chapter build upon previous work of Huang and Moulin [HM12b], and similar to previous work on fingerprinting capacities, we will distinguish between simple and joint decoders, where the distinction roughly corresponds to the computational complexity of the decoder. An overview of the results of this chapter can be found in Table 3.1, which contains the asymptotics of

*This chapter is based on results from [Laa14, Laa15b].

[1] Some papers had previously investigated decoders which may be capacity-achieving for large c (e.g. [AT09, MF12, Mou08]), but unlike e.g. the symmetric score function, these did not admit simple, explicit expressions for the score $S(x, y, p)$ given the user's symbol x, pirate symbol y, and bias p, and did not come with provable bounds on the error probabilities for given code lengths.

the capacities for several pirate strategies. Note that capacities C translate to asymptotic code lengths ℓ according to $\ell \sim C^{-1} \log_2 n$, e.g., we show that any simple decoder for the coin-flip attack needs a code of length at least $\ell \sim \frac{4c \ln n}{(\ln 2)^2}$.

Considering the results in Table 3.1, it is worth noting that for most pirate strategies θ there is an asymptotic gap between the simple and joint capacities. In other words, a joint decoder can theoretically perform significantly better for most pirate strategies. In many cases the optimal code length further scales only linearly (rather than quadratically) in c, and from the attacks listed in Table 3.1 clearly the interleaving attack is the hardest considered attack to deal with from the point of view of the tracer; dealing with this pirate strategy is (almost) as hard as dealing with unknown attacks.

Arbitrary attacks. Surprisingly, one of the results of this chapter is that the maximum achievable code rate for the joint fingerprinting game when the colluders use the interleaving attack is of the order $\beta/c^2 \approx 0.84/c^2$ which is strictly higher than the joint capacity of $\frac{1}{2c^2 \ln 2} \approx 0.72/c^2$ for the general, uninformed fingerprinting game. This result contradicts earlier statements of Huang and Moulin [HM12b, Corollaries 6, 7], who claimed that the interleaving attack and the arcsine distribution form an asymptotic saddle-point solution in the joint fingerprinting game, in the sense that neither the distributor nor the colluders have anything to gain by unilaterally departing from this point. Instead, we show that the distributor can deviate from this saddle point and achieve higher code rates when the colluders fix their attack to be the interleaving attack, regardless of c and regardless of the tracing strategy. A consequence of this result is the following:

Theorem 3.1. *The interleaving attack is a suboptimal attack for large* c *and* ***cannot*** *be part of a saddle-point solution of the joint fingerprinting game.*

The flaw in Huang and Moulin's results is exactly in the assumption they made to perform their analysis [HM12b, Condition 3]: whereas they claim that their conditions for the asymptotic regime (including this particular assumption) are only "mild regularity assumptions" which do not affect the overall outcome, we show that this condition is too restrictive on the capabilities of the tracer: there exist classes of encoders which do not satisfy [HM12b, Condition 3] but which are of significant importance to the fingerprinting game and therefore cannot be ignored.

A further consequence of the above result is that the joint fingerprinting capacity result of Huang and Moulin is incorrect or at least incomplete, and so deriving the joint fingerprinting capacity is again an open problem. By achievability results of e.g. Amiri and Tardos [AT09] and the above result on the joint capacity for the interleaving attack, currently the best provable bounds on the joint fingerprinting capacity under the marking assumption are[2]

$$\frac{0.72}{c^2} \approx \frac{1}{2c^2 \ln 2} \leqslant C^{j}(\mathcal{P}_{\text{mark}}) \leqslant \frac{\beta}{c^2} \approx \frac{0.84}{c^2}. \tag{3.1}$$

Note that while Huang and Moulin's proof of the joint fingerprinting capacity may be considered *incomplete* (i.e. the joint capacity may be equal to the given value, but the proof contains a gap), the result that fixing the attack as the interleaving attack (without restrictions on the tracing strategy) and then taking the limit of large c leads to a joint capacity of $1/(2c^2 \ln 2)$ is provably *incorrect*.

[2]Formal definitions of $C^{j}(\mathcal{P}_{\text{mark}})$ and β can be found in Section 3.3.

Pirate strategy	Simple capacity	Joint capacity
θ_{int}: interleaving attack	$\left(\frac{1}{2\ln 2}\right)/c^2 \approx 0.72/c^2$ [HM12b]	$\beta/c^2 \approx 0.84/c^2$
θ_{all1}: all-1 attack	$(\ln 2)/c \approx 0.69/c$	$1/c \approx 1.00/c$ [Mal78]
θ_{maj}: majority voting	$\left(\frac{1}{\pi\ln 2}\right)/c \approx 0.46/c$	$1/c \approx 1.00/c$
θ_{min}: minority voting	$(\ln 2)/c \approx 0.69/c$	$1/c \approx 1.00/c$
θ_{coin}: coin-flip attack	$\left(\frac{1}{4}\ln 2\right)/c \approx 0.17/c$	$\left(\log_2 \frac{5}{4}\right)/c \approx 0.32/c$

Table 3.1: An overview of the capacity results derived in this chapter, for simple and joint decoders.

Outline. The remainder of this chapter is devoted to proving the results in Table 3.1, and studying the implications for the general fingerprinting game with unknown attacks. In Section 3.2 we derive the exact asymptotics of the simple capacities for various pirate strategies. In Section 3.3 we do the same for joint capacities, leading to the results in the right-most column of Table 3.1. Section 3.4 finally discusses the implications of the new result on the joint interleaving capacity on the uninformed fingerprinting game.

3.2 — Simple capacities

In a simple-decoding traitor tracing scheme, the decoder bases the decision whether or not to accuse user j only on the j'th code word of $\mathcal{X}$, and not on code words corresponding to other users. This means that the decoding step of deciding whether or not to accuse user j will generally be fast, but less accurate than when all information available to the decoder (the entire code $\mathcal{X}$) is taken into account.

Huang and Moulin [HM09a, HM09b, HM10, HM12b] previously studied simple capacities in the context of fingerprinting, and showed that given a set of allowed collusion channels (pirate strategies) $\mathcal{P}^c$ and a set of allowed encoders (bias distribution functions) $\mathcal{P}^e$, a fingerprinting rate[3] R^s is achievable[4] by a simple decoder if and only if

$$R^s \leqslant C^s(\mathcal{P}^e, \mathcal{P}^c) = \max_{F_P \in \mathcal{P}^e} \min_{\theta \in \mathcal{P}^c} \mathbb{E}_P I^s(p, \theta), \tag{3.2}$$

where $\mathbb{E}_p$ is the expectation over values of p sampled from F_P, and $I^s(p,\theta) = I(X_1; Y|P = p)$ is the mutual information between a colluder symbol $X_{j,i}$ and the pirate output Y_i (given the bias p) in one segment i. In this chapter we let $\mathcal{P}^e$ be the set of all distribution functions on $(0,1)$, denoted by $(\mathcal{P}^e)^*$, and we will commonly omit the argument $\mathcal{P}^e$ and write $C^s(\mathcal{P}^c) = C^s((\mathcal{P}^e)^*, \mathcal{P}^c)$. For fixed collusion channels $\mathcal{P}^c = \{\theta\}$, choosing $F_P(p) = \mathbb{1}\{p \geqslant p_0\}$ is optimal [HM12b, Section IV.B], where $p_0 \in (0,1)$ is the value of p maximizing the mutual information. In that case the expression from (3.2) reduces to

$$C^s(\{\theta\}) = \max_{F_P \in \mathcal{P}^e} \mathbb{E}_P I^s(p,\theta) = \max_{p \in (0,1)} I(X_1; Y|P = p). \tag{3.3}$$

[3] Formally, a *fingerprinting rate* R_c is defined to be *achievable* if there exists a sequence of traitor tracing schemes such that, for a fixed number of colluders c and a total number of users $n = 2^{\ell R_c}$, both the false-positive and false-negative error probabilities tend to 0 as $\ell \to \infty$. The *fingerprinting capacity* R_c^* is then defined as the supremum over all achievable fingerprinting rates, and large-c asymptotics of the fingerprinting capacity correspond to taking the limit of R_c^* for large c. Informally, the rate of a code corresponds to $R \sim \log_2(n)/\ell$, and the fingerprinting capacity R defines the minimum value of $\ell \sim \log_2(n)/R$ that may be achieved.

[4] Certain conditions on $\mathcal{P}^c$ and $\mathcal{P}^e$ must be satisfied for this to hold, which can be found in [HM12b].

In case the set of allowed pirate attacks consists of just one attack $\mathcal{P}^c = \{\theta\}$, with slight abuse of notation we will abbreviate the left hand side as $C^s(\theta) = C^s(\{\theta\})$.

To study the mutual information payoff function $I^s(p, \theta)$ we will use the following identity [HM12b, Equation (61)]:

$$I^s(p, \theta) = p\, d(a_1 \| a) + (1-p)\, d(a_0 \| a), \tag{3.4}$$

where $d(\cdot\|\cdot)$ denotes the relative entropy or Kullback–Leibler divergence, defined by $d(\alpha\|\beta) = \alpha \log_2(\frac{\alpha}{\beta}) + (1-\alpha)\log_2(\frac{1-\alpha}{1-\beta})$, and a, a_0, a_1 are defined as

$$a = \sum_{z=0}^{c} \binom{c}{z} p^z (1-p)^{c-z} \theta_z, \tag{3.5}$$

$$a_0 = \sum_{z=0}^{c-1} \binom{c-1}{z} p^z (1-p)^{c-z-1} \theta_z, \tag{3.6}$$

$$a_1 = \sum_{z=1}^{c} \binom{c-1}{z-1} p^{z-1} (1-p)^{c-z} \theta_z. \tag{3.7}$$

Given p and θ, the above formulas allow us to compute the mutual information $I^s(p, \theta)$ explicitly, and we can perform the optimization described in (3.3).

For obtaining asymptotic expressions for the simple capacities for various models, we will extensively work with the Kullback–Leibler divergence. Some of the involved derivations can be simplified using the following Taylor expansion of $d(\alpha\|\beta)$ around $\alpha = \beta$:

$$d(\alpha\|\beta) = \frac{(\alpha-\beta)^2}{2\beta(1-\beta)\ln 2}\left(1 + O\left(\frac{|\alpha-\beta|}{\beta(1-\beta)}\right)\right). \tag{3.8}$$

Intuitively, this says that the divergence is bigger if α and β are further apart, but for α and β both close to 0 or 1 note that the divergence may be large as well due to the β and $1-\beta$ in the denominator. In that case one has to be careful and see whether $|\alpha-\beta|$ approaches 0 faster than β or $1-\beta$. A special case of (3.8) for $\alpha \approx \frac{1}{2}$ and $\beta = \frac{1}{2}$ is

$$d\left(\frac{1}{2} \pm \gamma \middle\| \frac{1}{2}\right) = \frac{2\gamma^2}{\ln 2} + O(\gamma^4). \tag{3.9}$$

Finally, if $\alpha = \frac{1}{2}$ and $\beta \approx \frac{1}{2}$, we can approximate $d(\alpha\|\beta)$ as

$$d\left(\frac{1}{2}\middle\|\frac{1}{2}(1 \pm \gamma)\right) = \frac{1}{2}\, d\left(1\|1-\gamma^2\right), \tag{3.10}$$

and regardless of α and β, one always has $d(\alpha\|\beta) = d(1-\alpha\|1-\beta)$.

Next, we will study the asymptotics of the simple capacities for five commonly considered fingerprinting attacks.

3.2.1 – Interleaving attack. The interleaving attack in fingerprinting (considered in e.g. [BS12, CXFF09, FPF09a, HM12b, Laa13a, OŠD15]) is characterized by the coalition choosing one of its members at random, and outputting his symbol. Given z members

with a 1 and $c - z$ members with a 0, the probability of outputting a 1 is then equal to $\frac{z}{c}$, regardless of z and c. In other words, $(\theta_{\text{int}})_z = \frac{z}{c}$ for $0 \leqslant z \leqslant c$. This is known to be one of the strongest pirate strategies, and the exact asymptotics of the simple capacity for the interleaving attack were previously also derived by Huang and Moulin.

Proposition 3.2. *[HM12b, Theorem 6] The simple capacity for the interleaving attack is:*

$$C^s(\theta_{int}) = \frac{1}{2c^2 \ln 2} + O\left(\frac{1}{c^4}\right), \tag{3.11}$$

and the maximizing value of p *is* $p^s_{int} = \frac{1}{2}$.

Surprisingly, fighting against arbitrary unknown pirate attacks is as hard as fighting against the interleaving attack, as both require an asymptotic code of length $2c^2 \ln n$ to trace the colluders.

3.2.2 – All-1 attack. Another commonly considered attack in fingerprinting is the all-1 attack, where pirates output a 1 whenever they can [CXFF09, Laa13a, MF11a, OŠD15]. Due to the marking assumption they are forced to output a 0 when they did not receive any ones, but otherwise a coalition using the all-1 attack will always output a 1. In other words, $(\theta_{\text{all1}})_z = \mathbb{1}\{z > 0\}$. The following result shows that this attack is significantly weaker than the interleaving attack.

Proposition 3.3. *The simple capacity and the corresponding maximizing value of* p *for the all-1 attack are:*

$$C^s(\theta_{all1}) = \frac{\ln 2}{c} + O\left(\frac{1}{c^2}\right), \qquad p^s_{all1} = \frac{\ln 2}{c} + O\left(\frac{1}{c^2}\right). \tag{3.12}$$

Proof. First, consider a, a_0 and a_1. Using $\theta_z = 0$ if $z = 0$ and $\theta_z = 1$ otherwise, we get

$$a = \sum_{z=0}^{c} \binom{c}{z} p^z (1-p)^{c-z} \theta_z = 1 - (1-p)^c. \tag{3.13}$$

Working out a_0 and a_1 in a similar way, we get $a_0 = 1 - (1-p)^{c-1}$ and $a_1 = 1$. For ease of notation, let us write $s = (1-p)^c$ and $I(p) = I^s(p, \theta_{\text{all1}})$, so that we get

$$I(p) = p\,d\,(1 \| 1 - s) + (1-p)\,d\left(\frac{s}{1-p} \middle\| s\right). \tag{3.14}$$

Now, consider the second term in (3.14). For large c, we will argue that this term is small regardless of p:

$$(1-p)\,d\left(\tfrac{s}{1-p} \parallel s\right) = -s \log_2(1-p) + (1-p-s) \log_2\left(\frac{1-p-s}{(1-p)(1-s)}\right) \tag{3.15}$$

$$= -s \log_2(1-p) + (1-p-s) \log_2\left(1 - \frac{ps}{(1-p)(1-s)}\right). \tag{3.16}$$

Noting that $\frac{ps}{(1-p)(1-s)} = O(\frac{1}{c})$ and $ps = O(\frac{1}{c})$ for all p, we obtain

$$(1-p)d(\tfrac{s}{1-p} \parallel s) = -s\log_2(1-p) + \frac{1-p-s}{\ln 2}\left[\frac{-ps}{(1-p)(1-s)} + O\left(\frac{1}{c^2}\right)\right] \quad (3.17)$$

$$\overset{(b)}{=} +\frac{ps}{\ln 2} - \frac{ps}{\ln 2}\left[1 - \frac{ps}{(1-p)(1-s)} + O\left(\frac{1}{c^2}\right)\right] \quad (3.18)$$

$$\overset{(c)}{=} +\frac{ps}{\ln 2} - \frac{ps}{\ln 2} + O\left(\frac{1}{c^2}\right) = O\left(\frac{1}{c^2}\right). \quad (3.19)$$

Here (b) follows from $p^2 s = O(\frac{1}{c^2})$, and (c) follows from $\frac{p^2 s^2}{(1-p)(1-s)} = O(\frac{1}{c^2})$ and $p^2 s^2 = O(\frac{1}{c^2})$ for arbitrary p. These bounds can all be derived by investigating the given expressions for p, looking for extreme values, and verifying that even for these values of p the bounds hold. So we are now left with:

$$I(p) = -p\log_2(1-s) + O\left(\frac{1}{c^2}\right). \quad (3.20)$$

For p to attain the global maximum we need either that $I'(p) = 0$ or p should be one of the end-points 0 or 1. For $p \to 0$ or 1 we get $I(p) \to 0$, so we need to find a value $p \in (0,1)$ with $I'(p) = 0$. Writing out the remaining term and differentiating, this condition is equivalent to

$$\frac{cps}{(1-p)(1-s)} = -\ln(1-s). \quad (3.21)$$

Since the left hand side is $O(1)$ regardless of p, the right hand side must be too, so $s = 1 - o(1)$ is excluded. To exclude the case $s = o(1)$ we rewrite (3.21) to get

$$\frac{cp}{1-p} = \frac{1-s}{s}\ln\left(\frac{1}{1-s}\right). \quad (3.22)$$

Now if $s = o(1)$ then the right hand side becomes $1 - o(1)$, which implies in the left hand side that $p = \frac{1}{c} - o(\frac{1}{c})$, which implies that $s \neq o(1)$, contradicting our assumption that $s = o(1)$. So for large c a maximum can only occur at $o(1) < s < 1 - o(1)$. Suppose that $s(c) \to s^* \in (0,1)$ for $c \to \infty$, with $s^* \neq s^*(c)$ not depending on c. Then $p(c) \to p^* = \frac{-1}{c}\ln s^*$, so the condition on p and s is then asymptotically equivalent to:

$$s^* \ln s^* = (1-s^*)\ln(1-s^*) + O\left(\frac{1}{c}\right). \quad (3.23)$$

This has a unique solution at $s^* = \frac{1}{2} + O(\frac{1}{c})$, leading to the given values of p^s_{all1} and $C^s(\theta_{\text{all1}})$. □

In terms of code lengths, this means that any simple decoding algorithm for the all-1 attack requires a code of length of at least $\ell \sim \frac{c\log_2 n}{\ln 2} \approx 2.08c\ln n$ for large c and n. This initially seems to contradict previous results of [Laa13a], which suggested that under a certain Gaussian assumption, only $\ell \sim 2c\ln n$ fingerprint positions are required using Oosterwijk–Škorić–Doumen's decoding scheme [OŠD13]. This apparent contradiction is

caused by the fact that the Gaussian assumption does not hold in the regime of small $p = O(\frac{1}{c})$, for which those results were derived. Rigorous analysis of the scores in [Laa13a] shows that with that scheme, an asymptotic code length of about $\ell \approx 3c \ln n$ is sufficient when $p \sim \frac{1}{c}\ln(2)$, which is well above the lower bound obtained above. Note that this also implies that the decoding scheme of [OŠD13] is asymptotically suboptimal for the all-1 attack.

3.2.3 – Majority voting. The majority voting attack [BS12, CXFF09, FPF09a, Laa13a, MF12, OŠD15] is characterized by the colluders choosing the symbol which they have received the most often. To avoid ambiguity in the definition of $(\theta_{\mathrm{maj}})_{c/2}$ for even c, we will assume c is odd in which case the attack is described by $(\theta_{\mathrm{maj}})_z = \mathbb{1}\{z > \frac{c}{2}\}$. For this attack we obtain the following result.

Proposition 3.4. *For the majority voting attack, the simple capacity is*

$$C^s(\theta_{maj}) = \frac{1}{\pi c \ln 2} + O\left(\frac{1}{c^2}\right), \tag{3.24}$$

and the maximizing value of p *is* $p^s_{maj} = \frac{1}{2}$.

Proof. As mentioned before, to avoid ambiguity we will focus on the case where $c = 2\hat{c}+1$ is odd, and due to symmetry w.l.o.g. we may assume that $p \leqslant \frac{1}{2}$. First, we have:

$$a = \sum_{z=\hat{c}+1}^{2\hat{c}+1} \binom{2\hat{c}+1}{z} p^z (1-p)^{2\hat{c}+1-z}, \tag{3.25}$$

and a_0 and a_1 satisfy $a_0 = a + pu$ and $a_1 = a - (1-p)u$, where $u = \binom{2\hat{c}}{\hat{c}} p^{\hat{c}}(1-p)^{\hat{c}}$. Now if $p = O(\frac{1}{c})$, then a_1 and a_0 quickly approach 0 leading to $I(p) = I^s(p, \theta_{\mathrm{maj}}) = o(\frac{1}{c})$. For the remaining case $p = \omega(\frac{1}{c})$ we expand a using Sanov's theorem [CT06, Theorem 11.4.1] we get

$$a \sim \exp\left[(2\hat{c}+1)\ln(2)\, d\left(\tfrac{1}{2}\|p\right)\right] \sim p^{\hat{c}+\frac{1}{2}}(1-p)^{\hat{c}+\frac{1}{2}} 2^{2\hat{c}+1}. \tag{3.26}$$

Using Stirling's formula for the central binomial coefficient in u, we obtain

$$u = \binom{2\hat{c}}{\hat{c}} p^{\hat{c}}(1-p)^{\hat{c}} \sim \frac{2^{2\hat{c}} p^{\hat{c}}(1-p)^{\hat{c}}}{\sqrt{\pi\hat{c}}}. \tag{3.27}$$

As a consequence, $\frac{u}{a} = o(1)$, and using (3.8) we get

$$d(a_0\|a) \sim \frac{p^2u^2}{2a(1-a)\ln 2}, \qquad d(a_1\|a) \sim \frac{(1-p)^2u^2}{2a(1-a)\ln 2}. \tag{3.28}$$

Combining these expressions, we get

$$I(p) = p\,d(a_1\|a) + (1-p)d(a_0\|a) \sim \frac{2^{4\hat{c}} p^{2\hat{c}+1}(1-p)^{2\hat{c}+1}}{2\pi\,\hat{c}\,a(1-a)\ln 2}. \tag{3.29}$$

To see that this has a maximum at $p = \frac{1}{2}$, writing out the inverse of the above expression (ignoring constants) we see that, in terms of p,

$$\frac{1}{I(p)} \propto \sum_{z_1,z_2=0}^{\hat{c}} \binom{2\hat{c}+1}{z_1}\binom{2\hat{c}+1}{z_2}\left(\frac{p}{1-p}\right)^{z_1-z_2} \tag{3.30}$$

$$= C_1 + C_2 \sum_{z_1<z_2}\left[\left(\frac{p}{1-p}\right)^{z_2-z_1} + \left(\frac{1-p}{p}\right)^{z_2-z_1}\right] \tag{3.31}$$

$$= C_1 + C_2 \sum_{z_1<z_2}\left[2\cosh((z_2-z_1)\ln x)\right], \tag{3.32}$$

where $x = \frac{1-p}{p} > 1$ for $p < \frac{1}{2}$ and $x = 1$ if $p = \frac{1}{2}$, and C_1, C_2 are expressions that do not depend on p. The function between square brackets is positive and increasing in x for $x \geqslant 1$, so it has a global minimum at $x = 1$, corresponding to $p = \frac{1}{2}$. So the maximum for $I(p)$ is attained at $p = \frac{1}{2}$, in which case u satisfies

$$u = \sqrt{\frac{2}{\pi c}}\left(1 + O\left(\frac{1}{c}\right)\right). \tag{3.33}$$

To get exact asymptotics for $I(\frac{1}{2})$, we return to the expression for $I(p)$ of (3.8). Since from (3.25) it follows that $a = \frac{1}{2}$, and both terms are identical, using (3.9) we obtain:

$$I\left(\frac{1}{2}\right) = d\left(\frac{1}{2} + \frac{u}{2}\middle\|\, \frac{1}{2}\right) = \frac{u^2}{2\ln 2} + O(u^4). \tag{3.34}$$

Combining this with the previous expression for u, we obtain the claimed result. □

This result matches the bounds obtained in [Laa13a], which showed that with an almost trivial decoding algorithm of assigning users a score of $+1$ for a match and -1 for a mismatch, we asymptotically achieve a code length of $\ell \sim \pi c \ln n$ for large n and c.

3.2.4 – Minority voting. As the name suggests, when pirates use the minority voting attack [BS12, CXFF09, FPF09a, Laa13a, OŠD15], they output the symbol they have received the least often. Due to the marking assumption they are not able to output symbols they have not received, so in the binary setting the attack is defined as $(\theta_{\min})_z = \mathbb{1}\{0 < z < \frac{c}{2} \text{ or } z = c\}$, where we again assumed for convenience that c is odd. As shown below, this attack has the same simple capacity as the all-1 attack.

Proposition 3.5. *The simple capacity and the corresponding optimal value of* p *for the minority voting attack are:*

$$C^s(\theta_{min}) = \frac{\ln 2}{c} + O\left(\frac{1}{c^2}\right), \qquad p^s_{min} = \frac{\ln 2}{c} + O\left(\frac{1}{c^2}\right). \tag{3.35}$$

Proof. In this case the function $I(p)$ is symmetric around $p = \frac{1}{2}$, so w.l.o.g. we may assume $p \leqslant \frac{1}{2}$. For small values of p, minority voting is equivalent to the all-1 attack up to negligible order terms, while for $p \approx \frac{1}{2}$ the attack is very similar to majority voting by $\theta_{\min} \approx 1 - \theta_{\text{maj}}$. This means that for small p the mutual information payoff will be asymptotically equivalent to that of the all-1 attack, while for $p \approx \frac{1}{2}$ we get the same values as for majority voting. Since the simple capacity for the all-1 attack is higher than for majority voting, the distributor should choose p close to p^s_{all1}, leading to the result. □

3.2.5 – Coin-flip attack. Instead of choosing a pirate at random and outputting his symbol (the interleaving attack), the pirates may also decide to choose a symbol at random from their set of received symbols, without paying attention to how often they received each symbol [BS12, FPF09a, Laa13a, OŠD15]. In other words, when a coalition receives both symbols, they let a fair coin-flip decide which symbol to output. This means that the collusion channel satisfies $(\theta_{\text{coin}})_z = \frac{1}{2}(\mathbb{1}\{z > 0\} + \mathbb{1}\{z = c\})$. This pirate attack is weaker than the interleaving attack, but stronger than the other pirate attacks considered above, as the following result shows.

Proposition 3.6. *For the coin-flip attack, the simple capacity and the corresponding maximizing value of* p *are:*

$$C^s(\theta_{coin}) = \frac{\ln 2}{4c} + O\left(\frac{1}{c^2}\right), \qquad p^s_{coin} = \frac{\ln 2}{2c} + O\left(\frac{1}{c^2}\right). \tag{3.36}$$

Proof. Since $I(p)$ is symmetric around $p = \frac{1}{2}$, let us assume w.l.o.g. that $p \leqslant \frac{1}{2}$. For a, a_0 and a_1 we obtain:

$$a = \tfrac{1}{2}\left(1 + p^c - (1-p)^c\right), \quad a_0 = \tfrac{1}{2}\left(1 - (1-p)^{c-1}\right), \quad a_1 = \tfrac{1}{2}\left(1 + p^{c-1}\right). \tag{3.37}$$

For the mutual information $I(p) = I^s(p, \theta_{\text{coin}})$ we obtain

$$I(p) = p\,d\left(\frac{1}{2}\left(1 + p^{c-1}\right) \middle\| \frac{1}{2}\left(1 + p^c - (1-p)^c\right)\right) \tag{3.38}$$

$$+ (1-p)\,d\left(\frac{1}{2}\left(1 - (1-p)^{c-1}\right) \middle\| \frac{1}{2}\left(1 + p^c - (1-p)^c\right)\right). \tag{3.39}$$

As $p \leqslant \frac{1}{2}$, the terms p^c and p^{c-1} are exponentially small in c and thus negligible, so up to small order terms we get

$$I(p) = p\,d\left(\tfrac{1}{2}\middle\|\tfrac{1}{2}\left(1 - (1-p)^c\right)\right) + (1-p)\,d\left(\tfrac{1}{2}\left(1 - (1-p)^{c-1}\right) \middle\| \tfrac{1}{2}\left(1 - (1-p)^c\right)\right). \tag{3.40}$$

Similar to the proof of the all-1 attack, the second term is $O(\frac{1}{c^2})$, while using (3.10) we can rewrite the first term to a recognizable form:

$$I(p) = \frac{1}{2}\left[-p \log\left(1 - (1-p)^{2c}\right)\right] + O\left(\frac{1}{c^2}\right). \tag{3.41}$$

The term between square brackets is exactly the dominating term for the simple capacity of the all-1 attack for $c_0 = 2c$ colluders. In other words, if we denote the mutual information corresponding to given p, θ, c by $I^s_{(c)}(p, \theta)$, we have:

$$I^s_{(c)}(p, \theta_{\text{coin}}) = \frac{1}{2}\, I^s_{(2c)}(p, \theta_{\text{all1}}) + O\left(\frac{1}{c^2}\right). \tag{3.42}$$

Using Proposition 3.3, the result follows. □

For this attack, the result in [Laa13a] based on Oosterwijk–Škorić–Doumen's decoder [OŠD15] was again too optimistic due to the Gaussian assumption. Any simple decoder that successfully defends against the coin-flip attack must have a code length of at least $\ell \sim \frac{4c \log_2 n}{\ln 2} \approx 8.33c \ln n$ for large c and n.

3.2.6 – Numerical evaluation. Finally, to get a basic idea how accurate these asymptotic expressions are, Figure 3.1 shows the (normalized) simple capacities for various pirate strategies as a function of p, evaluated for the case $c = 15$. The marked points correspond to the asymptotic optimal values derived above, and one can see that these quite closely resemble the actual maximizing values of p and corresponding values of the capacity.

3.3 — Joint capacities

Where a simple decoder bases its decision to accuse user j only on the jth code word of $\mathcal{X}$ (and not on other code words), a joint decoder is allowed to use all information available to make a more informed decision. In particular, the whole code $\mathcal{X}$ may be taken into account. Huang and Moulin [HM09a, HM09b, HM10, HM12b] previously studied joint capacities as well, and showed that given a set of allowed collusion channels $\mathcal{P}^c$ and a set of allowed encoders $\mathcal{P}^e$, a fingerprinting rate R^j is achievable by a joint decoder if and only if

$$R^j \leqslant C^j(\mathcal{P}^e, \mathcal{P}^c) = \max_{F_P \in \mathcal{P}^e} \min_{\theta \in \mathcal{P}^c} \mathbb{E}_P I^j(p, \theta), \tag{3.43}$$

where $I^j(p, \theta) = \frac{1}{c} I(X_1, \ldots, X_c; Y|P = p)$ is the mutual information between all colluder symbols $X_1, \ldots, X_c$ and the pirate output Y in one segment i. Note that from the assumption that Y only depends on $X_1, \ldots, X_c$ through θ, it follows that $I(X_1, \ldots, X_c; Y|P = p) = I(Z; Y|P = p)$, where $Z = \sum_{i=1}^{c} X_i$. To study the payoff function $I^j(p, \theta) = I(Z; Y|P = p)$, we will use the following identity [HM12b, Equation (59)]:

$$I^j(p, \theta) = \frac{1}{c} \left[h(a) - a_h \right], \qquad a_h = \sum_{z=0}^{c} \binom{c}{z} p^z (1-p)^{c-z} h(\theta_z). \tag{3.44}$$

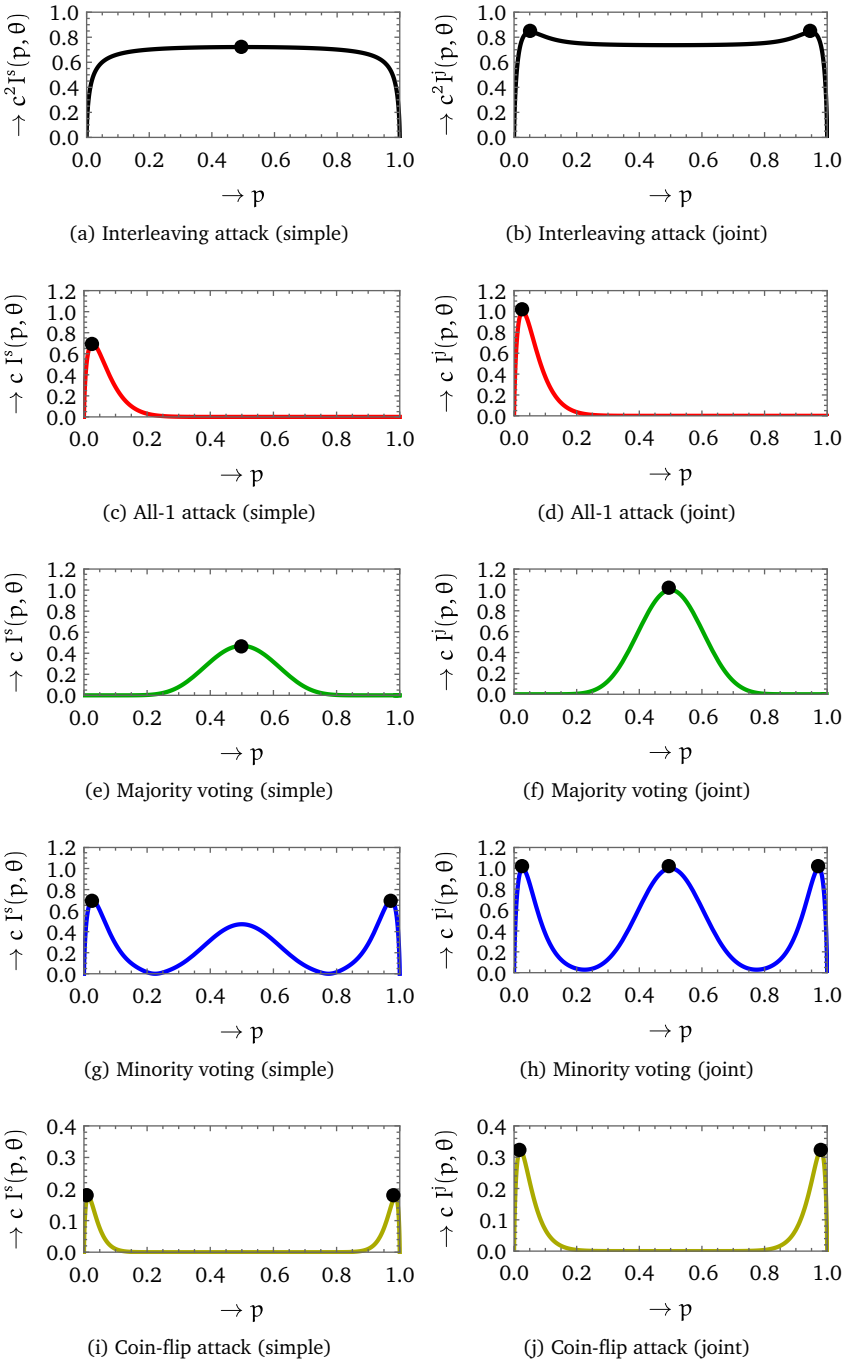
Here $h(\cdot)$ denotes the binary entropy function, defined by $h(\alpha) = -\alpha \log_2 \alpha - (1 - \alpha) \log_2(1 - \alpha)$. Given p and θ, this allows us to compute $I^j(p, \theta)$ explicitly. In the analysis of specific models θ, we will again commonly omit θ as an argument of I and write $I(p)$.

For obtaining the joint capacities for various models, we will extensively work with the binary entropy function. Again, this function can be quite ugly for arbitrary arguments α, but in some cases we can somewhat simplify the expressions. For instance, for arguments close to 0 or $\frac{1}{2}$ we have

$$h(\gamma) = \frac{\gamma(1 - \ln \gamma)}{\ln 2} - O(\gamma^2) = O(\gamma \ln \gamma), \tag{3.45}$$

$$h\left(\frac{1}{2} \pm \gamma\right) = 1 - \frac{2\gamma^2}{\ln 2} - O(\gamma^4) = 1 - O(\gamma^2). \tag{3.46}$$

The most important properties to keep in mind are that $h(0) = h(1) = 0$ and h takes its maximum at $\alpha = \frac{1}{2}$ with $h(\frac{1}{2}) = 1$. Using only these latter properties, we immediately get the following lemma regarding deterministic attacks, i.e., attacks satisfying $\theta \in \{0, 1\}^{c+1}$.



(a) Interleaving attack (simple)

(b) Interleaving attack (joint)

(c) All-1 attack (simple)

(d) All-1 attack (joint)

(e) Majority voting (simple)

(f) Majority voting (joint)

(g) Minority voting (simple)

(h) Minority voting (joint)

(i) Coin-flip attack (simple)

(j) Coin-flip attack (joint)

Figure 3.1: Normalized capacities for various attack, evaluated for $c = 15$. Points indicate asymptotic optima.

Lemma 3.7. *For any deterministic attack θ satisfying the marking assumption $\theta_0 = 0$ and $\theta_c = 1$, the joint capacity equals $C^j(\theta) = \frac{1}{c}$, and p is a maximizing value if it satisfies*

$$a = \sum_{z:\ \theta_z = 1} \binom{c}{z} p^z (1-p)^{c-z} = \frac{1}{2}. \tag{3.47}$$

Proof. Since $\theta_z \in \{0,1\}$ for all z, we have $h(\theta_z) = 0$ for each z, so $a_h = 0$ and it thus follows that

$$C^j(\theta) = \max_{p\in(0,1)} \frac{1}{c}\left[h(a) - a_h\right] = \frac{1}{c} \max_{p\in(0,1)} h(a). \tag{3.48}$$

Since $a = a(p)$ is continuous in p, and $a(0) = 0$ and $a(1) = 1$ due to the marking assumption, from the intermediate value theorem it follows that there must be a value $p \in (0,1)$ for which $a(p) = \frac{1}{2}$. So we get

$$C^j(\theta) = \frac{1}{c} \max_{p\in(0,1)} h(a) = \frac{1}{c}\, h\left(\frac{1}{2}\right) = \frac{1}{c}, \tag{3.49}$$

and p is a maximizing value iff $a(p) = \frac{1}{2}$. □

This lemma makes finding the joint capacities and the optimal values of p very easy for several of the following models.

3.3.1 – Interleaving attack. We previously saw that the simple capacity for the interleaving attack (with $(\theta_{\text{int}})_z = \frac{z}{c}$) is proportional to $\frac{1}{c^2}$. The asymptotics for the joint capacity were previously studied by Huang and Moulin as well [HM09a, Section 4.2], but only numerical evidence was provided to support that the capacity is approximately $1.160/(2c^2 \ln 2)$. After inspecting the joint capacity asymptotics, we eventually obtain the following expression, which does not seem solvable by elementary functions.

Proposition 3.8. *The joint capacity for the interleaving attack θ_{int} is:*

$$C^j(\theta_{int}) = \frac{\beta}{c^2} + O\left(\frac{1}{c^4}\right), \qquad p^j_{int} = \frac{\alpha}{c}, \tag{3.50}$$

where $\alpha \approx 1.3382$ is the maximizing value (and $\beta \approx 0.8371$ is the maximum value) of

$$\beta = \max_{\alpha>0} \left\{ -\alpha \log_2 \alpha + \frac{\alpha}{e^\alpha} \sum_{z=1}^{\infty} \frac{\alpha^z}{z!} \log_2(1+z) \right\}. \tag{3.51}$$

Proof. Due to symmetry, w.l.o.g. let us assume that $p \leqslant \frac{1}{2}$, and suppose that $p = \frac{\alpha}{c}$ for some α. From the expressions for $I(p) = I^j(p, \theta_{\text{int}})$ and θ_{int}, we obtain:

$$I(p) = \frac{1}{c}\left[h\left(\frac{\alpha}{c}\right) - \sum_{z=1}^{c-1} \binom{c}{z} \left(\frac{\alpha}{c}\right)^z \left(1 - \frac{\alpha}{c}\right)^{c-z} h\left(\frac{z}{c}\right) \right]. \tag{3.52}$$

For $\alpha = \omega(1)$ (i.e. $1/\alpha = o(1)$), $I(p)$ is asymptotically equal to $(2c^2 \ln 2)^{-1}$. This can be seen by expanding $h(\frac{z}{c})$ around $z = cp$ using a Taylor series expansion, so that

$$h(\tfrac{z}{c}) = h(p) + (\tfrac{z}{c} - p)h'(p) + \tfrac{1}{2}(\tfrac{z}{c} - p)^2 h''(p) + \tfrac{1}{6}(\tfrac{z}{c} - p)^3 h^{(3)}(p) + \ldots. \tag{3.53}$$

Substituting this expansion into the expression for I, we obtain

$$I(p) = \frac{1}{c}\sum_{z=0}^{c}\binom{c}{z}p^z(1-p)^{c-z}\left[h(p) - h\left(\frac{z}{c}\right)\right] \tag{3.54}$$

$$= -\frac{1}{c}\sum_{z=0}^{c}\binom{c}{z}p^z(1-p)^{c-z}\left[(\tfrac{z}{c}-p)h'(p) + \tfrac{1}{2}(\tfrac{z}{c}-p)^2h''(p) + \dots\right] \tag{3.55}$$

$$\overset{(a)}{=} \frac{1}{2c^2\ln 2} - \frac{1}{c}\sum_{z=0}^{c}\binom{c}{z}p^z(1-p)^{c-z}\left[\tfrac{1}{6}(\tfrac{z}{c}-p)^3h^{(3)}(p) + \dots\right]. \tag{3.56}$$

Here (a) follows from noting that the linear term in $\frac{z}{c} - p$ evaluates to 0 (as $\mathbb{E}[Z] = cp$), and the quadratic term evaluates to $(2c^2\ln 2)^{-1}$ (as $\mathrm{Var}(Z) = cp(1-p)$ and $h''(p) = -\frac{1}{p(1-p)\ln 2}$).

Now, for $\alpha = \Theta(c)$ the remaining term can be bounded by $o(c^{-2})$ by writing out the derivatives of h as $h^{(k)}(p) = (k-2)!((-p)^{-k+1} - (1-p)^{-k+1})$ and bounding the resulting expressions appropriately. The result that for fixed $p \in (0,1)$ the payoff is asymptotically equal to $(2c^2\ln 2)^{-1}$ also follows from [HM12b, Section V].

For $\alpha = o(c)$ and $\alpha = \omega(1)$, proving that the payoff does not lead to a global maximum can be done as follows. First, we find two functions h_0 and h_1 which are lower and upper bounds on h on the biggest part of the interval of z, say $z = z_0$ up to $z = c - z_0$, where $z_0 = \Theta(cp)$. This can be done by taking the Taylor expansion above, and using bounds on the fourth derivative. Then we split the summation in I into a summation over small $z \leqslant z_0$, a summation over $z_0 < z < c - z_0$, and a summation over $z \geqslant c - z_0$. For the two tails, we establish that the contribution is small for this choice of z_0, while the middle terms are bounded from above and below by $(2c^2\ln 2)^{-1} + o(c^{-2})$ due to our choices of h_0 and h_1. For complete details on this part, see [McK15].

Finally, if we assume that $\alpha = O(1)$ is at most constant, it is clear that the dominating terms in the summation in $I(p)$ come from small values z. In that case we can simplify various expressions, as (i) $h(\frac{\alpha}{c}) \sim \frac{\alpha}{c}(1 - \log_2(\frac{\alpha}{c}))$, (ii) $\binom{c}{z} \sim \frac{c^z}{z!}$, (iii) $(1-\frac{\alpha}{c})^{c-z} \sim e^{-\alpha}$, and (iv) $h(\frac{z}{c}) \sim \frac{z}{c}(1 - \log_2(\frac{z}{c}))$. Substituting these expressions, we obtain

$$I(p) \sim \frac{1}{c}\left[\frac{\alpha}{c}\left(1 - \log_2\frac{\alpha}{c}\right) - \sum_{z=1}^{c-1}\frac{c^z}{z!}\left(\frac{\alpha}{c}\right)^z e^{-\alpha}\frac{z}{c}\left(1 - \log_2\frac{z}{c}\right)\right] \tag{3.57}$$

$$\sim \frac{1}{c^2}\left[\alpha - \alpha\log_2\alpha + \alpha\log_2 c - \sum_{z=1}^{c-1}\frac{\alpha^z}{z!}e^{-\alpha}(z - z\log_2 z + z\log_2 c)\right]. \tag{3.58}$$

Next, note that the term $z + z\log_2 c = z(1 + \log_2 c)$ in the summation corresponds to the first moment of the Poisson distribution, which leads to a term $\alpha(1 + \log_2 c)$, canceling out two terms before the summation. This leaves us with

$$I(p) \sim \frac{1}{c^2}\left[-\alpha\log_2\alpha + \sum_{z=1}^{c-1}\frac{\alpha^z}{z!}e^{-\alpha}z\log_2 z\right]. \tag{3.59}$$

One can eliminate the factor z inside the summation by canceling it with the factor z in the factorial, pulling out a factor α from the summation, and substituting $z' \leftarrow z - 1$ in

the summation. Finally, as $c \to \infty$, this leads to

$$I(p) \sim \frac{1}{c^2}\left[-\alpha \log_2 \alpha + \frac{\alpha}{e^\alpha}\sum_{z=0}^{\infty}\frac{\alpha^z}{z!}\log_2(1+z)\right]. \tag{3.60}$$

The summation on the right can be read as an expectation of $\log_2(1+Z)$, where Z follows a Poisson distribution with parameter α. Alternatively, one might try to replace the logarithm with a Taylor expansion and note that this leads to a summation of Bell polynomials evaluated at α. Neither method seems to lead to a more explicit expression for the capacity, and the expression above is easy to compute and converges rather quickly. Numerically finding the optimal value of α, maximizing the capacity, then leads to the claimed numbers for α and β. □

Numerical evaluation of the capacity previously led Huang and Moulin to the same asymptotic constant $\beta \approx 0.8371 \approx \frac{1.1604}{2\ln 2}$ [HM09a, Section 4.2].

3.3.2 – All-1 attack. Since the all-1 attack ($(\theta_{\text{all1}})_z = \mathbb{1}\{z > 0\}$) is a deterministic attack and satisfies the marking assumption, the capacity follows immediately from Lemma 3.7, and finding the optimal value of p is straightforward.

Proposition 3.9. *For the all-1 attack, the joint capacity and maximizing value of* p *are:*

$$C^j(\theta_{all1}) = \frac{1}{c}, \qquad p^j_{all1} = \frac{\ln 2}{c} + O\left(\frac{1}{c^2}\right). \tag{3.61}$$

To be precise, the optimal value for p is $p = 1 - 2^{-1/c}$.

3.3.3 – Majority voting. Lemma 3.7 also applies to the majority voting attack, defined as $(\theta_{\text{maj}})_z = \mathbb{1}\{z > \frac{c}{2}\}$, and since this attack is symbol-symmetric the optimal value for p is trivially $p = \frac{1}{2}$.

Proposition 3.10. *For the majority voting attack, the joint capacity and the corresponding optimal value of* p *are:*

$$C^j(\theta_{maj}) = \frac{1}{c}, \qquad p^j_{maj} = \frac{1}{2}. \tag{3.62}$$

Note that the joint capacity for majority voting is equal to the joint capacity of the all-1 attack, while their simple capacities differ. Also note that again, the optimal value for p is asymptotically the same as for the simple capacity.

3.3.4 – Minority voting. Since the minority voting attack $(\theta_{\min})_z = \mathbb{1}\{0 < z < \frac{c}{2}\} + \mathbb{1}\{z = c\}$ is also a deterministic and symbol-symmetric attack, the following result directly follows from Lemma 3.7.

Proposition 3.11. *The joint capacity and a corresponding optimal value of* p *for the minority voting attack are:*

$$C^j(\theta_{min}) = \frac{1}{c}, \qquad p^j_{min} = \frac{1}{2}. \tag{3.63}$$

In fact, there are three values of p that are asymptotically optimal, the other two being $p \approx \frac{\ln 2}{c}$ and $p \approx 1 - \frac{\ln 2}{c}$. That these other values are also optimal (up to first order constants) follows from the fact that the difference between the minority voting attack and the all-1 attack vanishes for large c and $p = \frac{\ln 2}{c}$. By symmetry, the value $p = 1 - \frac{\ln 2}{c}$ leads to the same capacity as $p = \frac{\ln 2}{c}$.

3.3.5 – Coin-flip attack. Besides the interleaving attack, the only other non-trivial fingerprinting attack with respect to joint capacities is the coin-flip attack. This attack is not deterministic, so $a_h > 0$. Working out the details, we obtain the following result.

Proposition 3.12. *For the coin-flip attack, the joint capacity and maximizing value of* p *are:*

$$C^j(\theta_{coin}) = \frac{\log_2(\frac{5}{4})}{c} + O\left(\frac{1}{c^2}\right), \qquad p^j_{coin} = \frac{\ln(\frac{5}{3})}{c} + O\left(\frac{1}{c^2}\right). \tag{3.64}$$

Proof. For a_h, note that $h(\theta_0) = h(\theta_c) = 0$ and $h(\theta_z) = 1$ otherwise, so $a_h = 1 - p^c - (1-p)^c$. For a, recall from the proof of Proposition 3.6 that $a = \frac{1}{2}(1 - (1-p)^c + p^c)$. Combining the above, we get

$$I(p) = I^j(p, \theta_{coin}) = \frac{1}{c}\left[h\left(\frac{1 - (1-p)^c + p^c}{2}\right) - (1 - p^c - (1-p)^c)\right]. \tag{3.65}$$

Since the attack is symbol-symmetric, w.l.o.g. we may assume that $p \leqslant \frac{1}{2}$, in which case the terms p^c are negligible for large c. Writing $t = 1 - (1-p)^c$, we get

$$I(t) = \frac{1}{c}\left[h\left(\frac{t}{2}\right) - t\right] + O\left(\frac{1}{c^2}\right). \tag{3.66}$$

This function has a maximum at $t = 1 - (1-p)^c = \frac{2}{5}$, which leads to the given values of $p^j_{coin} = 1 - \sqrt[5]{3/5}$ and $C^j(\theta_{coin})$. □

3.3.6 – Numerical evaluation. The right half of Figure 3.1 shows the (normalized) joint capacities for various pirate strategies as a function of p, also evaluated for the case $c = 15$. Note that the figures on the left and on the right have the same axes, so the curves on the right (joint capacities) are always at least as high as the curves on the left. The marked points again correspond to the asymptotic optimal values derived in this section, which closely resemble the actual maximizing value of p and corresponding optimal value of the capacity.

3.4 — Arbitrary attacks

Let us now return to the setting that is by far the most considered setting in fingerprinting, where the pirate strategy is not known to the distributor, and the generation of biases p is aimed at defending against arbitrary pirate strategies (that adhere to the marking assumption). In other words, the set of allowed attacks can be described as

$$\mathcal{P}_{mark} = \left\{\theta \in [0,1]^{c+1} \mid \theta_0 = 0, \theta_c = 1\right\}. \tag{3.67}$$

3.4.1 – Simple capacities. For simple decoders, the result of the capacity for the interleaving attack and the sufficiency result of the symmetric Tardos scheme immediately lead to the following upper and lower bounds on the simple capacity for the uninformed fingerprinting game $C^{s}(\mathcal{P}_{\text{mark}})$, which are also stated in e.g. [HM12b, Corollary 2]:

$$\frac{2}{\pi^2 c^2 \ln 2} \lesssim C^{s}(\mathcal{P}_{\text{mark}}) \lesssim \frac{1}{2c^2 \ln 2}. \tag{3.68}$$

By a saddle-point analysis of [OŠD15, Proposition 17], showing that in case the arcsine distribution function is used, the best attack against their scheme is the interleaving attack (leading to an asymptotic code length of $\ell \sim 2c^2 \ln n$), it follows that the simple capacity is asymptotically exactly equal to

$$C^{s}(\mathcal{P}_{\text{mark}}) \sim \frac{1}{2c^2 \ln 2}. \tag{3.69}$$

This implies that the simple decoder of Oosterwijk–Škorić–Doumen [OŠD15] achieves capacity in the simple fingerprinting game.

3.4.2 – Joint capacities. The joint capacity game was previously solved by Huang and Moulin, showing that assuming [HM12b, Condition 3] holds, due to a saddle-point solution at the interleaving attack and the arcsine embedding distribution one has

$$C^{j}(\mathcal{P}_{\text{mark}}) \overset{?}{\sim} \frac{1}{2c^2 \ln 2}. \tag{3.70}$$

However, due to Proposition 3.8 we know that the interleaving attack and the arcsine distribution do *not* form a saddle-point solution in the fingerprinting game for finite c, as the embedder can choose F_P differently to achieve a higher code rate. Therefore the proof of (3.70) of Huang and Moulin is incorrect/incomplete, and the best asymptotic bounds on the joint fingerprinting capacity are currently given by

$$\frac{1}{2c^2 \ln 2} \lesssim C^{j}(\mathcal{P}_{\text{mark}}) \lesssim \frac{\beta}{c^2}, \tag{3.71}$$

where $\beta \approx 0.84$ is defined in Proposition 3.8.

CHAPTER 4

Non-adaptive decoding schemes

4.1 — Overview

Context. In the previous chapter, we argued why looking at slightly simplified models where the pirate strategy is completely known to the distributor may be a useful thing to do, and we asked two questions: Can we find optimal solutions in case the tracer knows exactly how the pirates will mix their copies? And what do 'optimal solutions' actually correspond to for the case of known attacks?

As the previous chapter dealt with the latter question, showing that the results for known pirate attacks have implications for the general setting as well, in this chapter we will look at the former question: Can we construct decoding schemes that provably achieve the capacities derived in the previous chapter for known pirate strategies? And can the results for these simpler models also be applied to the general setting where the pirate strategy is not assumed to be known to the distributor?

Known attacks. Using log-likelihood decoders, as motivated by the Neyman–Pearson lemma, we show that we can obtain provable bounds on the code lengths that asymptotically have the optimal scaling for any known pirate strategy. More precisely: given a pirate strategy, the simple log-likelihood decoder tailored against this pirate strategy will always achieve the optimal simple code rate corresponding to this attack, and we can provide explicit values of ℓ that provably satisfy given bounds on the false positive and false negative error probabilities.

Whereas the question is relatively easy to answer for simple decoders, and asymptotically optimal accusation schemes can be designed similar to Tardos' scheme [Tar03], finding tight bounds on the error probabilities for joint decoders seems harder. Although for most pirate strategies considered in the previous chapter we can show that the corresponding joint decoder achieves capacity, and we can show that with high probability we will always accuse the all-guilty tuple and never accuse the all-innocent tuple, proving that these joint decoders always accuse the right set of users with probability at least $1-\varepsilon$ for given c, n, ℓ is left as an open problem. The difficulty in proving this statement is best explained by the existence of mixed tuples, consisting of both innocent and guilty users, for which we are so far unable to prove that the tuple scores will always be below the given threshold.

Arbitrary attacks. Besides studying decoders designed against specific attacks, we also consider whether these decoders can be used against arbitrary attacks. Motivated

*This chapter is based on results from [Laa14, Laa15a].

by previous results of Abbe and Zheng [AZ10], we take a look at the simple and joint decoders designed against the interleaving attack, and show that both achieve an asymptotic code length of $\ell \sim 2c^2 \ln n$. The simple decoder designed against the interleaving attack corresponds to the following score function g, where $g(x, y, p)$ is the score assigned to a user who receives symbol x in a position where the colluders output symbol y and the bias is p:

$$g(x,y,p) = \begin{cases} \ln\left(1 + \frac{p}{c(1-p)}\right) & \text{if } x = y = 0; \\ \ln\left(1 - \frac{1}{c}\right) & \text{if } x \neq y; \\ \ln\left(1 + \frac{1-p}{cp}\right) & \text{if } x = y = 1. \end{cases} \tag{4.1}$$

For $0 \ll p \ll 1$ this decoder is asymptotically equivalent to the decoder of Oosterwijk–Škorić–Doumen [OŠD13], which achieves the optimal scaling of ℓ as well [OŠD15]. However, as we will argue, due to not making a Gaussian assumption and purely looking at the Neyman–Pearson motivated optimal decoder, we obtain a decoder which performs better for $p \approx 0, 1$, and in particular scores do not blow up for small p, allowing us to eliminate the cutoffs on the bias distribution function and use the arcsine distribution function without cutoffs. Choosing the code length ℓ and threshold η appropriately, we can then defend against unknown attack strategies with an asymptotic code length of $\ell \sim 2c^2 \ln n$.

Outline. The remainder of this chapter is devoted to proving the results mentioned above regarding optimal simple and joint decoding schemes. In Section 4.2 we study simple decoding schemes in case the attack is known to the tracer, and in Section 4.3 we do the same for joint decoders. Section 3.4 finally discusses the implications of these results for the uninformed setting, and how the decoders designed against the interleaving attack allow us to defend against arbitrary pirate strategies as well.

4.2 — Simple decoders

In this section we will discuss simple decoders with explicit scheme parameters (code lengths, accusation thresholds) that provably satisfy given bounds on the error probabilities. The asymptotics of the resulting code lengths further show that these schemes are capacity-achieving; asymptotically, the code lengths achieve the lower bounds that follow from the simple capacities, as derived in Chapter 3.

We will follow the score-based framework introduced by Tardos [Tar03]. For simple decoding, this means that a user j receives a score S_j of the form

$$S_j = \sum_{i=1}^{\ell} S_{j,i} = \sum_{i=1}^{\ell} g(x_{j,i}, y_i, p_i), \tag{4.2}$$

and he is accused iff $S_j \geqslant \eta$ for some fixed threshold η. The function g is sometimes called the score function. Note that since g depends on $\mathcal{X}$ only via $\mathbf{X}_j$, any decoder that fits this framework is a simple decoder.

4.2.1 – Simple log-likelihood decoders. Several different score functions have been considered before [Laa13a, OŠD15, ŠKC08, Tar03], but in this chapter we will restrict our

attention to log-likelihood scores, which are known to perform well and which turn out to be quite easy to analyze for known pirate strategies.

First, when building a decoder we naturally want to be able to distinguish between two cases: user j is guilty or user j is not guilty. To do this, we assign scores to users based on the available data, and we try to obtain an optimal trade-off between the false positive error (accusing an innocent user) and the false negative error (not accusing a guilty user). This problem is well known in statistics as a hypothesis testing problem, where in this case we want to distinguish between the following two hypotheses H_0 and H_1:

$$H_0: \quad \text{user } j \text{ is innocent } (j \notin \mathcal{C}), \tag{4.3}$$

$$H_1: \quad \text{user } j \text{ is guilty } (j \in \mathcal{C}). \tag{4.4}$$

The Neyman–Pearson lemma [NP33] tells us that the most powerful test to distinguish between H_0 and H_1 is to test whether the following likelihood ratio exceeds an appropriately chosen threshold η'':

$$\Lambda(\mathbf{x}, \mathbf{y}, \mathbf{p}) = \frac{f_{H|\mathbf{X},\mathbf{Y},\mathbf{P}}(H_1|\mathbf{x}, \mathbf{y}, \mathbf{p})}{f_{H|\mathbf{X},\mathbf{Y},\mathbf{P}}(H_0|\mathbf{x}, \mathbf{y}, \mathbf{p})}. \tag{4.5}$$

Here $f_{H|\mathbf{X},\mathbf{Y},\mathbf{P}}(H_1|\mathbf{x}, \mathbf{y}, \mathbf{p})$ denotes the probability that user j is guilty, based on the evidence $\mathbf{x}, \mathbf{y}, \mathbf{p}$. Using Bayes' theorem, this can be rewritten to

$$\Lambda(\mathbf{x}, \mathbf{y}, \mathbf{p}) = \frac{f_{H|\mathbf{P}}(H_1|\mathbf{p})}{f_{H|\mathbf{P}}(H_0|\mathbf{p})} \cdot \frac{f_{\mathbf{X},\mathbf{Y}|\mathbf{P},H}(\mathbf{x}, \mathbf{y}|\mathbf{p}, H_1)}{f_{\mathbf{X},\mathbf{Y}|\mathbf{P},H}(\mathbf{x}, \mathbf{y}|\mathbf{p}, H_0)}. \tag{4.6}$$

Taking logarithms, and assuming that different positions i are i.i.d., it is clear that testing whether a user's likelihood ratio exceeds η'' is equivalent to testing whether his score S_j exceeds $\eta' = \ln \eta''$ for $\hat{g}$ defined as

$$\hat{g}(x, y, p) = \ln\left(\frac{f_H(H_1)}{f_H(H_0)}\right) + \ln\left(\frac{f_{X,Y|P}(x, y|p, H_1)}{f_{X,Y|P}(x, y|p, H_0)}\right). \tag{4.7}$$

Alternatively, assuming that the a priori probabilities $f_H(H_1) = \mathbb{P}(j \in \mathcal{C})$ and $f_H(H_0) = \mathbb{P}(j \notin \mathcal{C})$ do not depend on j, and noting that scalings and translations of the score function do not affect its performance, this is equivalent to testing whether a user's score S_j exceeds $\eta = \eta' - C$ (for some constant C) for the score function g defined as

$$g(x, y, p) = \ln\left(\frac{f_{X,Y|P}(x, y|p, H_1)}{f_{X,Y|P}(x, y|p, H_0)}\right). \tag{4.8}$$

Thus, the score function g from (4.8) corresponds to using a Neyman–Pearson score over the entire code word $\mathbf{X}_j$, and therefore g is in a sense optimal for minimizing the false positive error for a fixed false negative error. Score functions of this form were previously considered in the context of fingerprinting in e.g. [MF11a, PFF09].

4.2.2 – Theoretical evaluation. Let us investigate how we can choose ℓ and η so that we can prove that the false positive and false negative error probabilities are bounded from above by certain values ε_0 and ε_1. For the analysis below, we will make use of

the following function M, which is closely related to the moment-generating function of scores in one position i for both innocent and guilty users. For fixed p, this function M is defined on $[0,1]$ as

$$M(t) = \sum_{x,y} f_{X,Y|P}(x,y|p,H_1)^t f_{X,Y|P}(x,y|p,H_0)^{1-t}. \tag{4.9}$$

This function satisfies $M(t) = \mathbb{E}(e^{tS_{j,i}} \mid p, H_0) = \mathbb{E}(e^{(t-1)S_{j,i}} \mid p, H_1)$.

Theorem 4.1. *Let p and θ be fixed and known to the decoder. Let $\gamma = \ln(1/\varepsilon_1)/\ln(n/\varepsilon_0)$, and let the code length ℓ and threshold η be chosen as*

$$\ell = \frac{\sqrt{\gamma}(1+\sqrt{\gamma}-\gamma)}{-\ln M(1-\sqrt{\gamma})} \ln\left(\frac{n}{\varepsilon_0}\right), \qquad \eta = (1-\gamma)\ln\left(\frac{n}{\varepsilon_0}\right). \tag{4.10}$$

Then with probability at least $1-\varepsilon_0$ no innocent users are accused, and with probability at least $1-\varepsilon_1$ at least one colluder is caught.

Proof. For innocent users j, we would like to prove that $\mathbb{P}(j \text{ is accused}) = \mathbb{P}(S_j > \eta \mid p, H_0) \leqslant \frac{\varepsilon_0}{n}$, where S_j is the user's total score over all segments. If this can be proved, then since innocent users have independent scores, it follows that with probability at least $(1-\frac{\varepsilon_0}{n})^n \geqslant 1-\varepsilon_0$ no innocent users are accused. We start by applying the Markov inequality to $e^{\alpha S_j}$ for some $\alpha > 0$:

$$\mathbb{P}(S_j > \eta \mid p, H_0) = \mathbb{P}(e^{\alpha S_j} > e^{\alpha\eta} \mid p, H_0) \leqslant \frac{\mathbb{E}(e^{\alpha S_j} \mid p, H_0)}{e^{\alpha\eta}} = \frac{M(\alpha)^\ell}{e^{\alpha\eta}}. \tag{4.11}$$

For a guilty user j, we would like to prove that $\mathbb{P}(j \text{ is not accused}) = \mathbb{P}(S_j < \eta \mid p, H_1) \leqslant \varepsilon_1$. If we can prove this for an arbitrary colluder j, then clearly with high probability we will accuse at least one of the traitors. Using Markov's inequality with a fixed constant $\beta > 0$, we analogously get

$$\mathbb{P}(S_j < \eta \mid p, H_1) = \mathbb{P}(e^{-\beta S_j} > e^{-\beta\eta} \mid p, H_1) \leqslant \frac{\mathbb{E}(e^{-\beta S_j} \mid p, H_1)}{e^{-\beta\eta}} = \frac{M(1-\beta)^\ell}{e^{-\beta\eta}}. \tag{4.12}$$

For both guilty and innocent users, we can now obtain bounds by choosing appropriate values for α and β. Investigating the resulting expressions, we observe that good choices for α, β leading to sharp bounds are $\alpha = 1-\sqrt{\gamma}$ and $\beta = \sqrt{\gamma}$. Substituting these choices for α and β, and setting the bounds equal to the desired upper bounds $\frac{\varepsilon_0}{n}$ and ε_1, we get

$$\mathbb{P}(S_j > \eta \mid p, H_0) \leqslant \frac{M(1-\sqrt{\gamma})^\ell}{e^{-\sqrt{\gamma}\eta}} e^{-\eta} = \frac{\varepsilon_0}{n}, \tag{4.13}$$

$$\mathbb{P}(S_j < \eta \mid p, H_1) \leqslant \frac{M(1-\sqrt{\gamma})^\ell}{e^{-\sqrt{\gamma}\eta}} = \varepsilon_1. \tag{4.14}$$

Combining these equations we obtain the value η given in (4.10), and solving for ℓ (using e.g. that $\eta = (1-\gamma)\ln(n/\varepsilon_0)$) then leads to the given expression for ℓ of (4.10). □

Compared to previous papers analyzing provable bounds on the error probabilities, such as [BT08,IŠO14,LdW14,ŠKC08,ŠVCT08,Tar03], the proof of Theorem 4.1 is remarkably short and simple. Note however that the proof that a colluder is caught assumes that the attack used by the colluders is the same as the one the decoder is built against, and that the actual value of ℓ is still somewhat mysterious due to the term $M(1-\sqrt{\gamma})$.

4.2.3 – Practical evaluation. Before describing how the code lengths of Theorem 4.1 scale with n and c, note that Theorem 4.1 only shows that with high probability we provably catch *at least one* colluder with this decoder. Although this is commonly the best you can hope for when dealing with arbitrary attacks in fingerprinting[1], if the attack is colluder-symmetric it is actually possible to catch *all* colluders with high probability. So instead we would like to be able to claim that with high probability, the set of accused users *equals* the set of colluders. Similar to the proof for innocent users, we could simply replace ε_1 by $\frac{\varepsilon_1}{c}$, and argue that the probability of finding all pirates is the product of their individual probabilities of getting caught, leading to a lower bound on the success probability of $(1-\frac{\varepsilon_1}{c})^c \geqslant 1-\varepsilon_1$. This leads to the following heuristic estimate for the code length required to catch *all* colluders.

Heuristic 4.2. *Let γ in Theorem 4.1 be replaced by $\gamma' = \ln(c/\varepsilon_1)/\ln(n/\varepsilon_0)$. Then with probability at least $1-\varepsilon_0$ no innocent users are accused, and with probability at least $1-\varepsilon_1$ all colluders are caught.*

The problem with this claim is that the pirate scores are related through $\mathbf{Y}$, and so they are not independent. As a result, we cannot simply take the product of the individual probabilities $(1-\frac{\varepsilon_1}{c})$ to get a lower bound on the success probability of $1-\varepsilon_1$. On the other hand, especially when the code length ℓ is large and ε_1 is small, we do not expect the event $\{S_j > \eta_0\}$ to tell us much about the probability of $\{S_{j'} > \eta_0\}$ occurring for $j' \neq j$. One might expect that $\{S_{j'} > \eta_0\}$ does not become much less likely when $\{S_j > \eta_0\}$ occurs. Proving a rigorous upper bound on the *catch-all* error probability is left for future work.

4.2.4 – Asymptotic code lengths. Let us now study how the code lengths from (4.10) scale in terms of c and n. Focusing on the regime of large n (and fixed ε_0 and ε_1), it turns out that the code length always has the optimal asymptotic scaling, regardless of p and θ.

Theorem 4.3. *For large n and fixed ε_0 and ε_1, the code length ℓ of Theorem 4.1 scales as*

$$\ell(p,\theta) = \frac{\log_2 n}{I^s(p,\theta)}[1+O(\sqrt{\gamma})], \tag{4.15}$$

where $I^s(p,\theta)$ is the mutual information pay-off defined in (3.4)*. As $\gamma = o(1)$ for large n and fixed $\varepsilon_0, \varepsilon_1$, we conclude that ℓ has the optimal asymptotic scaling.*

Proof. Let us first study the behavior of $M(1-\sqrt{\gamma})$ for small γ, by computing the first order Taylor expansion of $M(1-\sqrt{\gamma})$ around $\gamma = 0$. For convenience, below we will abbreviate $f_{X,Y|P}(x,y|p,H_i)$ by $f(x,y|p,H_i)$.

$$M(1-\sqrt{\gamma}) = \sum_{x,y} f(x,y \mid p,H_1) \exp\left[-\sqrt{\gamma}\ln\left(\frac{f(x,y \mid p,H_1)}{f(x,y \mid p,H_0)}\right)\right] \tag{4.16}$$

$$= \sum_{x,y} f(x,y|p,H_1)\left[1-\sqrt{\gamma}\ln\left(\frac{f(x,y|p,H_1)}{f(x,y|p,H_0)}\right)+O(\gamma)\right]. \tag{4.17}$$

[1] In those cases attacks exist guaranteeing you will not catch more than one colluder, such as the 'scapegoat' strategy [LDR+13].

Here the second equality follows from the fact that if $f(x, y \mid p, H_1) = 0$, then the factor $f(x, y \mid p, H_1)$ in front of the exponentiation causes this term to be 0, while if $f(x, y \mid p, H_1) > 0$, then also $f(x, y \mid p, H_0) > 0$ and thus their ratio is bounded and does not depend on γ. Next, recognizing the first order term as the mutual information (in natural units) between a colluder symbol X_1 and the pirate output Y, we obtain:

$$M(1-\sqrt{\gamma}) = 1 - \sqrt{\gamma} \sum_{x,y} f(x, y|p, H_1) \ln\left(\frac{f(x, y|p, H_1)}{f(x, y|p, H_0)}\right) + O(\gamma) \tag{4.18}$$

$$= 1 - I^s(p, \theta)\sqrt{\gamma}\ln 2 + O(\gamma). \tag{4.19}$$

Substituting this result in the original equation for ℓ, and noting that the factor ε_0 inside the logarithm is negligible for large n, we finally obtain the result of (4.15). □

Note that in the discussion above, we did not make any assumptions on p. In fact, both Theorems 4.1 and 4.3 hold for *arbitrary* values of p; the decoder always achieves the optimal code rate associated to that value of p. As a result, if we optimize and fix p based on θ (using results from Chapter 3), we automatically end up with a decoder that provably achieves capacity for this attack.

4.2.5 – Explicit attacks. Let us now consider specific pirate attacks which are often considered in the fingerprinting literature, and investigate the resulting code lengths. We will consider the same five attacks as in Chapter 3, which were also considered in e.g. [CXFF09, FPF09a, HM12b, Laa13a, MF11a, MF12, OŠD15]. Using Theorem 4.1 we can obtain exact, provable expressions for the code length ℓ in terms of $\theta, p, c, n, \varepsilon_0, \varepsilon_1$. Performing a Taylor series expansion around $c = \infty$ for the optimal values of p from Chapter 3 we obtain the following expressions for ℓ in the optimal points p. Note that $\ell(\theta_{\text{min}}) \sim \ell(\theta_{\text{all1}})$, and that these expressions are more precise than those of Theorem 4.3.

$$\ell(\theta_{\text{int}}) = 2c^2 \ln\left(\frac{n}{\varepsilon_0}\right)\left[\frac{1+\sqrt{\gamma}-\gamma}{1-\sqrt{\gamma}} + O\left(\frac{1}{c^2}\right)\right], \tag{4.20}$$

$$\ell(\theta_{\text{all1}}) = \frac{c \ln\left(\frac{n}{\varepsilon_0}\right)}{\ln(2)^2}\left[\frac{\sqrt{\gamma}\left(1+\sqrt{\gamma}-\gamma\right)\ln 2}{1-2^{-\sqrt{\gamma}}} + O\left(\frac{1}{c}\right)\right], \tag{4.21}$$

$$\ell(\theta_{\text{maj}}) = \pi c \ln\left(\frac{n}{\varepsilon_0}\right)\left[\frac{1+\sqrt{\gamma}-\gamma}{1-\sqrt{\gamma}} + O\left(\frac{1}{c}\right)\right], \tag{4.22}$$

$$\ell(\theta_{\text{coin}}) = \frac{4c}{\ln(2)^2} \ln\left(\frac{n}{\varepsilon_0}\right)\left[\frac{\sqrt{\gamma}\left(1+\sqrt{\gamma}-\gamma\right)\ln 2}{2-\left(1+\frac{1}{\sqrt{2}}\right)^{\sqrt{\gamma}}-\left(1-\frac{1}{\sqrt{2}}\right)^{\sqrt{\gamma}}} + O\left(\frac{1}{c}\right)\right]. \tag{4.23}$$

If we assume that both $c \to \infty$ and $\gamma \to 0$, then we can further simplify the above expressions for the code lengths. The first terms between brackets all scale as $1 + O(\sqrt{\gamma})$, so the leading terms of the code lengths are those terms outside the brackets.

4.3 — Joint decoders

In this section we will discuss informed joint decoders which we conjecture are able to find colluders with shorter code lengths than simple decoders. The asymptotics of the

resulting code lengths motivate why these schemes seem to be optimal, but some open problems remain for proving that they are indeed optimal.

Following the score-based framework for joint decoders of Moulin [Mou08], to tuples $\mathcal{T}$ of size c we assign a score of the form

$$S_{\mathcal{T}} = \sum_{i=1}^{\ell} S_{\mathcal{T},i} = \sum_{i=1}^{\ell} g(x_{\mathcal{T},i}, y_i, p_i). \qquad (x_{\mathcal{T},i} = \{x_{j,i} : j \in \mathcal{T}\}) \tag{4.24}$$

Note that if θ is colluder-symmetric, then this is equivalent to

$$S_{\mathcal{T}} = \sum_{i=1}^{\ell} g(z_{\mathcal{T},i}, y_i, p_i), \tag{4.25}$$

where $z_{\mathcal{T},i} = \sum_{j\in\mathcal{T}} x_{j,i}$ is the tally of the number of ones received by the tuple $\mathcal{T}$ in position i. For the accusation phase, we now accuse all users in $\mathcal{T}$ iff $S_{\mathcal{T}} \geqslant \eta$ for some fixed threshold η. Note that this accusation algorithm is not exactly well-defined, since it is possible that a user appears both in a tuple that is accused and in a tuple that is not accused. For the analysis we will assume that the scheme is only successful if the single tuple consisting of all colluders has a score exceeding η and no other tuples have a score exceeding η, in which case all users in that guilty tuple are accused.

4.3.1 – Joint log-likelihood decoders. For building a joint decoder we would like to be able to distinguish between the all-guilty tuple and other tuples, so a natural generalization of the hypotheses H_0 and H_1 for simple decoding would be $H_0 : \mathcal{T} \neq \mathcal{C}$ and $H_1 : \mathcal{T} = \mathcal{C}$. However, with this choice of H_0, computing probabilities $f_{Z,Y|P,H}(z, y \mid p, H_0)$ is complicated: the event H_0 does not completely determine $f_{Y|Z}(y|z)$ without making further assumptions on the a priori probabilities of users being colluders, since this probability depends on exactly how many colluders are present in $\mathcal{T}$. To be able to compute the likelihood ratios, we therefore use the following two hypotheses, which were also used in e.g. [MF12]:

$$H_0 : \quad \text{all users } j \in \mathcal{T} \text{ are innocent } (\mathcal{T} \cap \mathcal{C} = \varnothing), \tag{4.26}$$

$$H_1 : \quad \text{all users } j \in \mathcal{T} \text{ are guilty } (\mathcal{T} = \mathcal{C}). \tag{4.27}$$

We again use the corresponding log-likelihood ratio per position as our score function g, which after rewriting is again equivalent to using the logarithm of the likelihood ratio over all positions i as our score function:

$$g(z, y, p) = \ln\left(\frac{f_{Z,Y|P,H}(z, y \mid p, H_1)}{f_{Z,Y|P,H}(z, y \mid p, H_0)}\right). \tag{4.28}$$

Using this joint score function g corresponds to the most powerful test according to the Neyman–Pearson lemma [NP33], so g is in a sense optimal for distinguishing between H_0 and H_1.

4.3.2 – Theoretical evaluation. Let us again study how to choose ℓ and η so that we can prove that the false positive and false negative error probabilities are bounded from above by certain values ε_0 and ε_1. Below we will again make use of the function M of (4.9) where the simple hypotheses H_0 and H_1 have been replaced by the joint hypotheses H_0 and H_1.

Theorem 4.4. *Let p and θ be fixed and known to the decoder. Let $\gamma = \ln(1/\varepsilon_1)/\ln(n^c/\varepsilon_0)$, and let the code length ℓ and the threshold η be chosen as*

$$\ell = \frac{\sqrt{\gamma}(1+\sqrt{\gamma}-\gamma)}{-\ln M(1-\sqrt{\gamma})} \ln\left(\frac{n^c}{\varepsilon_0}\right), \qquad \eta = (1-\gamma)\ln\left(\frac{n^c}{\varepsilon_0}\right). \tag{4.29}$$

Then with probability at least $1-\varepsilon_0$ none of the all-innocent tuples are accused, and with probability at least $1-\varepsilon_1$ the single all-guilty tuple is accused.

Proof. The proof is analogous to the proof of Theorem 4.1. Instead of n innocent and c guilty users we now have $\binom{n}{c} < n^c$ all-innocent tuples and just 1 all-guilty tuple, which changes some of the numbers in γ, η and ℓ above. We then again apply the Markov inequality with $\alpha = 1-\sqrt{\gamma}$ and $\beta = \sqrt{\gamma}$ to obtain the given expressions for η and ℓ. □

We remark that for deterministic strategies $\theta \in \{0,1\}^{c+1}$, choosing the scheme parameters is much simpler. Similar to Lemma 3.7, which showed that for deterministic attacks the joint capacity is exactly $\frac{1}{c}$, in this case we get a code length of $\ell \sim c \log_2 n$.

Theorem 4.5. *Let θ be a deterministic attack, and let p be chosen such that $f_{Y|P}(1|p) = \frac{1}{2}$. Let ℓ and η be chosen as:*

$$\ell = \log_2\left(\frac{n^c}{\varepsilon_0}\right), \qquad \eta = \ln\left(\frac{n^c}{\varepsilon_0}\right). \tag{4.30}$$

Then with probability $1-\varepsilon_0$ none of the all-innocent tuples are accused, and the single all-guilty tuple will always be accused.

Proof. For deterministic attacks, we have

$$f_{Z,Y|P,H}(z,y \mid p, H_0) = f_{Z|P}(z \mid p) f_{Y|P}(y \mid p), \tag{4.31}$$
$$f_{Z,Y|P,H}(z,y \mid p, H_1) = f_{Z|P}(z \mid p) f_{Y|Z}(y \mid z). \tag{4.32}$$

As a result, the score function g satisfies

$$g(z,y,p) = \begin{cases} -\ln f_{Y|P}(y|p) & \text{if } \theta_z = y; \\ -\infty & \text{if } \theta_z \neq y. \end{cases} \tag{4.33}$$

With the capacity-achieving choices of p of Lemma 3.7, we always have $f_{Y|P}(y|p) = \frac{1}{2}$ for $y = 0, 1$ leading to a score of $+\ln 2$ for a match, and $-\infty$ for cases where y_i does not match the output that follows from θ and the assumption that $\mathcal{T}$ is the all-guilty tuple. For $\mathcal{T} = \mathcal{C}$, clearly we will always have a match, so this tuple's score will always be $\ell \ln 2 = \eta$, showing that this tuple is always accused.

For innocent tuples, since $f(z,y|p,H_0) = \frac{1}{2} f(z,y|p,H_1)$ it follows that in each position i, with probability $\frac{1}{2}$ this tuple's score will not be $-\infty$. So with probability $2^{-\ell}$, the tuple's score after ℓ segments will not be $-\infty$, in which case it equals $\ell \ln 2$. To make sure that this probability is at most ε_0/n^c so that the total error probability is at most ε_0, we set $2^{-\ell} = \varepsilon_0/n^c$, leading to the given expression for ℓ. □

Note that for deterministic attacks, any choice of $-\infty < \eta_0 \leqslant \eta$ works just as well as choosing η; after ℓ segments all tuples will either have a score of $-\infty$ or η.

4.3.3 – Practical evaluation. Theorems 4.4 and 4.5 do not prove that we can actually find the set of colluders with high probability, since mixed tuples (consisting of some innocent and some guilty users) also exist and these may or may not have a score exceeding η. Theorem 4.4 only proves that with high probability we can find a set $\mathcal{C}'$ of c users which contains at least one colluder. We expect that in most cases the only tuple with a score exceeding η is the all-guilty tuple and all mixed tuples will have a score below η. Proving that mixed tuples indeed get a lower score is left as an open problem.

4.3.4 – Asymptotic code lengths. To further motivate why using this joint decoder may be the right choice, the following proposition shows that at least the resulting code lengths are optimal. The proof is analogous to the proof of Theorem 4.3.

Theorem 4.6. *For $\gamma = o(1)$, the code length ℓ of Theorem 4.4 scales as*

$$\ell = \frac{\log_2 n}{I^j(p,\theta)}\left[1 + O(\sqrt{\gamma})\right], \tag{4.34}$$

thus asymptotically achieving the joint capacity for arbitrary values of p.

Since the asymptotic code length is optimal regardless of p, these asymptotics are also optimal when p is optimized to maximize the mutual information.

Finally, although it is hard to estimate the scores of mixed tuples with this decoder, just like in [ODL14] we expect that the joint decoder score for a tuple is roughly equal to the sum of the c individual simple decoder scores. So a tuple of c users consisting of k colluders and $c-k$ innocent users is expected to have a score roughly a factor k/c smaller than the expected score for the all-guilty tuple. So after computing the scores for all tuples of size c, we can get rough estimates of how many guilty users are contained in each tuple, and we might try to find the set $\mathcal{C}'$ of c users that best matches these estimates. There are several options for post-processing that may improve the accuracy of using this joint decoder, which are left for future work.

4.3.5 – Explicit attacks. Using Theorem 4.4 we can obtain exact expressions for ℓ in terms of $\theta, p, c, n, \varepsilon_0, \varepsilon_1$ for known attack strategies. For the optimal values of p of Chapter 3 we can use Theorem 4.6 to obtain the following expressions. Note again that $\ell(\theta_{\min}) \sim \ell(\theta_{\text{all1}})$, and β is described in Chapter 3.

$$\ell(\theta_{\text{int}}) = \frac{c^2 \log_2 n}{\beta}\left[1 + O\left(\sqrt{\gamma} + \frac{1}{c}\right)\right], \tag{4.35}$$

$$\ell(\theta_{\text{all1}}) = c \log_2 n\left[1 + O\left(\sqrt{\gamma} + \frac{1}{c}\right)\right], \tag{4.36}$$

$$\ell(\theta_{\text{maj}}) = c \log_2 n\left[1 + O\left(\sqrt{\gamma} + \frac{1}{c}\right)\right], \tag{4.37}$$

$$\ell(\theta_{\text{coin}}) = c \log_{5/4} n\left[1 + O\left(\sqrt{\gamma} + \frac{1}{c}\right)\right]. \tag{4.38}$$

A code length with an explicit dependence on γ (and order terms only in c) can be obtained as well using Taylor expansions of $M(1-\sqrt{\gamma})$.

4.4 — Arbitrary attacks

While the previous sections described simple decoders designed against specific pirate strategies, let us now return to the setting where θ is not known to the tracer.

4.4.1 – The simple interleaving decoder. Let us first investigate how we might build a decoder that works against arbitrary attacks in fingerprinting. To build such a decoder, Meerwald and Furon [MF12] previously noted that Abbe and Zheng [AZ10] proved in a more general context that under certain conditions on the set of allowed pirate strategies $\mathcal{P}^c$, a decoder that works against the worst-case attack[2] $\theta^* \in \mathcal{P}^c$ actually defends against all attacks $\theta \in \mathcal{P}^c$. Recall that in this case the set of allowed pirate strategies we consider is the set of all attacks satisfying the marking assumption $\mathcal{P}^c = \mathcal{P}_{\text{mark}} = \{\theta \in [0,1]^{c+1} \mid \theta_0 = 0, \theta_c = 1\}$.

For finite c, the worst-case attack in fingerprinting (from an information-theoretic perspective) has been studied in e.g. [HM12b, MF12], but in general this attack is quite hard to describe analytically. Since this attack is not so easy to analyze, let us therefore focus on the regime of large c and n. Huang and Moulin [HM12b] previously proved that for large coalitions, the optimal pirate attack in the simple decoding setting is the interleaving attack. Combining this knowledge with the result of Abbe and Zheng, one could speculate that a good choice for a universal decoder is the interleaving decoder, designed against the interleaving attack. Working out the likelihood ratio in case $\theta = \theta_{\text{int}}$, we obtain the following score function g:

$$g(x,y,p) = \begin{cases} \ln\left(1 + \frac{p}{c(1-p)}\right) & \text{if } x = y = 0; \\ \ln\left(1 - \frac{1}{c}\right) & \text{if } x \neq y; \\ \ln\left(1 + \frac{1-p}{cp}\right) & \text{if } x = y = 1. \end{cases} \tag{4.39}$$

Let us take a closer look at this decoder. For fixed $\delta > 0$, if we look at values $p \in [\delta, 1-\delta]$ and focus on the regime of large c, we can perform a Taylor series expansion around $c = \infty$ to get $\ln(1+x) \sim x$. The resulting expressions then turn out to be closely related to the Oosterwijk–Škorić–Doumen [OŠD15] decoder h:

$$c \cdot g(x,y,p) \sim h(x,y,p) = \begin{cases} +\frac{p}{1-p} & \text{if } x = y = 0; \\ -1 & \text{if } x \neq y; \\ +\frac{1-p}{p} & \text{if } x = y = 1. \end{cases} \tag{4.40}$$

This implies that g and h are asymptotically equivalent in case p is sufficiently far away from 0 and 1. Since for Oosterwijk–Škorić–Doumen's score function one generally uses *cutoffs* on f_P (i.e. only using values $p \in [\delta, 1-\delta]$ for fixed $\delta > 0$) to guarantee that $h(x,y,p) = o(c)$ (cf. [IŠO14]), and since the decoder [OŠD15] is known to achieve capacity using these cutoffs, we immediately get the following result.

[2] Here, the worst-case attack is defined in an information-theoretic sense as the attack minimizing the mutual information pay-off function for given c.

Proposition 4.7. *The score function* g *of* (4.39) *together with the bias density function (encoder)* $f_P^{(\delta)}$ *on* $[\delta, 1-\delta]$ *of the form*

$$f_P^{(\delta)}(p) = \frac{1}{(\pi - 4\arcsin\sqrt{\delta})\sqrt{p(1-p)}} \tag{4.41}$$

asymptotically achieve the simple capacity for uninformed fingerprinting when the same cutoffs δ *as those of [IŠO14] are used.*

So combining the log-likelihood decoder tuned against the asymptotic worst-case attack (the interleaving attack) with the arcsine distribution with cutoffs, we obtain a universal decoder that works against arbitrary attacks.

Cutting off the cutoffs. While Proposition 4.7 shows that when using cutoffs, the log-likelihood decoder designed against the interleaving attack achieves capacity in the simple fingerprinting model, the cutoffs δ have been a nagging inconvenience ever since Tardos introduced them in 2003 [Tar03]. In previous settings it was well-known that this cutoff δ had to be large enough to guarantee that innocent users are not falsely accused, and small enough to guarantee that large coalitions can still be caught. For instance, when using Tardos' original score function, it was impossible to do without cutoffs, and the same seems to hold for the decoder h as scores blow up for $p \approx 0, 1$.

Looking at the universal log-likelihood decoder in (4.39), one thing to notice is that the logarithm has a mitigating effect on the tails of the score distributions (cf. [Ško15]). For $0 \ll p \ll 1$ the resulting scores are roughly a factor c smaller than those obtained with h, but where the blowup effect of h for small p is proportional to $\frac{1}{p}$, the function g only scales as $\ln(\frac{1}{p})$ in the region of small p. This motivates the following claim, showing that with this decoder g we finally do not need any cutoffs anymore!

Theorem 4.8. *The decoder* g *of* (4.39) *and the arcsine distribution encoder* $f_P^*(p)$*, defined on* $[0, 1]$ *by*

$$f_P^*(p) = \frac{1}{\pi\sqrt{p(1-p)}}, \tag{4.42}$$

together asymptotically achieve the simple capacity for the uninformed fingerprinting game.

Proof. We will argue that using this new universal decoder g, the difference in performance between using and not using cutoffs on f_P is negligible for large c. Since the encoder with cutoffs asymptotically achieves capacity, it then follows that without cutoffs this scheme also achieves capacity. To do this, we will prove that all moments of innocent user scores are finite. In that case, for large c from the Central Limit Theorem it follows that the distributions of user scores will converge to Gaussians. If the scores of innocent and guilty users are indeed Gaussian for large c, then as discussed in e.g. [OŠD15, ŠKC08] all that matters for assessing the performance of the scheme are the mean and variance of both curves. Similar to (4.49), the effects of small cutoffs of the distribution function f_P on the distribution of user scores are negligible as both means and variances stay the same up to small order terms. So indeed, in both cases the 'performance indicator' [OŠD15] asymptotically stays the same, leading to equivalent code lengths with and without cutoffs.

So let us prove that all moments of user scores are finite, even if no cutoffs are used. We will show that $\mathbb{E}[g(x,y,p)^k] < \infty$ for any x and y, so that after taking weighted combinations we also get $\mathbb{E}(S^k_{j,i}|H_{0/1}) < \infty$. Let us consider the case where $x = y = 1$; other cases can be analyzed in a similar fashion. Using the density function f^*_p of (4.42) and the function g from (4.39), we have

$$E_k = \mathbb{E}[g(1,1,p)^k] = \int_0^1 \frac{dp}{\pi\sqrt{p(1-p)}} \log^k\left(1 + \frac{1-p}{cp}\right). \tag{4.43}$$

Splitting the interval $[0,1]$ into two parts $[\kappa,1]$ and $[0,\kappa]$ (where κ depends on k but not on c) we obtain

$$E_k = \int_\kappa^1 \frac{dp}{\pi\sqrt{p(1-p)}} \log^k\left(1 + \frac{1-p}{cp}\right) + \int_0^\kappa \frac{dp}{\pi\sqrt{p(1-p)}} \log^k\left(1 + \frac{1-p}{cp}\right). \tag{4.44}$$

Let us denote the two terms by $E_{k,1}$ and $E_{k,2}$ respectively. For the first term, we can perform a Taylor series expansion to obtain:

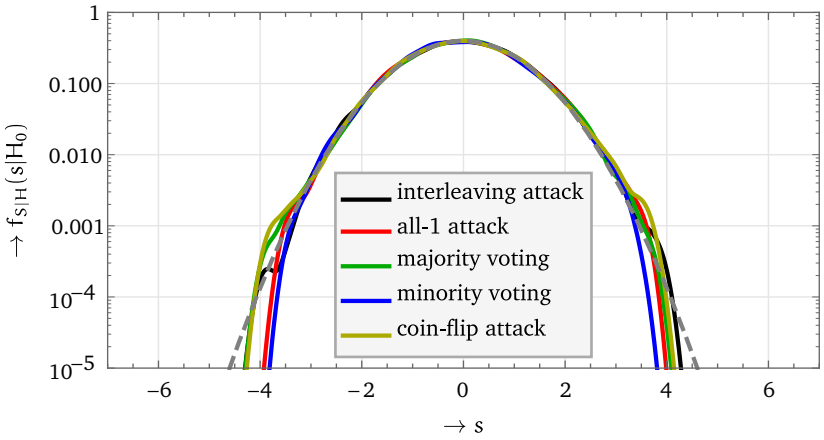
$$\begin{aligned}
E_{k,1} &= \int_\kappa^1 \frac{dp}{\pi\sqrt{p(1-p)}} \log^k\left(1 + \frac{1-p}{cp}\right) && (4.45)\\
&= \int_\kappa^1 \frac{dp}{\pi\sqrt{p(1-p)}} \left(\frac{1-p}{cp} + O\left(\frac{(1-p)^2}{c^2p^2}\right)\right)^k && (4.46)\\
&\leqslant \int_\kappa^1 \frac{dp}{\pi\sqrt{p(1-p)}} \left(\frac{1}{c\kappa} + O\left(\frac{1}{c^2\kappa^2}\right)\right)^k && (4.47)\\
&\overset{(a)}{\leqslant} \int_\kappa^1 \frac{dp}{\pi\sqrt{p(1-p)}} < 1 < \infty. && (4.48)
\end{aligned}$$

Here (a) follows from considering sufficiently large c while κ remains fixed. (Note that for large c we even have $E_{k,1} \to 0$.) For the other term we do not expand the logarithm:

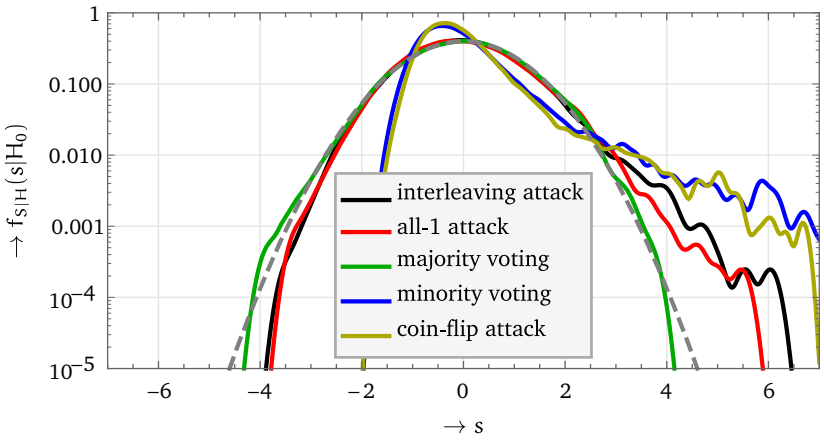
$$E_{k,2} = \int_0^\kappa \frac{dp}{\pi\sqrt{p(1-p)}} \log^k\left(1 + \frac{1-p}{cp}\right) \propto \int_0^\kappa \frac{dp}{\sqrt{p}} \log^k\left(\frac{1}{p}\right) \overset{(b)}{\to} 0. \tag{4.49}$$

The last step (b) follows from the fact that the integration is done over an interval of width κ, while the integrand scales as $\frac{1}{\sqrt{p}}$ times less important logarithmic terms. For arbitrary k, we can thus let $\kappa = \kappa(k) \to 0$ as a function of k to see that this is always bounded. Similar arguments can be used to show that for other values of x, y we also have $\mathbb{E}[g(x,y,p)^k] < 0$. □

Note that the same result does *not* apply to the score function h of Oosterwijk–Škorrć–Doumen [OŠD15], for which the effects of values $p \approx 0, 1$ are not negligible. The main difference is that for small p, the score function h scales as $\frac{1}{p}$, while the log-likelihood decoder g only scales as $\ln(1/p)$. Figure 4.1 illustrates the difference in the convergence of normalized innocent user scores to the standard normal distribution, when using the score functions g and h against various different pirate strategies. These are experimental



(a) The PDF $f_{S|H}(s|H_0)$ of cumulative innocent user scores using the score function g



(b) The PDF $f_{S|H}(s|H_0)$ of cumulative innocent user scores using the score function h

Figure 4.1: Estimates of the probability density functions of normalized innocent user scores for $c = 10$ and $\ell = 10\,000$ based on 10 000 simulations of innocent user scores for each attack. The different lines correspond to the interleaving attack (black), all-1 attack (red), majority voting (green), minority voting (blue), and the coin-flip attack (yellow), while the gray dashed line corresponds to the standard normal distribution with $f_Z(z) \propto \exp(-\frac{1}{2}z^2)$. Note that the drops near densities of 10^{-4} (the far ends of the curves) occur due to smoothening the curve with only 10^4 samples, and do not accurately reflect the ends of the distribution tails.

results for $c = 10$ and $\ell = 10\,000$ based on 10 000 simulated scores for each curve, and for both score functions we did not use any cutoffs. As we can see, using the Neyman–Pearson decoder g, the normalized scores $\tilde{S}_j = (S_j - \mathbb{E}S_j)/\sqrt{\operatorname{Var} S_j}$ are almost Gaussian, while using h the curves especially do not appear to be Gaussian for $\tilde{S}_j \gg 0$; in most cases the distribution tails are much too large. For the minority voting and coin-flip attack, the curves do not even seem close to a normal distribution (cf. [Ško15]).

Designing a universal scheme. With the above result in mind, let us now briefly discuss how to actually build a universal decoding scheme with the interleaving decoder g which

defends against arbitrary attacks. From Theorem 4.8 we know that for generating biases we can simply use the arcsine distribution f_P^* without cutoffs, and our decoder will be the interleaving decoder g of (4.39). What remains is figuring out how to choose ℓ and η to defend against arbitrary attacks.

First, it is important to note that the expected innocent and guilty scores per segment ($\mu_0 = \mathbb{E}(S_{j,i}|H_0)$ and $\mu_1 = \mathbb{E}(S_{j,i}|H_1)$) and the variance of the innocent and guilty scores ($\sigma_0^2 = \mathrm{Var}(S_{j,i}|H_0)$ and $\sigma_1^2 = \mathrm{Var}(S_{j,i}|H_1)$) heavily depend on the collusion channel θ. This was not the case for Tardos' original decoder [Tar03] and the symmetrized decoder [ŠKC08], for which η could be fixed in advance regardless of the collusion strategy. This means that we will either have to delay fixing η until the decoding stage, or scale/translate scores per segment accordingly at each position i.

For choosing the code length ℓ and threshold η, let us focus on the regime of reasonably large c. In that case, as argued above the total innocent and guilty scores S_0 and S_1 will behave like Gaussians, with parameters $S_0 \sim \mathcal{N}(\ell\mu_0, \ell\sigma_0^2)$ and $S_1 \sim \mathcal{N}(\ell\mu_1, \ell\sigma_1^2)$. To distinguish between these two distributions, using e.g. Sanov's theorem the code rate $(\log_2 n)/\ell$ should be proportional to the Kullback–Leibler divergence between the two distributions:

$$d(S_1\|S_0) = \frac{(\mu_1-\mu_0)^2}{\sigma_0^2} + \frac{1}{2}\left(\frac{\sigma_1^2}{\sigma_0^2} - 1 - \ln\frac{\sigma_1^2}{\sigma_0^2}\right). \tag{4.50}$$

A similar expression appears in [ŠKC08, OŠD15], where it was noted that $\sigma_1 \ll \sigma_0$, so that the first term is the most important term. In [IŠO14, OŠD15] the ratio $\frac{(\mu_1-\mu_0)^2}{\sigma_0^2}$ was coined the 'performance indicator', and it was argued that this ratio should be maximized. In [OŠD15] it was further shown that when using their decoder h, this ratio is minimized by the colluders when they use the interleaving attack θ_{int}. In other words: assuming scores are Gaussian for large c, and assuming that the decoder score function is h, the best attack the traitors can use is the interleaving attac.

Since the decoder g is very similar to h (by $c \cdot g \approx h$), a natural conjecture would be that also for this new score function g, asymptotically the best pirate attack maximizing the decoder's error probabilities is the interleaving attack. Experiments with g and previous experiments of [OŠD15] with h indeed show that other pirate attacks generally perform worse than the interleaving attack. As a result, a natural choice for selecting ℓ would be to base ℓ on the code length needed to deal with the (asymptotic) worst-case attack for this decoder, the interleaving attack. And for the interleaving attack we know how to choose ℓ by Theorem 4.1, Equation 4.20 and Theorem 4.3:

$$\ell = \frac{\sqrt{\gamma}(1+\sqrt{\gamma}-\gamma)}{-\ln M(1-\sqrt{\gamma})}\ln\left(\frac{n}{\varepsilon_0}\right) \tag{4.51}$$

$$= 2c^2\ln\left(\frac{n}{\varepsilon_0}\right)\left[\frac{1+\sqrt{\gamma}-\gamma}{1-\sqrt{\gamma}} + O\left(\frac{1}{c^2}\right)\right] \tag{4.52}$$

$$= 2c^2\ln n\left[1 + O\left(\sqrt{\gamma}+\frac{1}{c^2}\right)\right]. \tag{4.53}$$

Here the expressions are given in order of precision; the first is exact (in case the interleaving attack is used), the second considers large-n asymptotics, and the third considers

both large-n and large-c asymptotics. These choices for ℓ seem reasonable estimates for the code lengths required to deal with arbitrary attacks.

For choosing η, as argued before this parameter depends on the pirate strategy θ, which may lead to different scalings and translations of the curves of innocent and guilty user scores. What we could do is compute the parameters $\mu_{0/1}, \sigma^2_{0/1}$ based on the pirate output y empirically and normalize the scores accordingly. This means that after computing user scores S_j, we apply the following transformation:

$$\tilde{S}_j = \frac{S_j - \ell\mu_0}{\sqrt{\ell}\sigma_0}. \tag{4.54}$$

This guarantees that the scores of innocent users will roughly be distributed as $\mathcal{N}(0,1)$, and for guilty users this results in a distribution of the form $\mathcal{N}(\sqrt{\ell}(\frac{\mu_1-\mu_0}{\sigma_0}), \frac{\sigma_1^2}{\sigma_0^2})$. To guarantee that the probability that an innocent user has a score below η is at least $1 - \frac{\varepsilon_0}{n}$, it then suffices to let

$$\tilde{\eta} = \frac{\eta - \ell\mu_0}{\sqrt{\ell}\sigma_0} \approx \Phi^{-1}\left(1 - \frac{\varepsilon_0}{n}\right) \sim \sqrt{\ln\left(\frac{n}{\varepsilon_0}\right)}, \tag{4.55}$$

where Φ denotes the cumulative distribution function of the standard normal distribution $\mathcal{N}(0,1)$. This means that after transforming the scores, the threshold can be fixed independently of the pirate strategy.

Another simple universal decoder. Besides Oosterwijk–Škorić–Doumen's approach using Lagrange multipliers, and the above Neyman–Pearson-based approach to obtaining efficient decoders, let us now mention a third way of obtaining a similar capacity-achieving universal decoder.

To construct this decoder, we start with the empirical mutual information decoder proposed by Moulin [Mou08], and for now let us assume $p_i \equiv p$ is fixed[3]. With this decoder, a user is assigned a score of the form

$$S_j = \sum_{x,y} \hat{f}_{X,Y|P}(x,y|p) \ln\left(\frac{\hat{f}_{X,Y|P}(x,y \mid p)}{\hat{f}_{X|P}(x \mid p)\hat{f}_{Y|P}(y \mid p)}\right), \tag{4.56}$$

and again the decision to accuse user j or not depends on whether his score S_j exceeds some fixed threshold η. Here $\hat{f}$ is the empirical estimate of the actual probability f, i.e., $\hat{f}_{X,Y|P}(x,y|p) = |\{i : (x_{j,i}, y_i) = (x,y)\}|/\ell$. Writing out the empirical probability outside the logarithm, and replacing the summation over x, y by a summation over the positions i, this is equivalent to

$$S_j = \frac{1}{\ell}\sum_{i=1}^{\ell} \ln\left(\frac{\hat{f}_{X,Y|P}(x_{j,i}, y_i \mid p)}{\hat{f}_{X,Y|P,H}(x_{j,i}, y_i \mid p, H_0)}\right). \tag{4.57}$$

[3]When p_i is not fixed and is drawn from a continuous distribution function f_P, the empirical probabilities considered in the text do not make much sense, as each value of p_i only occurs once. In that case one could e.g. build a histogram of values of p, and compute empirical probabilities for each bin, or discretize the distribution function for p using e.g. discrete arcsine distributions described in Chapter 2.

Now, this almost fits the score-based simple decoder framework, except that the empirical probabilities inside the logarithm are not independent for different positions i. To overcome this problem, we could try to replace the empirical probabilities $\hat{f}$ by the actual probabilities f, but to compute $f_{X,Y|P}(x_{j,i}, y_i|p)$ we need to know whether user j is guilty or not. Solving this final problem using Bayesian priors, we get the following result.

Lemma 4.9. *Approximating the empirical probabilities in the empirical mutual information decoder using Bayesian priors (with $\mathbb{P}(j \in \mathcal{C}) = \frac{c}{n}$), this decoder corresponds to using the following score function m:*

$$m(x, y, p) = \ln\left(1 + \frac{c}{n}\left[\frac{f_{X,Y|P,H}(x, y|p, H_1)}{f_{X,Y|P,H}(x, y|p, H_0)} - 1\right]\right). \tag{4.58}$$

Proof. The value of $f_{X,Y|P,H}(x_{j,i}, y_i|p, H_0)$ can be computed without any problems, so let us focus on the term $f_{X,Y|P}(x_{j,i}, y_i|p)$. Using Bayesian inference, we have:

$$f_{X,Y|P}(x, y \mid p) = f_H(H_1) f_{X,Y|P,H}(x, y \mid p, H_1) + f_H(H_0) f_{X,Y|P,H}(x, y \mid p, H_0). \tag{4.59}$$

Assuming an a priori probability of guilt of $f_H(H_1) = \mathbb{P}(j \in \mathcal{C}) = \frac{c}{n}$ and dividing by $f_{X,Y|P,H}(x, y|p, H_0)$ we get

$$\frac{f_{X,Y|P}(x, y \mid p)}{f_{X,Y|P,H}(x, y \mid p, H_0)} = \left(\frac{c}{n}\right) \cdot \frac{f_{X,Y|P,H}(x, y \mid p, H_1)}{f_{X,Y|P,H}(x, y \mid p, H_0)} + \left(1 - \frac{c}{n}\right) \cdot 1. \tag{4.60}$$

Taking logarithms, this leads to the expression of (4.58). □

Although this score function looks very similar to the Neyman–Pearson based log-likelihood decoder g, there are some essential differences. For instance, for the all-1 attack we have $g(1, 0, p) = -\infty$ while $m(1, 0, p) = \ln(1 - \frac{c}{n}) > -\infty$. For the interleaving attack, for which we may again hope to obtain a universal decoder using this approach, we do get a familiar result.

Corollary 4.10. *For the interleaving attack, the Bayesian approximation of the empirical mutual information decoder of* (4.58) *satisfies*

$$m(x, y, p) = \begin{cases} \ln\left(1 + \frac{p}{n(1-p)}\right) & \text{if } x = y = 0; \\ \ln\left(1 - \frac{1}{n}\right) & \text{if } x \neq y; \\ \ln\left(1 + \frac{1-p}{np}\right) & \text{if } x = y = 1. \end{cases} \tag{4.61}$$

For values of $p \in [\delta, 1 - \delta]$ with $\delta > 0$ and large c, this decoder is again equivalent to both the log-likelihood score function g and Oosterwijk–Škorić–Doumen's score function h [OŠD13]:

$$n \cdot m(x, y, p) \sim c \cdot g(x, y, p) \sim h(x, y, p). \tag{4.62}$$

For $p \approx 0, 1$ the logarithms in m guarantee that scores do not blow up, but due to the factor n in the denominator (rather than the factor c, as in g) the scores relatively increase more when p approaches 0 than for the score function g.

4.4.2 – The joint interleaving decoder. Let us finally consider the setting of defending against arbitrary attacks using a joint decoder. Through a series of reductions we can prove that the joint interleaving decoder achieves an asymptotic code length of $\ell \sim 2c^2 \ln n$. If the joint capacity for the uninformed setting is $1/(2c^2 \ln 2)$, this decoder is then capacity-achieving for the joint setting.

Theorem 4.11. *The joint log-likelihood decoder designed against the interleaving attack, corresponding to the score function* g *defined by*

$$g(z, y, p) = \begin{cases} \ln(1 - \frac{z}{c}) - \ln(1-p) & \text{if } y = 0; \\ \ln(\frac{z}{c}) - \ln(p) & \text{if } y = 1. \end{cases} \tag{4.63}$$

and the arcsine distribution encoder f_p^* *of* (4.42) *together achieve an asymptotic code length of* $\ell \sim 2c^2 \ln n$.

Proof. We will prove that asymptotically, the proposed universal joint decoder is equivalent to the universal simple decoder described above, and therefore achieves similar code lengths for large c and n.

Suppose we have a tuple $\mathcal{T}$ of size c, and suppose in some segment i there are z users who received a 1 and $c-z$ users who received a 0. For now also assume that $p \in [\delta, 1-\delta]$ for some $\delta > 0$ that does not depend on c. In case $y = 0$, the combined simple decoder score of this tuple $\mathcal{T}$ (using the simple universal decoder g described previously) would be:

$$\sum_{j \in \mathcal{T}} S_{j,i} = z \ln\left(1 - \frac{1}{c}\right) + (c-z) \ln\left(1 + \frac{p}{c(1-p)}\right) \sim -\frac{z}{c} + \frac{(c-z)p}{c(1-p)} = \frac{p - z/c}{1-p}\,. \tag{4.64}$$

On the other hand, if we look at this tuple's joint score with the joint universal decoder g of (4.63), we have

$$S_{\mathcal{T},i} = g(z, 0, p) = \ln\left(\frac{1 - z/c}{1-p}\right) = \ln\left(1 + \frac{p - z/c}{1-p}\right) \sim \frac{p - z/c}{1-p}\,. \tag{4.65}$$

Note that the last step follows from the fact that for large c, with overwhelming probability we have $z = cp + O(\sqrt{cp})$ (since Z is binomially distributed with mean cp and variance $cp(1-p)$), in which case $(p - z/c)/(1-p) = o(1)$. Combining the above, we have that $S_{\mathcal{T},i} \sim \sum_{j \in \mathcal{T}} S_{j,i}$. So the joint universal decoder score for a tuple $\mathcal{T}$ is asymptotically equivalent to the sum of the simple universal decoder scores for the members in this tuple, if $p \in [\delta, 1-\delta]$.

Since as argued before the distribution tails $[0, \delta]$ and $[1-\delta, 1]$ are negligible for the performance of the scheme for sufficiently small δ, and since the same result holds for $y = 1$, the simple and joint decoders are asymptotically equivalent and therefore achieve equivalent code lengths. □

CHAPTER 5

Sequential decoding schemes

5.1 — Overview

Context. In some scenarios, the pirate output is generated and distributed in real-time, and might also be detected by the tracer in real-time. Think of live streams of football matches, broadcast live on a pirate website. In this scenario, the tracer can adjust future parts of the watermarks based on the pirate broadcast he has observed so far, and may already be able to disconnect users before the broadcast is over. In this scenario the tracer is clearly more powerful than in the non-adaptive setting; he can simply choose to use a non-adaptive construction as e.g. described in Chapter 4, or he could try to make use of the extra knowledge to do even better.

While some results in the past already showed [BPS01, FT99, FT01, LOD12, Roe11, SNW03, Tas05] that it was possible to design more efficient adaptive schemes (performing significantly better than their non-adaptive counterparts), it was only in 2013 when Laarhoven–Doumen–Roelse–Škorić–De Weger [LDR$^+$13] showed that significant gains can also be obtained in the bias-based framework. In particular, in [LDR$^+$13] it was shown that in the adaptive setting, where code words are sent out symbol by symbol and the distributor is allowed to base future decisions on previous pirate output, *all* colluders can provably be found with a code length $\ell \propto c^2 \log n$, using a dynamic version of Tardos' scheme. This is in contrast with non-adaptive schemes, where it can never be guaranteed that more than one pirate is caught (due to the so-called scapegoat attack, where one pirate sacrifices himself and always outputs his content, leaving the other colluders unharmed).

Although various of the insights obtained for the non-adaptive setting directly carry over to the adaptive setting, such as the possible use of the log-likelihood scores of Chapter 4 in the 'dynamic Tardos scheme', several questions remain with respect to the adaptive setting. For instance:

- Is the "dynamic Tardos scheme" proposed in [LDR$^+$13, Laa13a] optimal?
- What motivates the design of this construction?

Answering these and related questions may ultimately lead to the same level of understanding for the adaptive case as for the non-adaptive setting, which may allow practitioners to make well-motivated design choices in the adaptive setting as well.

Results. In this chapter we answer the second question by showing a connection between the ideas behind the dynamic Tardos scheme and what is known in the literature as

*This chapter is based on results from [LDR$^+$13, Laa13a, Laa15c].

the *sequential probability ratio test (SPRT)*, invented by Wald in the 1940s [Wal47]. As a result, we are also able to take a first step towards answering the first question: within the class of *sequential* fingerprinting schemes, where the code book is not constructed adaptively and only accusations are done adaptively, both the dynamic Tardos scheme and schemes built from Wald's SPRT are asymptotically optimal for the uninformed fingerprinting game. We discuss in detail how sequential fingerprinting schemes can naturally be constructed from Wald's SPRT, and how various results from the corresponding SPRT literature can be used to tune these schemes to different fingerprinting scenarios. We further compare the dynamic Tardos scheme to Wald's SPRT, and highlight why in general Wald's scheme should be preferred.

Outline. First, in Section 5.2 we review the dynamic Tardos scheme and its variants. Then, in Section 5.3 we describe Wald's sequential probability ratio test procedure, and how it can be applied to fingerprinting to obtain optimal sequential fingerprinting schemes. Finally, Section 5.4 illustrates the similarities and differences between these schemes, and concludes with an overview of both schemes.

5.2 — The sequential Tardos scheme

5.2.1 – Non-adaptive scheme. Recall that in Tardos' original scheme [Tar08] and many of its subsequent variants, decoding in the non-adaptive setting is done as follows. First, for each segment i and user j, scores $S_{j,i} = g(x_{j,i}, y_i, p_i)$ are computed using a score function g. Then, a user $j \in \mathcal{U}$ is accused iff $S_j = \sum_{i=1}^{\ell} S_{j,i} > \eta$ for some well-chosen threshold η. As described in Chapter 4, the following Neyman–Pearson-motivated log-likelihood score function g is a good choice to deal with arbitrary attacks:

$$g(x, y, p) = \begin{cases} \ln\left(1 + \frac{p}{c(1-p)}\right) & \text{if } x = y = 0; \\ \ln\left(1 - \frac{1}{c}\right) & \text{if } x \neq y; \\ \ln\left(1 + \frac{1-p}{cp}\right) & \text{if } x = y = 1. \end{cases} \tag{5.1}$$

To illustrate the situation of cumulative user scores and the accusation procedure in the non-adaptive setting, Figure 5.1a sketches the scores $S_j(i_0) = \sum_{i=1}^{i_0} S_{j,i}$ against i_0, for $i_0 = 0$ up to the final moment of decision $i_0 = \ell$. Assuming a colluder-symmetric collusion channel, scores of users $j \notin \mathcal{C}$ follow a certain random walk with a negative drift $\mu_0 < 0$ and a relatively large variance σ_0^2, while scores of guilty users $j \in \mathcal{C}$ follow a random walk with a positive drift $\mu_1 > 0$ and a smaller variance σ_1^2.

5.2.2 – Sequential scheme. The improvement described in [LDR+13] for the adaptive setting does not change the code generation phase at all, so although it was coined the *dynamic* Tardos scheme, it may more suitably be called the *sequential* Tardos scheme. The modification compared to the non-adaptive scheme described above, to make better use of the sequential setting, is the following: instead of only cutting off users from the content at the very end, when their scores exceed η, disconnect users as soon as their (normalized) scores exceed the (normalized) threshold η. This prevents the colluder from contributing to the remaining parts of the content, and allows the distributor to find the remaining colluders as well. Here the normalization refers to translating the scores by $+i_0\mu_0$, so that innocent users are always expected to have an average score of $\mathbb{E}(S_j(i_0)|H_0) = 0$.

To illustrate the effect of this change to the scheme, Figure 5.1b sketches the cumulative user scores in the sequential setting *without normalization*, and the new accusation criterion. Without normalization, the scores follow the same general path as in Figure 5.1a, and the red accusation threshold becomes a decreasing line, rather than a horizontal line as in [LDR$^+$13,Laa13a]. As discussed in [LDR$^+$13], with this modification one can provably find *all* colluders with a similar provable code length as required in the non-adaptive setting to catch at least one colluder. The central result of [LDR$^+$13] can be stated as follows.

Theorem 5.1. *[LDR$^+$13] Suppose ℓ and η are chosen in the non-adaptive Tardos scheme to guarantee that*

(i) with probability at least $1-\varepsilon_0$ no innocent users are accused;

(ii) with probability at least $1-\varepsilon_1$ at least one colluder is accused.

Then, using almost the same scheme parameters as before[1]*, with this sequential construction we can guarantee that*

(i) with probability at least $1-2\varepsilon_0$ no innocent users are ever accused;

(ii) with probability at least $1-2\varepsilon_1$ all colluders are accused.

In practice, this means that to turn a non-adaptive scheme into a sequential scheme that provably finds all colluders, we just have to replace ε_0 and ε_1 by $\frac{1}{2}\varepsilon_0$ and $\frac{1}{2}\varepsilon_1$ in the formulas for ℓ and η of the non-adaptive setting. Since ℓ only depends logarithmically on ε_0 and ε_1, for large n and c the resulting increase in the code length is negligible.

5.2.3 – Sequential variants. While the sequential scheme described above deals well with the setting where c is known and users can be accused after every position i, the paper [LDR$^+$13] also discussed slight variations of this setting, which may occur in practice. In particular, the two problems of not being able to cut off users after every segment i, and not knowing c, were addressed in [LDR$^+$13, Sections IV and V].

Weakly sequential decoding. To make tracing harder, pirates may delay the pirate output, so that a user whose score exceeds η at time i_0 can only be disconnected at time, say, i_0+B. As we are now quite certain that he is guilty, and since he contributed to segments $i_0+1,\dots,i_0+B$, we could consider these segments *tainted* and disregard them completely for tracing the remaining colluders. This solution was proposed in [LDR$^+$13, Section IV.A] and it was shown to lead to a moderate increase in the code length of $(c-1)B$.

Universal sequential decoding. For the setting where c is unknown and only a crude upper bound $c_0 \geqslant c$ is known, [LDR$^+$13, Section V] proposed a method where each user is assigned several scores $S_j^{(1)},\dots,S_j^{(c_0)}$ based on how large the coalition is estimated to be, and disconnecting a user as soon as one of his scores crosses one of the corresponding boundaries $\eta^{(1)},\dots,\eta^{(c_0)}$. It was noted in [LDR$^+$13] that the scores are very similar and the boundaries seem to correspond to a continuous function $\eta(i_0) \propto \sqrt{i_0}$. One of the open problems posed in [LDR$^+$13, Section VII.B] was therefore whether schemes with single scores and curved boundaries are provably secure.

[1] This disregards a small technical detail regarding the overshoot over the boundary η; see the discussion of Z and $\tilde{Z}$ in [LDR$^+$13, Section III.C]. To be sure that the scheme still works we can disregard scores right after a user is removed from the system [Laa13a, Section II] with a negligible increase in ℓ. We omit details here, and only present the simplified result.

Joint decoding. Finally, another topic of interest in fingerprinting is joint decoding, where the entire code $\mathcal{X}$ (instead of only the user's code word $\mathbf{x}_j$) is used to decide whether user j should be accused. Assigning scores to tuples of users was considered before in e.g. [ODL14], but no explicit decision criterion with provable bounds on the code length and error probabilities were provided, and finding such a joint decoding scheme was left as an open problem.

5.3 — The sequential Wald scheme

5.3.1 – Sequential scheme. To understand the motivation behind the sequential Tardos scheme, and to see how the design can possibly be improved, we now turn our attention to what has long been known in statistics literature to be a solution for hypothesis testing in sequential settings: Wald's sequential probability ratio test (SPRT). This scheme originated in the 1940s [Wal45, Wal47], and countless follow-up works have appeared since, which have been summarized in various books on this topic [BLS13, Che72, Gov04, JT00, MS09, Sie85, Wal47, WG86].

Let us recall the formulation of the fingerprinting problem in terms of hypothesis testing, where we want to distinguish between the following two hypotheses:

$$H_0: \quad \text{user } j \text{ is innocent } (j \notin \mathcal{C}), \tag{5.2}$$

$$H_1: \quad \text{user } j \text{ is guilty } (j \in \mathcal{C}). \tag{5.3}$$

Now, to decide between these two hypotheses in sequential settings, Wald proposed the following procedure. Let η_1 and η_0 be two constants, with $\eta_1 > 0 > \eta_0$, and again let us use the optimal log-likelihood score function g from (5.1). Now we decide in favor of H_1 as soon as a user's cumulative score exceeds η_1, and we decide to accept H_0 as soon as the user's score drops below η_0. As long as a user score stays in the interval $[\eta_0, \eta_1]$, we continue testing. This accusation procedure is sketched in Figure 5.1c.

Choosing the thresholds. To understand how the parameters η_0 and η_1 should be chosen, a connection is commonly made with the continuous-time analog of random walks, Brownian motions. Assuming that user scores are continuous rather than discrete, so that when a score crosses one of the boundaries it really *hits* the boundary (rather than jumping over it, in the discrete model), then to guarantee that an innocent user is acquitted with probability at least $1-\varepsilon_0'$ and a guilty user is accused with probability at least $1-\varepsilon_1'$, the following choice is optimal:

$$\eta_0 = \ln\left(\frac{\varepsilon_1'}{1-\varepsilon_0'}\right), \qquad \eta_1 = \ln\left(\frac{1-\varepsilon_1'}{\varepsilon_0'}\right). \tag{5.4}$$

To guarantee that all innocent users are acquitted and all guilty users are found, we need to let $\varepsilon_0' = O(\frac{1}{n})$ and $\varepsilon_1' = O(\frac{1}{c})$, which for large c, n means $\eta_0 \sim -\ln c$ and $\eta_1 \sim \ln n$. For instance, writing $\varepsilon_0' = \varepsilon_0/n$ and $\varepsilon_1' = \varepsilon_1/c$, so that the probability of not accusing innocents (accusing all guilty users) is at least $1-\varepsilon_0$ ($1-\varepsilon_1$), this corresponds to taking

$$\eta_0 = \ln\left(\frac{\varepsilon_1/c}{1-\varepsilon_0/n}\right), \qquad \eta_1 = \ln\left(\frac{1-\varepsilon_1/c}{\varepsilon_0/n}\right). \tag{5.5}$$

There are two important issues that we need to address, the first of which is that we are not dealing with continuous user scores but discrete scores. One of the effects of

having discrete jumps in the scores is that there may be a slight *overshoot* over one of the boundaries when a user is accused or acquitted; a score may cross one of the lines at a non-integral point so to say, and at the next measurement the score may significantly exceed η_1 or drop below η_0. As a result the error probabilities for the above choice of thresholds are not exact. A useful property of the above choice of parameters is that if by $\tilde{\varepsilon}_0'$ and $\tilde{\varepsilon}_1'$ we denote the *real* probabilities of accusing innocent and guilty users, when using these thresholds η_0 and η_1, we have [Wal45, Equation (3.30)]

$$\tilde{\varepsilon}_0' + \tilde{\varepsilon}_1' \leqslant \varepsilon_0' + \varepsilon_1'. \tag{5.6}$$

In other words, the total error probability does not increase, and at most one of ε_0' and ε_1' might increase. Alternatively, exact bounds on the error probabilities can be obtained, showing that the following slightly conservative choice of parameters guarantees that the error bounds are satisfied:

$$\eta_0 = -\ln(1/\varepsilon_1'), \qquad \eta_1 = \ln(1/\varepsilon_0'). \tag{5.7}$$

The second issue that we should address is that having a threshold η_0 only makes sense if all colluders have an increasing score. If the colluders know about the tracing algorithm, and use an asymmetric pirate strategy, e.g. by letting one colluder be inactive at the start and letting him join in later, this colluder will incorrectly be acquitted early on. In this setting one could say that innocence is impossible to "prove", as a colluder could remain inactive and hidden for long periods of time, and it is only possible to prove with high probability that someone is in fact guilty. To deal with this problem, a simple solution is not to use a lower threshold η_0 at all. This is equivalent to setting $\varepsilon_1' = 0$, as that way we will never say a colluder is innocent and he will always be caught. In that case, the conservative choice of thresholds from (5.4) can be stated as

$$\eta_0 = -\infty, \qquad \eta_1 = \ln(1/\varepsilon_0') = \ln(n/\varepsilon_0). \tag{5.8}$$

Note that in this case, the aggressive and conservative expressions from (5.4) and (5.7) match, i.e., ε_0' is a tight bound on the probability of incorrectly accusing a single innocent user. This more realistic implementation of the sequential probability ratio test in the uninformed fingerprinting game is sketched in Figure 5.1d.

Optimality of the SPRT. Although reaching a decision with this procedure may theoretically take a very long time, Wald proved [Wal47, Appendix A] that his test procedure always terminates, regardless of ε_0 and ε_1. Furthermore, if by μ_0 (μ_1) and σ_0^2 (σ_1^2) we denote the expected score in one segment for innocent (guilty) users, then we know that with high probability, the procedure will terminate not long after $i_0 \cdot \mu_0 + O(\sigma_0)$ ($i_0 \cdot \mu_1 + O(\sigma_1)$) crosses the boundary η_0 (η_1).

More formally, Wald analyzed the expected time by which his procedure terminates, under either H_0 or H_1, and together with Wolfowitz he proved [WW48] that his SPRT is optimal in that it minimizes the expected time before a decision is reached, both under H_0 and under H_1. Ignoring overshoots over the boundary (i.e., assuming we are dealing with continuous random walks), he further derived explicit expressions for both these expected termination times, which are stated below. In the following theorem, as in Chapter 3 we write $d(a\|b) = a\ln(\frac{a}{b}) + (1-a)\ln(\frac{1-a}{1-b})$ for the Kullback–Leibler divergence (in nats) between a and b.

Theorem 5.2. *[Wal47, WW48] Suppose we have a sequential test procedure, for which*

- *an innocent user is accused with probability at most ε_0';*
- *a guilty user is acquitted with probability at most ε_1';*
- *the probability of termination is* 1.

Let T denote the time at which a decision is reached. Then:

$$\mathbb{E}(\mathsf{T}|H_0) \geqslant \frac{1}{-\mu_0} d(\varepsilon_0' \| 1-\varepsilon_1') \approx \frac{\ln(1/\varepsilon_1')}{-\mu_0}, \tag{5.9}$$

$$\mathbb{E}(\mathsf{T}|H_1) \geqslant \frac{1}{\mu_1} d(\varepsilon_1' \| 1-\varepsilon_0') \approx \frac{\ln(1/\varepsilon_0')}{\mu_1}. \tag{5.10}$$

Furthermore, the sequential probability ratio test is a sequential test simultaneously minimizing both $\mathbb{E}(\mathsf{T}|H_0)$ and $\mathbb{E}(\mathsf{T}|H_1)$, and assuming that there is no overshoot over the boundaries, both inequalities above are equalities for the SPRT.

For large n, the per-user false positive error probability scales as $\varepsilon_0' = \Theta(1/n)$ while $\varepsilon_1' = \Theta(1)$ based on the argument that if the average pirate score exceeds η_1, all pirate scores exceed η_1 [LDR+13]. We further have that

$$\mu_1 = \mathbb{E}_P \sum_{x,y} f_{X,Y|P}(x,y|p,H_1) \ln \frac{f_{X,Y|P}(x,y|p,H_1)}{f_{X,Y|P}(x,y|p,H_0)} = (\ln 2) I(X_1;Y|P), \tag{5.11}$$

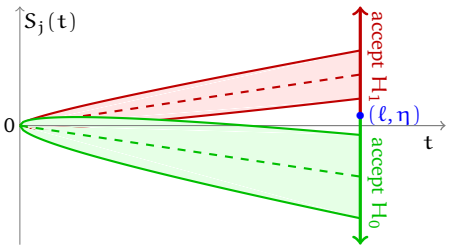
where $I(X_1, Y|P = p)$ is the mutual information between a pirate symbol and the pirate output, as described in Chapter 3. This leads to the following corollary.

Theorem 5.3. *For sequential tests satisfying the conditions stated in Theorem 5.2, we have:*

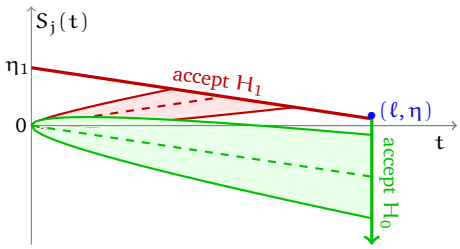
$$\mathbb{E}(\mathsf{T}|H_1) \gtrsim \frac{\log_2 n}{I(X_1;Y|P)}. \tag{5.12}$$

This result implies that in general, sequentiality does not lead to a decrease in the asymptotic code length; with non-adaptive schemes it is also possible to achieve this asymptotic code length, as discussed in Chapter 4. The two gains of sequential testing are that (i) in fact *all* colluders, rather than at least one of them, can provably be caught with this asymptotic code length; and (ii) in practice, for finite c and n, the time needed to find and trace all colluders will generally be shorter than in the non-adaptive setting. Although the asymptotic code lengths are the same, the convergence to this limit is significantly faster for sequential schemes than for non-adaptive schemes.

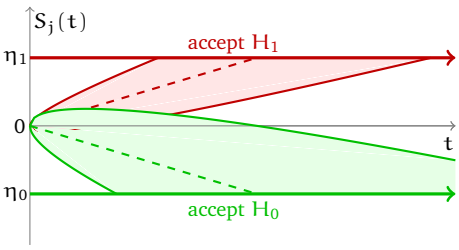
While most of the analyses and results above are based on running this scheme with parallel infinite boundaries, it is not impossible to force an early decision. As already described by Wald [Wal47, Section 3.8], one might ultimately prefer to *truncate* the test procedure at some fixed time ℓ, at which we make a decision similar to the sequential Tardos scheme, and similar to the non-adaptive setting. This may be done with and without a lower boundary; a sketch for the case with a lower boundary is given in Figure 5.1e. Analyzing these variants rigorously seems difficult, even with Brownian approximations, but an interested reader may refer to e.g. one of the books on sequential testing listed at the beginning of this section. With truncation, one should ask the question whether forcing a decision by some fixed time ℓ is really important. After all, if the main goal is to minimize the *worst-case* code length ℓ needed to make a decision, then it is commonly best to wait until the very end and to take all evidence into account before making any decisions at all; which exactly corresponds to the non-adaptive setting.



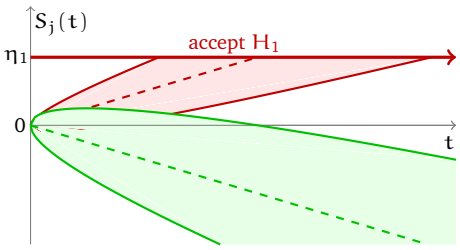
(a) Non-adaptive decoding (cf. Chapter 4)



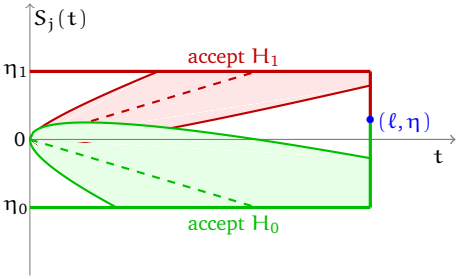
(b) The sequential Tardos scheme



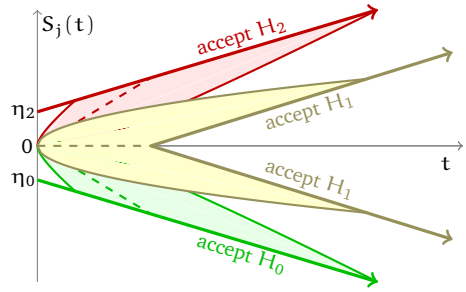
(c) Wald's sequential probability ratio test (SPRT)



(d) Wald's SPRT with no early innocent decisions



(e) Wald's SPRT with truncated thresholds



(f) Sobel and Wald's multihypothesis SPRT

Figure 5.1: Various flavors of non-adaptive (5.1a) and sequential decoding schemes (5.1b–5.1f). The green and red marked areas (dashed lines) indicate the range (average) of innocent and guilty user scores respectively. Accusing a user or not is commonly based on whether a user score $S_j(t)$ exceeds a certain threshold η or not. Figure 5.1f further illustrates how joint decoding might work, where scores are assigned to pairs of users.

5.3.2 – Sequential variants. The SPRT has received extensive attention in the literature, with thorough analyses of the effects of the overshoot over the boundaries, slight modifications of the scheme (such as the truncated SPRT mentioned above), and the effects of using different boundaries than the horizontal lines in the figures above. We highlight two variants which we also considered for the sequential Tardos scheme, and we consider how joint decoding may be done with the SPRT. For further details we refer the interested reader to e.g. [BLS13, Che72, Gov04, JT00, MS09, Sie85, Wal47, WG86].

Weakly sequential decoding. In the setting of weakly adaptive decoding, where pirates delay their rebroadcast of the content (or where content is sent out in blocks of size B), the results based on continuous approximations using Brownian motions become less

and less accurate. For large B, the overshoot over the boundary becomes more significant, which was also discussed in [LDR+13, Section IV.B].

To deal with this problem, we can use the method described in [LDR+13, Section IV.A]: ignore the tainted segments i to which a user who is now deemed guilty may have contributed. Then the increase in the code length is at most $(c-1)B$, which in the uninformed fingerprinting game is negligible with respect to $\ell \propto c^2 \log n$.

Universal decoding. Recall that in the universal decoding setting, we assume that c is unknown, and only a crude bound $c_0 \gg c$ may be known. To deal with this, the paper [LDR+13] proposed to keep multiple scores per user, and multiple accusation boundaries. It was conjectured that using a single score for each user, and using a curved boundary function of the form $\eta_1(i_0) \propto \sqrt{i_0}$ may be possible.

In terms of hypothesis testing, testing whether $j \in \mathcal{C}$ or $j \notin \mathcal{C}$ for unknown coalition sizes c could be considered a test of a simple null hypothesis $H_0 : \mu = \mu_0$ against a one-sided alternative $H_1 : \mu > \mu_0$. In the informed setting, where the collusion channel is known, we might know exactly what μ_0 is, and so such a one-sided test may form a solution. In that case, curved stopping boundaries (in particular, having a boundary of the shape $\eta_1(i_0) \propto \sqrt{i_0}$) has been suggested before; see e.g. [Sie85, Chapter IV]. When using the symmetric Tardos score function rather than the optimized log-likelihood ratios or MAP decoders, this approach may work well, although the issue remains that it seems that no single encoder and decoder can be used for arbitrary c and θ: in all known cases, either the decoder depends on c or c_0, or the encoder uses a cutoff which depends on c.

To work with different score functions than Tardos' score function [Tar08] and Škorić–Katzenbeisser–Celik's symmetric score function [ŠKC08], where μ_0 may be considered fixed, we need to circumvent the issue that μ_0 may depend on c and θ as well. In the universal uninformed decoding setting we therefore do not even know what μ_0 is. Two hypotheses that may be more realistic to consider are $H_0 : \mu \leqslant 0$ against $H_1 : \mu > 0$: an innocent user will have a negative average score, while a guilty user will have a positive average score. However, depending on the collusion strategy, the values of μ_0 and μ_1 may both be small or large. This does not really help the colluders, as decreasing $|\mu_0|$ and $|\mu_1|$ also leads to a decrease in the variance of the scores, but it makes using a single linear decoder problematic.

To deal with these problems, the best solution for the universal setting may be to use a generalized linear decoder [AZ10, DHPG13, MF12], and to normalize the scores during the decoding phase, as described in [Laa13a]. A generalized linear decoder is better suited for the setting of unknown c, and by normalizing user scores (which can be done based on $\mathcal{X}, \mathbf{p}, \mathbf{y}$) we know what μ_0 is. Then we can again use a hypothesis test of the form $H_0 : \mu = \mu_0$ against $H_1 : \mu > \mu_0$, where a curved boundary may be optimal [JT00].

Joint decoding. Recall that in joint decoding the entire code $\mathcal{X}$ is taken into account to decide whether users should be accused. As in [ODL14] one might assign scores to

tuples $\mathcal{T}$ of c users, and try to distinguish between the following $c+1$ hypotheses:

$$H_0: \quad \text{tuple } \mathcal{T} \text{ contains no guilty users } (|\mathcal{T} \cap \mathcal{C}| = 0), \tag{5.13}$$
$$H_1: \quad \text{tuple } \mathcal{T} \text{ contains one guilty user } (|\mathcal{T} \cap \mathcal{C}| = 1), \tag{5.14}$$
$$\vdots$$
$$H_c: \quad \text{tuple } \mathcal{T} \text{ contains only guilty users } (|\mathcal{T} \cap \mathcal{C}| = c). \tag{5.15}$$

Although not quite as well studied as the case of two hypotheses, this topic has also received attention in SPRT literature, with the earliest work dating back to Sobel and Wald from 1949 [SW49]. There the problem of deciding between three simple hypotheses was considered, and a solution was given as sketched in Figure 5.1f. Using joint Neyman–Pearson decoders to assign scores to pairs of users, several stopping boundaries may be used, each corresponding to a decision of accepting one of the three hypotheses. The distribution of scores then depends on whether the tuple contains $0, 1$ or 2 colluders, as illustrated by the green, yellow, and red marked areas in Figure 5.1f. This procedure can be generalized to multiple hypotheses as well, to deal with joint decoding with $c > 3$ colluders. For details on how to choose these stopping boundaries $\eta_t(i_0)$, see e.g. [SW49].

5.4 — Tardos vs. Wald: A comparison

In the previous two sections we saw how to construct sequential schemes based on the (sequential) Tardos scheme, and based on Wald's SPRT. Here we briefly consider possible applications of both schemes, and how the two schemes compare in these settings. We consider four scenarios as follows:

1. Defending against small collusions.
2. Defending against the interleaving attack.
3. Defending against the all-1 attack.
4. Defending against arbitrary pirate attacks.

These settings are studied in the following subsections.

5.4.1 – Defending against small collusions. One of the challenges in designing an effective traitor tracing scheme is to choose the parameter c_0, the number of colluders to defend against. Ultimately c_0 should be a good estimate for the real collusion size c, but it should also not be smaller than any collusion size c that may appear in practice; otherwise certain collusions may be able to get away with piracy. So far none of the known (linear) decoders work perfectly well against arbitrary collusion sizes, and so in practice one commonly just has to choose a somewhat large parameter c_0 which is certainly higher than any practical collusion size.

To illustrate how both approaches, using the sequential Tardos scheme and Wald's SPRT, deal slightly differently with this inability to choose c_0 exactly equal to c, we will consider an extreme example: a scheme is designed against collusions of size c_0, and in practice the collusions consist only of single users; pirates operate almost independently, and for simplicity we assume they also operate sequentially. This could be viewed as an instance of the so-called scapegoat attack, where the pirate output is always the same colluder's output; until be is accused and he cannot contribute anymore, in which case another colluder takes over.

Tardos' scheme. In the sequential Tardos scheme with log-likelihood scores, setting the parameters is quite difficult. Various provable bounds on the error probabilities for given parameters are not tight, leading to pessimistic estimates and higher thresholds and code lengths than required. One could also estimate the actual required code length for a given set of parameters directly, leading to better scheme parameters, but this would have to be done for each instance separately; if any of the parameters $c, n, \varepsilon_0, \varepsilon_1$ then changes, one would have to redo the simulations or computations to find good practical parameters for the new setting.

To illustrate how far the provable parameters are off from reality, let us use the toy example of $c_0 = c = 10$, $n = 1000$ and $\varepsilon_0 = \varepsilon_1 = 10^{-3}$, and let us use the provable bounds from Theorems 4.1 and 5.1. This leads to the following parameters:

$$\mu_0 \approx 0.00382, \qquad \mu_1 \approx -0.00343, \tag{5.16}$$

$$\ell \approx 17\,953, \qquad \eta_1^{(\ell)} \approx 6.9078, \qquad \eta_1^{(0)} \approx 68.41. \tag{5.17}$$

The scheme is guaranteed to terminate within time $\ell \approx 18\,000$, but as previously described in [LDR+13] the scheme commonly terminates much sooner. We expect innocent and guilty user scores to behave roughly as sketched in Figure 5.1b, and Figure 5.2a shows that this is indeed the case.

Wald's scheme. Using Wald's solution, note that if we were to use a lower threshold, colluders that only become active later on may well be acquitted early on. Instead it should be preferred to not use a lower boundary (setting $\varepsilon_1 = 0$ and $\eta_0 = -\infty$), as previously illustrated in Figure 5.1d. A typical outcome of a simulation is illustrated in Figure 5.2b, which shows all colluders are commonly found after $t \approx 2500$ symbols.

5.4.2 – Defending against the interleaving attack. When defending against the interleaving attack, which may be the most practical pirate strategy due to its simplicity and its strength, the Neyman–Pearson decoder of (5.1) designed against the interleaving attack [Laa14] may again be a good choice, even when the tracer's estimate c_0 is not exact [FD14]. With this score function, in each segment the average pirate score increases by $\mu_1 \sim \frac{1}{2c^2}$, while for innocent users we have $\mu_0 \sim \frac{-1}{2c^2}$ in case the arcsine distribution is used, as the following lemma shows.

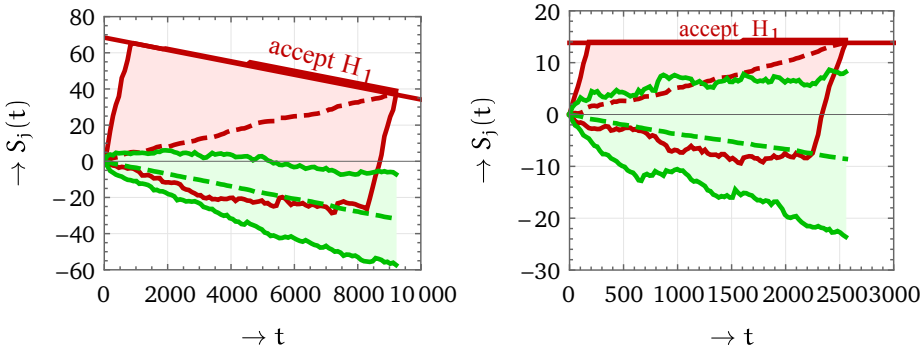
Lemma 5.4. *Suppose that:*

- *the encoder uses the arcsine distribution encoder* Γ_p^**;*
- *the collusion channel is the interleaving attack* θ_{int}*;*
- *the decoder is the interleaving log-likelihood decoder* g*.*

Then for large n *and* c*, the expected score of innocent users (*μ_0*) and guilty users (*μ_1*) in a single segment satisfy:*

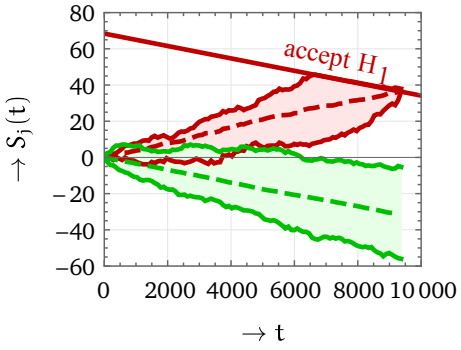
$$\mu_0 = \mathbb{E}_{x,y,p}\big[g(x,y,p) \mid H_0\big] \sim -\frac{1}{2c^2}, \tag{5.18}$$

$$\mu_1 = \mathbb{E}_{x,y,p}\big[g(x,y,p) \mid H_1\big] \sim +\frac{1}{2c^2}. \tag{5.19}$$

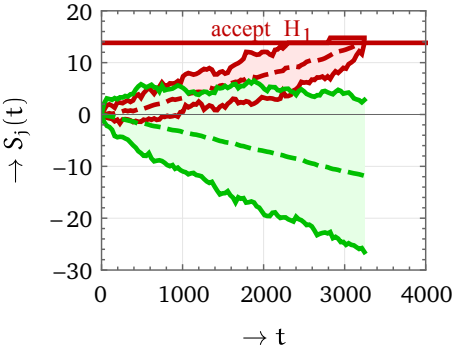


(a) Tardos' scheme: Scapegoat attack

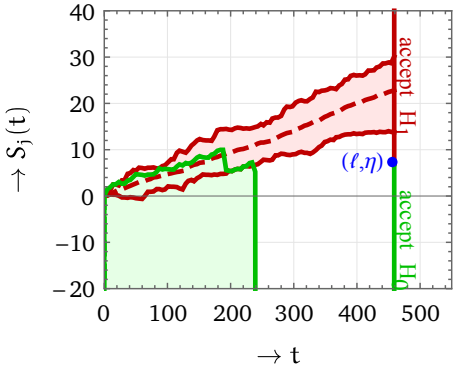
(b) Wald's scheme: Scapegoat attack



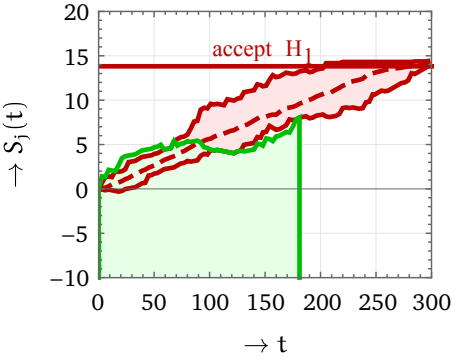
(c) Tardos' scheme: Interleaving attack



(d) Wald's scheme: Interleaving attack



(e) Tardos' scheme: All-1 attack



(f) Wald's scheme: All-1 attack

Figure 5.2: Simulations of both the sequential Tardos scheme and Wald's SPRT using parameters $c = 10$, $n = 1000$ and $\varepsilon_0 = \varepsilon_1 = 10^{-3}$. The dashed lines show the average scores and the boundaries of the marked areas are the highest and lowest scores of each group. Note that for the all-1 attack, the sequential Tardos scheme offers no improvement over non-adaptive decoding, while Wald's scheme does find the colluders faster.

Proof. For μ_0, we write out the expectation:

$$\mu_0 = \mathbb{E}_{x,y,p}\left[g(x,y,p) \mid H_0\right] \tag{5.20}$$

$$= \int_0^1 \frac{dp}{\pi\sqrt{p(1-p)}} \left[p^2 \ln\left(1+\frac{1-p}{cp}\right)\right. \tag{5.21}$$

$$\left. + \, 2p(1-p)\ln\left(1-\frac{1}{c}\right) + (1-p)^2 \ln\left(1+\frac{p}{c(1-p)}\right)\right]. \tag{5.22}$$

Similarly, we can write out the average colluder score in a single segment to get:

$$\mu_1 = \int_0^1 \frac{dp}{\pi\sqrt{p(1-p)}} \left[p^2\left(1+\frac{1-p}{cp}\right)\ln\left(1+\frac{1-p}{cp}\right)\right. \tag{5.23}$$

$$+ 2p(1-p)\left(1-\frac{1}{c}\right)\ln\left(1-\frac{1}{c}\right) \tag{5.24}$$

$$\left. + \, (1-p)^2\left(1+\frac{p}{c(1-p)}\right)\ln\left(1+\frac{p}{c(1-p)}\right)\right]. \tag{5.25}$$

Combining these results, and merging the logarithms into one term, we obtain the following expression for $\mu_1 - \mu_0$:

$$\mu_1 - \mu_0 = \int_0^1 \frac{\sqrt{p(1-p)}\,dp}{\pi c} \ln\left(1+\frac{c}{(c-1)^2 p(1-p)}\right). \tag{5.26}$$

Since we know that $\mu_1 \sim \frac{1}{2c^2}$ by (5.11), we need to prove that the right hand side is asymptotically similar to $\frac{1}{c^2}$. Rearranging terms, we need to prove that

$$I = \int_0^1 dp\sqrt{p(1-p)}\ln\left(1+\frac{c}{(c-1)^2 p(1-p)}\right) \sim \frac{\pi}{c}. \tag{5.27}$$

First, using $\ln(1+x) < x$ for all $x > 0$, we obtain:

$$I < \int_0^1 \frac{c\,dp}{(c-1)^2\sqrt{p(1-p)}} = \frac{\pi c}{(c-1)^2} \sim \frac{\pi}{c}. \tag{5.28}$$

To get a matching lower bound, we first reduce the range of integration from $[0,1]$ to $[\delta, 1-\delta]$ for some $\delta > 0$, noting that the integrand is strictly positive:

$$I > \int_\delta^{1-\delta} dp\sqrt{p(1-p)}\ln\left(1+\frac{c}{(c-1)^2 p(1-p)}\right). \tag{5.29}$$

Choosing $\delta = \frac{1}{\sqrt{c}}$, the term inside the logarithm is small and the following bound is tight enough to obtain the result:

$$I > \int_\delta^{1-\delta} \frac{c\,dp}{(c-1)^2\sqrt{p(1-p)}} - o\left(\frac{1}{c}\right) \sim \frac{\pi}{c} - \frac{4}{\pi c}\arcsin\sqrt{\delta} - o\left(\frac{1}{c}\right) \to \frac{\pi}{c}. \tag{5.30}$$

This proves that $I \sim \frac{\pi}{c}$, hence $\mu_0 \sim -\frac{1}{2c^2}$. □

Tardos' scheme. An illustration of how this scheme might work in practice is given in Figure 5.2c, where the same toy parameters as in the previous subsection are used. In most cases the scheme finds all pirates after roughly 9000 segments.

Wald's scheme. In Wald's scheme, without using a lower threshold, setting $\varepsilon_0' = \varepsilon_0/n$, i.e., $\eta_1 = \ln(n/\varepsilon_0)$ (and $\eta_0 = -\infty$) guarantees that with probability at least $1 - \varepsilon_0$, no innocent users are ever accused, and with probability 1 all colluders are eventually found. Figure 5.2d illustrates how the scheme might work, and again it takes around 3000 segments to trace the colluders.

Asymptotics. For large n and c we can derive what the parameters of both schemes converge to. First, for Wald's scheme, we see that the upper threshold converges to $\eta_1 \sim \ln n$ and the expected time until termination is $\ell \sim 2c^2 \ln n$. This corresponds to drawing a horizontal accusation threshold starting at $(0, \ln n)$, and the pirates are expected to be found around the point $(2c^2 \ln n, \ln n)$.

For the sequential Tardos scheme, we also obtain an asymptotic code length of $\ell \sim 2c^2 \ln n$, and this again corresponds to the asymptotic point $(2c^2 \ln n, \ln n)$; see e.g. Theorems 4.1 and 4.3. However, in the sequential Tardos scheme this accusation threshold is a decreasing line (cf. Figure 5.1b) with a slope equal to $\mu_0 \sim \frac{-1}{2c^2}$. This means that at time $i_0 = 0$, the accusation line starts at $\ln n - (2c^2 \ln n) \cdot \frac{1}{2c^2} = 2 \ln n$. In other words, the accusation threshold starts twice as high as in Wald's SPRT, at the point $(0, 2 \ln n)$. So although both schemes achieve the same asymptotic code length, even in the limit of large n and c these schemes are not quite equivalent.

Overall, in this scenario there may be several reasons to prefer Wald's SPRT approach: it is easier to choose good parameters, and asymptotically the accusation threshold lies slightly lower than in the sequential Tardos scheme.

5.4.3 – Defending against the all-1 attack. Let us further highlight how the sequential Tardos scheme and Wald's SPRT can actually behave quite differently, by showing how both schemes deal with the all-1 attack.

As described in Chapter 4, the Neyman–Pearson approach to the all-1 attack in fingerprinting leads to an optimal decoder of the following form for a user symbol x and pirate output y. In this case we assume p is fixed at the optimal value $p \approx \frac{\ln 2}{c}$.

$$g(x, y) = \begin{cases} \frac{1}{c} \ln(2) & \text{if } (x, y) = (0, 0); \\ \ln\left(2 - 2^{-1/c}\right) & \text{if } (x, y) = (0, 1); \\ -\infty & \text{if } (x, y) = (1, 0); \\ \ln(2) & \text{if } (x, y) = (1, 1). \end{cases} \tag{5.31}$$

In the non-adaptive, simple decoding setting, using this score function leads to a required code length of $\ell \sim \frac{c \ln n}{(\ln 2)^2}$, while in the joint decoding setting the required code length is asymptotically a factor $\ln 2$ less.

Tardos' scheme. In the sequential Tardos scheme, one would again first determine the point (ℓ, η) at which time a decision is taken (cf. Figure 5.1b), and then draw the accusation threshold by drawing a line towards the y-axis, with slope μ_0. Note however that with this score function, the event $(x, y) = (1, 0)$ is impossible for a guilty user (if a member $j \in \mathcal{C}$ has $(\boldsymbol{x}_j)_i = 1$, then by definition $y_i = 1$ as well), and so if this event

occurs we know for sure that this user is innocent, and we assign the user a score of $g(1,0) = -\infty$. Since the probability that this event occurs for innocent users is positive, it immediately follows that $\mu_0 = -\infty$. As a consequence, the accusation threshold starting from (ℓ, η) with a slope of $\mu_0 = -\infty$ becomes a vertical line leaving this point in the upwards direction. This is illustrated in Figure 5.2a which again illustrates how the scheme would work with the toy parameters used before. In this case, the provable bounds from Theorems 4.1 and 5.1 lead to $\ell \approx 459$ and $\eta_1^{(\ell)} \approx 6.91$.

Wald's scheme. In Wald's scheme, choosing scheme parameters is done similarly as before; we only set $\eta_1 = \ln(n/\varepsilon_0) \approx 13.82$ and we are ready to use the scheme. Figure 5.2b shows an example simulation of this scheme with these parameters, using the all-1 score function from (5.31).

Overall, one can see a big difference in the time needed to find the correct set of colluders: around 460 symbols for the Tardos-based scheme, and less than 300 symbols for the solution based on Wald's SPRT.

5.4.4 – Defending against arbitrary attacks. For the general, uninformed fingerprinting game, where it is not known what collusion channel was used, the tracer has to use a decoder that works well against arbitrary collusion channels.

The decoder described in the previous subsection, designed against the interleaving attack, can be used in the uninformed setting as well (cf. Chapter 4), and so a similar construction as in the previous subsection may be used, both in the sequential Tardos scheme and in Wald's sequential scheme. As described in [Laa13a], with this score function it can only be guaranteed that $(\mu_1 - \mu_0)/\sigma_0$ is sufficiently large regardless of the collusion channel, i.e., it is possible to distinguish between the innocent and guilty distributions. However, it could be that for different collusion channels, both μ_0 and μ_1 are smaller than when the interleaving attack is used. To cope with these difficulties, one could normalize the scores: based on y_i and p_i, compute μ_0 and σ_0 for segment i, and translate and scale the scores so that the normalized values μ_0 and σ_0 are the same as for the interleaving attack.

Alternatively, one could use a wide range of different methods, such as using several score functions simultaneously; estimating the collusion channel and using this estimate to choose the score function [CXFF09, FPF09a]; using a generalized linear decoder [AZ10, DHPG13, MF12]; or settle for slightly less and use the suboptimal but 'universal' symmetric score function of [ŠKC08] which works almost the same for any collusion strategy. For small collusion sizes this (asymptotically suboptimal) score function performs quite well [FD14, Figure 3], and it might make designing the scheme somewhat easier.

5.4.5 – Discussion. Let us conclude with a brief overview of the two solutions for the sequential fingerprinting game. For simplicity, we will compare the sequential Tardos scheme with Wald's scheme without a lower boundary, as that seems to be the most convenient solution in fingerprinting. Note that although the sequential Tardos scheme is different from Wald's basic description of the sequential probability ratio test procedure, it could be considered a variant of the latter; truncating the thresholds to force a decision by a certain time ℓ was already considered by Wald himself, and using different shapes for the stopping boundary has been discussed extensively in various literature on the sequential probability ratio test.

Characteristic	Wald	Tardos
Optimal (cf. Theorem 5.2)	✓	✗
Asymptotically optimal	✓	✓
No false negatives ($\varepsilon_1 = 0$)	✓	✗
Guaranteed decision at time ℓ	✗	✓
Parameters to choose	η_1	$\ell, \eta_1^{(0)}, \eta_1^{(\ell)}$

Table 5.1: A quick summary of various characteristics of Wald's SPRT and the sequential Tardos scheme. For Wald's scheme we assume we are not using a lower boundary η_0, i.e., we set $\varepsilon_1 = 0$ and $\eta_0 = -\infty$.

To compare some of the characteristics, Table 5.1 gives a quick summary of the various characteristics of both schemes. Here optimality refers to the optimality described in Theorem 5.2, and asymptotic optimality refers to the large n and large $c \ll n$ regime. Note that by setting $\varepsilon_1 = 0$ in Wald's scheme, we guarantee that eventually all colluders are always caught. This solution of an infinite accusation boundary comes at the cost of not knowing in advance how many segments are at most needed to reach a decision, although in practice this does not seem to be an issue. As we saw in Section 5.4, often choosing parameters is easier for Wald's scheme than for the sequential Tardos scheme; both because fewer parameters have to be chosen, and because there is a simple approximate relation between the single parameter η_1 and the error probability ε_0, which holds exactly if the scores behave like true Brownian motions. We further saw that for the sequential group testing setting, the sequential Tardos scheme offers no improvement over non-adaptive decoding (while Wald's scheme does).

Finally, as mentioned before, Wald's scheme has already been studied since the 1940s, with many papers and books appearing on the topic since [BLS13, Che72, Gov04, JT00, MS09, Sie85, Wal47, WG86], while the sequential Tardos scheme [LDR+13] was more of an ad hoc construction to build a scheme that also works well in adaptive settings. With Wald's scheme being easier to design, in many cases performing better than the sequential Tardos scheme (and with an optimal performance), and being backed by decades of research on the topic, allowing practitioners to tweak the scheme to their needs using various thoroughly analyzed results from the literature, it seems that in most cases Wald's scheme is a more suitable choice than the sequential Tardos scheme.

CHAPTER 6

Applications in group testing

6.1 — Overview

Introduction. In the last chapter of the first part we will study the consequences of the results from the previous chapters to other, related research fields. In particular, we will study how these results apply to the area of group testing. Very recently, results in fingerprinting have also found useful applications in differential privacy; for more information on those applications and the link with non-adaptive and adaptive fingerprinting schemes, see e.g. [BUV14, DTTZ14, SU15, Ull13].

The field of group testing started with the seminal work of Dorfman [Dor43] in the 1940s, who considered the following problem. Suppose a large population of n items or people contains a small number c of infected (or defective) items or people. To identify this subset, it is possible to perform group tests: testing a subset of the population will lead to a positive test result if this subset contains at least one defective item, and a negative result otherwise. As an example, think of testing for the presence of a virus in a blood sample, and mixing blood samples of various people; if one of them is infected, the test of this mixed sample will come back positive, while if none of them is infected, the virus will not appear in the mixed blood sample either. It is commonly assumed that several subsets of blood samples to be tested/mixed are chosen in advance, and all group tests of these mixed samples are then performed simultaneously; this is known in the literature as non-adaptive group testing. (If the groups to be tested are chosen adaptively based on previous test results, we arrive at what is known in the literature as adaptive group testing.) Then, when the test results come back, the hidden subset of defective items needs to be identified. The goal is to identify the subset of defectives using as few group tests as possible, and with a probability of error as small as possible.

Context. It is not hard to see that the model described above (which we will refer to as the classical group testing model) is equivalent to defending against the all-1 attack in fingerprinting: if at least one defective (colluder) is included in the tested pool ($z > 0$), the test result will be positive ($y = 1$, so $\theta_z = 1$); while if none of the defectives (colluders) are included in the group test ($z = 0$), the test result is negative ($y = 0$, so $\theta_0 = 0$). This connection between fingerprinting and group testing was previously made in e.g. [CHS10, KHN+08, MF11a, SvTW00, TWWL03].

Several variants of the classical group testing model have been considered in group testing literature as well, such as noisy group testing, where $\theta_0 \approx 0$ and $\theta_z \approx 1$ for

*This chapter is based on results from [Laa13b, Laa14, Laa15b, Laa15c].

Model	Simple capacities	Joint capacities
θ_{all1}: classical model	$(\ln 2)/c \approx 0.69/c$	$(1)/c \approx 1.00/c$
θ_{add}: additive noise model	$(\ln 2 - r)/c \approx 0.69/c$	$(1 - \frac{1}{2}h(r))/c \approx 1.00/c$
θ_{dil}: dilution noise model	$(\ln 2 - O(r \ln r))/c \approx 0.69/c$	$(1 - \frac{1}{2}h(r)\ln 2)/c \approx 1.00/c$
$\theta_{\text{thr}}^{(u)}$: threshold (no gap)	between $0.46/c$ and $0.69/c$	$(1)/c \approx 1.00/c$
$\theta_{\text{int}}^{(l,u)}$: threshold (int. gap)	between $0.72/c^2$ and $0.69/c$	between $0.84/c^2$ and $1.00/c$
$\theta_{\text{coin}}^{(l,u)}$: threshold (coin. gap)	between $0.17/c$ and $0.69/c$	between $0.32/c$ and $1.00/c$

Table 6.1: An overview of the capacity results for various group testing models.

$z > 0$ [AS12, CCJS11, CHKV09, CHKV11, Hwa76, SJ10, SJ10]; and threshold group testing, where $\theta_z = 0$ for $z \leqslant l$ and $\theta_z = 1$ for $z \geqslant u$ for given thresholds $0 \leqslant l < u \leqslant c$ [ADL11a, ADL11b, ADL13, CCB$^+$13, Che13, Dam06, Leb10]. Analyses of these models have been presented throughout the literature, but no general framework was ever used to analyze all these slightly different models all at once; most papers instead used ad hoc approaches for the specific models under consideration.

Results. Applying results and techniques from Chapters 3–5 to the field of group testing, we obtain several sharper results than those previously known in the group testing literature. We improve upon various previous results on the joint group testing capacities, and derive explicit asymptotics for the simple capacities of various models. For instance, we show that existing simple group testing algorithms of Chan–Jaggi–Saligrama–Agnihotri [CJSA14] are suboptimal, and that simple decoders cannot asymptotically be as efficient as joint decoders. In noisy models with noise parameter r, we further show that $\ell \sim \frac{c \log_2 n}{1-O(r)}$ group tests suffice for joint decoding. More precise results, as well as an overview of all capacity results in this chapter related to group testing, can be found in Table 6.1. As a consequence, previous results in this area of [AS12, CHKV11] are also suboptimal. We further provide explicit (adaptive and non-adaptive) decoding schemes to deal with each of these models efficiently.

Outline. The outline of this chapter is as follows. In Section 6.2 we describe how the information-theoretic approach of computing code rates and channel capacities carries over to the area of group testing, and what results we obtain as a consequence. Sections 6.3 and 6.4 then describe how the decoding schemes proposed in Chapters 3 and 4 can be applied in group testing as well, and how these results compare to previous work in this area.

6.2 — Non-adaptive group testing capacities

We will again study the highest achievable rates of various collusion channels θ, where the choices of θ now correspond to group testing models that may appear in practice.

6.2.1 – Simple capacities. We will study five different models: the classical (noiseless) model, the models with additive noise and dilution noise, and threshold group testing with and without gaps. Other models where the test result y_i depends only on the tally $z_i = \sum_{j \in \mathcal{C}} x_{j,i}$ may be analyzed in a similar fashion.

Classical model. In the classical model, the outcome of a group test is positive iff at least one defective item was present in the tested pool. This model is equivalent to the all-1 attack in fingerprinting, which immediately leads to the following result.

Corollary 6.1. *For the classical group testing model, the simple informed capacity and the corresponding optimal value of* p *are:*

$$C^s(\theta_{all1}) = \frac{\ln 2}{c} + O\left(\frac{1}{c^2}\right) \approx \frac{0.69}{c}, \qquad p^s_{all1} = \frac{\ln 2}{c} + O\left(\frac{1}{c^2}\right) \approx \frac{0.69}{c}. \tag{6.1}$$

In terms of group testing algorithms, this means that any simple decoding algorithm for c defectives and n items requires an asymptotic number of group tests ℓ of at least

$$\ell \sim \frac{c \log_2 n}{\ln 2} \approx 1.44\, c \log_2 n \approx 2.08\, c \ln n, \tag{6.2}$$

where the asymptotics are for $n \to \infty$ and fixed, large c. This improves upon the known lower bound for joint decoders of $\ell \geqslant c \log_2 n$ for large n [Seb85], and this shows that the algorithm of Chan–Che–Jaggi–Saligrama–Agnihotri [CCJS11, CJSA14] (which achieves a code length of $\ell \sim e \ln n$) is suboptimal.

Additive noise. The classical group testing model is sometimes considered to be too optimistic from the tracing point of view, as the outcome of the group tests may not always be accurate. One 'noisy' variant of the classical model that is sometimes considered in the literature is the additive noise model [AS12, CCJS11, CHKV11, SJ10], where a test result may even be positive (with some small probability r) if there were no defectives in the tested group (if $z = 0$). This corresponds to the following channel θ_{add}:

$$(\theta_{\text{add}})_z = \begin{cases} r & \text{if } z = 0; \\ 1 & \text{if } z > 0. \end{cases} \tag{6.3}$$

For small r we do not expect the simple capacity or the optimal choice of p to change drastically compared to the classical model, and the following analysis confirms this.

Proposition 6.2. *For the additive noise model with parameter* r*, the simple capacity and the maximizing value of* p *are:*

$$C^s(\theta_{add}) = \frac{\ln 2}{c}\left(1 - \frac{r}{\ln 2} + O(r^2)\right) + O\left(\frac{1}{c^2}\right), \tag{6.4}$$

$$p^s_{add} = \frac{\ln 2}{c}\left(1 + \frac{r(2\ln 2 - 1)}{2\ln 2(1 - \ln 2)} + O(r^2)\right) + O\left(\frac{1}{c^2}\right). \tag{6.5}$$

Proof. Working out a, a_0 and a_1, and substituting them into $I(p) = p\,d(a_1\|a) + (1-p)\,d(a_0\|a)$, we obtain

$$I(p) = p\,d(1\|1-(1-p)^c(1-r)) + (1-p)\,d((1-p)^{c-1}(1-r)\|(1-p)^c(1-r)). \tag{6.6}$$

For similar reasons as for the all-1 attack, for small values of r the second term is $O(\frac{1}{c^2})$ while the first term is $\Theta(\frac{1}{c})$ and dominates the expression for large c. This means that for small r we have

$$I(p) = -p\log_2(1-(1-p)^c(1-r)) + O\left(\frac{1}{c^2}\right). \tag{6.7}$$

To find the maximum we take the derivative with respect to p and set it equal to 0 to obtain

$$\ln(1-(1-p)^c(1-r)) = -\frac{cp}{1-p}\cdot\frac{(1-p)^c(1-r)}{1-(1-p)^c(1-r)}. \tag{6.8}$$

For small r, the above expression is very close to the one we had for the all-1 attack, and again the optimal value of p is close to $\frac{\ln 2}{c}$. Writing $s = (1-p)^c(1-r)$, so that $p = \frac{-1}{c}\ln(\frac{s}{1-r}) + O(\frac{1}{c^2})$ and $1-p = 1-O(\frac{1}{c})$, the above expression reduces to

$$\ln(1-s) = \ln\left(\frac{s}{1-r}\right)\cdot\frac{s}{1-s} + O\left(\frac{1}{c}\right). \tag{6.9}$$

For small r, this means that $s \approx \frac{1}{2}$, so suppose $s = \frac{1}{2}(1+\varepsilon)$. Filling this in in the above equation, Tayloring around $\varepsilon = 0$, and disregarding terms of the order $\varepsilon^2, r^2, \varepsilon r$, we get

$$-\ln 2 - \varepsilon = (-\ln 2 + r + \varepsilon)(1+2\varepsilon). \tag{6.10}$$

Rearranging the terms, this leads to

$$\varepsilon = -\frac{r}{2(1-\ln 2)} + O(r^2). \tag{6.11}$$

Substituting ε into s and solving for p, we get

$$p = -\frac{1}{c}\ln\left(\frac{1}{2}\cdot\frac{1-\frac{r}{2(1-\ln 2)}}{1-r}\right) + O\left(\frac{1}{c^2}\right) \tag{6.12}$$

$$= \frac{\ln 2}{c} + \frac{r}{c}\cdot\frac{2\ln 2 - 1}{2-2\ln 2} + O\left(\frac{r^2}{c} + \frac{1}{c^2}\right), \tag{6.13}$$

and for the capacity we get

$$I(p) = -\frac{p}{\ln 2}\ln(1-s) \tag{6.14}$$

$$= \left[-\frac{1}{c} + \frac{r}{c\ln 2}\cdot\frac{2\ln 2-1}{2-2\ln 2}\right]\left[-\ln 2 + \frac{r}{c}\cdot\frac{1}{2-2\ln 2}\right] \tag{6.15}$$

$$= \frac{\ln 2}{c}\left(1 - \frac{r}{\ln 2} + O(r^2)\right) + O\left(\frac{1}{c^2}\right). \tag{6.16}$$

These are indeed the given expressions for $C^s(\theta_{add})$ and p^s_{add}. □

For small values of r, the optimal choice for p is to take p slightly smaller than $\frac{\ln 2}{c}$, and the capacity will be slightly lower than in the classical model due to the noise on the channel.

Dilution noise. Another commonly considered noisy group testing model is the dilution noise model [AS12, CHKV09, CHKV11, Hwa76, SJ10], where the probability of a positive test outcome depends on the number of defectives in the tested pool. More precisely, θ_{dil} is defined as follows:

$$(\theta_{\text{dil}})_z = \begin{cases} 0 & \text{if } z = 0; \\ 1 - r^z & \text{if } z > 0. \end{cases} \tag{6.17}$$

Again, for small r this model is similar to the traditional group testing model, so both the capacity and the optimal value of p are close to the values of Proposition 3.3.

Proposition 6.3. *For the dilution noise model with parameter r, the simple capacity and the corresponding optimal value of p are:*

$$C^s(\theta_{dil}) = \frac{\ln 2}{c}\left(1 + \frac{r\ln r}{2\ln 2} - \frac{r(1-\ln 2)}{2\ln 2} + O(r^2 \ln r)\right) + O\left(\frac{1}{c^2}\right) \tag{6.18}$$

$$p^s_{dil} = \frac{\ln 2}{c}\left(1 + \frac{r\ln r}{4\ln 2} + \frac{r(-3(\ln 2)^2 + 5\ln 2 - 1)}{4\ln 2(1-\ln 2)} + O(r^2\ln r)\right) + O\left(\frac{1}{c^2}\right). \tag{6.19}$$

Proof. For a, a_0 and a_1 we get

$$a = 1 - (1 - p + pr)^c, \tag{6.20}$$

$$a_0 = 1 - (1 - p + pr)^{c-1}, \tag{6.21}$$

$$a_1 = 1 - r(1 - p + pr)^{c-1}, \tag{6.22}$$

so letting $s = (1 - p + pr)^c$, the mutual information satisfies

$$I(p) = p\,d\left(\frac{rs}{1-p+pr}\|s\right) + (1-p)\,d\left(\frac{s}{1-p+pr}\|s\right). \tag{6.23}$$

For small r, the second term is again small. Expanding the first term, noting that $p = \Theta(\frac{1}{c})$, we obtain:

$$I(p) = \frac{p}{\ln 2}\left(rs\ln r + (1-rs)\ln\left(\frac{1-rs}{1-s}\right)\right). \tag{6.24}$$

Writing $p = \frac{\ln 2}{c}(1+\varepsilon)$, we can perform a Taylor series expansion of s and rs (disregarding terms of the order $r^2, r\varepsilon^2, \varepsilon^3, \frac{1}{c}$) to obtain

$$s = \frac{1}{2}\left(1 - \varepsilon\ln 2 + r\ln 2 + \varepsilon r\ln 2(1-\ln 2) + \frac{\varepsilon^2}{2}(\ln 2)^2\right). \tag{6.25}$$

This means that up to small order terms, we have $rs = \frac{1}{2}(r - \varepsilon r\ln 2)$. Substituting this into the expression for $I(p)$, we eventually get

$$I(p) = \frac{\ln 2}{c}\Big(1 + r\left(\frac{\ln r - 1 + \ln 2}{2\ln 2}\right) + \varepsilon^2(\ln 2 - 1) \tag{6.26}$$

$$+ \varepsilon r\left(\frac{\ln r(1-\ln 2) - 3(\ln 2)^2 + 5\ln 2 - 1}{2\ln 2}\right)\Big). \tag{6.27}$$

This immediately leads to the given expression for the capacity by disregarding small terms, while differentiating with respect to ε and setting equal to 0 leads to

$$\varepsilon = \left(\frac{\ln r(1-\ln 2) - 1 + 5\ln 2 - 3(\ln 2)^2}{4\ln 2(1-\ln 2)} \right) r + O(r^2). \tag{6.28}$$

This leads to the given expression for p. □

Threshold without gaps. Besides accounting for possible mistakes in the test results (noisy group testing), group testing models have also been considered to account for sensitivity in detecting positive, defective items. In threshold group testing [ADL11a, ADL11b, ADL13, CCB+13, Che13, Dam06, Leb10], it is assumed that if the number of defectives z in the tested pool is at most some constant $l \geqslant 0$ then the test comes back negative, and if z is at least $u > l$ the test result is always positive. For the case $u = l+1$, which we will refer to as threshold group testing without a gap (where $g = u - l - 1$ is the gap size), this completely determines the model:

$$(\theta_{\text{thr}}^{(u)})_z = \begin{cases} 0 & \text{if } z < u; \\ 1 & \text{if } z \geqslant u. \end{cases} \tag{6.29}$$

Although simple to state, even for small u and c finding the simple capacity and optimal choice of p analytically seems very hard, if not impossible. We can intuitively see how the capacity will roughly behave though, since we know that:

- The case $u = 1$ corresponds to $\theta_{\text{thr}}^{(u)} = \theta_{\text{all1}}$, for which we know that asymptotically $p = \frac{\ln 2}{c}$ and $I(p) \approx \frac{\ln 2}{c} \approx \frac{1.44}{c}$ are optimal.
- The case of odd c and $u = \frac{c+1}{2}$ corresponds to $\theta_{\text{thr}}^{(u)} = \theta_{\text{maj}}$, for which $p = \frac{1}{2}$ and $I(p) = \frac{1}{\pi c \ln 2} \approx \frac{0.46}{c}$ are known to be asymptotically optimal.
- The mutual information is symmetric around $u - \frac{1}{2} = \frac{c}{2}$, e.g. $l = 0$ and $u = 1$ leads to the same capacity as $l = c - 1$ and $u = c$.

For values of u between 1 and $\frac{c}{2}$, we expect the capacity to decrease as u increases, and the optimal value p is expected to be close to $\frac{u}{c}$.

Numerical evidence supports this intuition, as it shows that the capacity strictly decreases from $u = 1$ up to $u = \frac{c+1}{2}$, and that the optimal values of p are almost evenly spaced for $u = 1$ up to $u = \frac{c}{2}$. The values of $C^s(\theta_{\text{thr}}^{(u)})$ for various u are shown in Figure 6.1a, which are based on $c = 25$. Note that the capacity quickly drops at small values of u, i.e., the gap between $C^s(\theta_{\text{thr}}^{(1)})$ and $C^s(\theta_{\text{thr}}^{(2)})$ is bigger than the gap between $C^s(\theta_{\text{thr}}^{(2)})$ and $C^s(\theta_{\text{thr}}^{(13)})$ for $c = 25$.

Threshold with gaps. An even harder case to deal with is threshold group testing with $g = u - l - 1 > 0$, which we will refer to as threshold group testing with a gap. If $u > l + 1$, then the model is not yet defined properly, as we do not know what θ_z is for $l + 1 \leqslant z \leqslant u - 1$. Different models were considered to capture the behavior of the outcome of the test results in these gaps, such as: [CCB+13]

- The test outcome is uniformly random:

$$(\theta_{\text{coin}}^{(l,u)})_z = \begin{cases} 0 & \text{if } z \leqslant l; \\ \frac{1}{2} & \text{if } l < z < u; \\ 1 & \text{if } z \geqslant u. \end{cases} \tag{6.30}$$

- The probability of a positive result increases linearly with z:

$$(\theta_{\text{int}}^{(l,u)})_z = \begin{cases} 0 & \text{if } z \leqslant l; \\ \frac{z-l}{u-l} & \text{if } l < z < u; \\ 1 & \text{if } z \geqslant u. \end{cases} \tag{6.31}$$

- We simply do not know what the test outcome will be.

Note that $\theta_{\text{coin}}^{(0,c)} = \theta_{\text{coin}}$ and $\theta_{\text{int}}^{(0,c)} = \theta_{\text{int}}$, so these models can be seen as generalizations of the coin-flip and interleaving attacks in fingerprinting. Also note that $\theta_{\text{coin}}^{(u-1,u)} = \theta_{\text{int}}^{(u-1,u)} = \theta_{\text{thr}}^{(u)}$ for all u.

Regardless of the gap model, for arbitrary l and u these models all seem hard to study analytically. Using results obtained previously, we can however try to 'interpolate' the results to get somewhat decent estimates. For instance, for the first model we can interpolate between the results for threshold group testing without a gap (Section 6.2.1) and the coin-flip attack (Section 3.2.5) to get upper and lower bounds on the simple capacity. For the second case, we can interpolate between threshold group testing without a gap (Section 6.2.1) and the interleaving attack (Section 3.2.1) to estimate how the capacity and the optimal value of p scale for arbitrary l and u.

To verify this intuition, Figures 6.1c and 6.1e show density plots of the capacities (multiplied by c) for both the coin-flip gap model and the interleaving gap model. These plots are based on numerics for $c = 25$, but already show some trends. For instance, there are sharp peaks in the lower left and upper right corner; even when moving on the diagonal, the capacity quickly drops when leaving the corners. The capacities further take their maxima on and near the diagonal. In the coin-flip gap model, the capacity quickly converges to its minimum at $g = c$ as the gap size increases, while this takes longer for the interleaving gap model. Finally, from Propositions 3.3, 3.4, 3.5, and 3.9, we know exactly how the corners and center of each plot behave asymptotically, so we have a decent idea how the capacity scales for large c and arbitrary values of l and u. Note that the values on the diagonal correspond to Figure 6.1a.

6.2.2 – Joint capacities. As in Chapter 3 we will also consider joint code rates for the five models considered above.

Classical model. Since the classical model is equivalent to the all-1 attack in group testing, the following result is immediate.

Corollary 6.4. *For the classical group testing model, the joint capacity and the optimal value of* p *are:*

$$C^j(\theta_{all1}) = \frac{1}{c}, \qquad p_{all1}^j = \frac{\ln 2}{c} + O\left(\frac{1}{c^2}\right). \tag{6.32}$$

This result was previously derived by Malyutov [Mal78] and Sebő [Seb85, Theorem 2], who also showed that $p = 1 - 2^{-1/c} \approx \frac{\ln 2}{c}$ is optimal.

Additive noise. The additive noise model described in Section 6.2.1 was previously studied in the context of capacities in e.g. [AS12, CHKV11, SJ10]. Cheraghchi–Hormati–Karbasi–Vetterli [CHKV11] showed that $C^j(\theta_{\text{add}}) = O(\frac{(1-r)^3}{c})$, while Atia and Saligrama

[AS12] showed that $C^j(\theta_{\text{add}}) = O(\frac{1-r}{c})$. Looking closely at their proof, they show[1] that $I(p) \geqslant \frac{1-r}{ec\ln 2} \approx \frac{1.88(1-r)}{c}$ using $p = \frac{1}{c}$ for large c.

Below we improve upon these results, by (i) providing the exact leading constant on the capacity; (ii) showing exactly how the first order term (in r) scales for small r; and (iii) showing how the optimal value p scales in terms of r.

Proposition 6.5. *For the additive noise model, the joint capacity and the corresponding optimal value of* p *are:*

$$C^j(\theta_{add}) = \frac{1}{c}\left(1 - \tfrac{1}{2}h(r) + O(r^2)\right) + O\left(\frac{1}{c^2}\right), \tag{6.33}$$

$$p^j_{add} = \frac{\ln 2}{c}\left(1 - \frac{r(1+\ln r)}{2\ln 2} + O(r^2)\right) + O\left(\frac{1}{c^2}\right). \tag{6.34}$$

Proof. First, from the definition of θ_{add} it follows that $a = 1-(1-p)^c(1-r)$, $h(\theta_0) = h(1-r)$ and $h(\theta_z) = 0$ for $z > 0$. So the mutual information satisfies

$$I(p) = \frac{1}{c}\left[h((1-p)^c(1-r)) - (1-p)^c h(1-r)\right]. \tag{6.35}$$

Writing $1-\alpha = (1-p)^c$ this can be rewritten to

$$I(s) = \frac{1}{c}\left[h((1-\alpha)(1-r)) - (1-\alpha)h(r)\right]. \tag{6.36}$$

One may recognize this expression as the capacity of the Z-channel [TABB02], which is well-known to be approximately $1 - \frac{1}{2}h(p)$, and the optimal value of α is known to be

$$\alpha = 1 - \frac{1}{(1-p)(1+2^{h(p)/(1-p)})}. \tag{6.37}$$

Substituting $\alpha = 1-(1-p)^c$ and solving for p (for large c), we obtain the given asymptotic expressions for p and $I(p)$. □

Note that this means that any valid group testing algorithm in the additive noise model asymptotically requires at least the following number of tests:

$$\ell \geqslant \frac{c\log_2 n}{1 - \frac{1}{2}h(r) + O(r^2)}\left(1 + O\left(\frac{1}{c}\right)\right). \tag{6.38}$$

Since $r = o(h(r))$ for small r, this shows that the result of [AS12] is slightly off; due to their suboptimal choice of p, they obtained a code length which scales "better" in r, but has a higher leading constant and thus converges to the wrong limit.

[1] The authors of [AS12] have confirmed that the formula below [AS12, Equation (45)] contains a mistake: there should be an extra e in the numerator of the code length T.

Dilution noise. The dilution noise model, as described in Section 6.2.1, was previously studied in the context of lower bounds by Atia and Saligrama [AS12]. In terms of capacities, they showed that for large c, one has $C^j(\theta_{\text{dil}}) = O(\frac{(1-r)^2}{c})$. Again, they were not interested in leading constants, so they fixed p to the suboptimal choice $p = \frac{1}{c}$. We improve upon their result by finding the leading constant explicitly, and proving how p^j_{dil} and $C^j(\theta_{\text{dil}})$ scale in terms of r.

Proposition 6.6. *For the dilution noise model with parameter r, the joint capacity and the corresponding maximizing value of p are:*

$$C^j(\theta_{dil}) = \frac{1}{c}\left(1 - \frac{\ln 2}{2}h(r) + O(r^2)\right) + O\left(\frac{1}{c^2}\right), \tag{6.39}$$

$$p^j_{dil} = \frac{\ln 2}{c}\left(1 + r - \frac{1-\ln 2}{2}h(r) + O(r^2)\right) + O\left(\frac{1}{c^2}\right). \tag{6.40}$$

Proof. For this attack, we have $\theta_z = 1 - r^z$. Let us first look at $h(a)$:

$$h(a) = h\left(\sum_{z=0}^{c}\binom{c}{z}p^z(1-p)^{c-z}(1-r^z)\right) = h(1-(1-p+pr)^c). \tag{6.41}$$

Next, consider a_h:

$$a_h = \sum_{z=1}^{c}\binom{c}{z}p^z(1-p)^{c-z}h(1-r^z). \tag{6.42}$$

For small r, the only significant contribution to the sum comes from the term with $z = 1$:

$$a_h = cp(1-p)^{c-1}h(r) + O(r^2). \tag{6.43}$$

The optimal value of p is again close to $\frac{\ln 2}{c}$; in particular, the value is mostly determined by the term $h((1-p+pr)^c)$, which has a maximum at $(1-p+pr)^c = \frac{1}{2}$. Writing $(1-p+pr)^c = \frac{1}{2}(1+\varepsilon)$, we have

$$p = \frac{1}{c}\left(\ln 2 + r\ln 2 - \varepsilon - r\varepsilon + \frac{\varepsilon^2}{2} + O(r^2, \varepsilon^2 r, \varepsilon^3)\right), \tag{6.44}$$

$$(1-p)^c = \frac{1}{2}\left(1 - r\ln 2 + \varepsilon + r\varepsilon - \frac{\varepsilon^2}{2} + O(r^2, \varepsilon^2 r, \varepsilon^3)\right). \tag{6.45}$$

This means that $I(p) = I(\varepsilon)$ satisfies (neglecting terms of the order $r^2, \varepsilon^2 r, \varepsilon^3, c^{-1}$)

$$I(\varepsilon) \sim 1 - \frac{1}{2}h(r)\ln 2 + \frac{1}{2}\varepsilon h(r)(1-\ln 2) - \frac{\varepsilon^2}{2\ln 2}. \tag{6.46}$$

Taking the derivative with respect to ε and setting it equal to 0, we obtain

$$\varepsilon = \frac{1}{2}h(r)\ln 2(1-\ln 2) + O(r^2). \tag{6.47}$$

Substituting this value for ε in the expressions for p and I, we get the results. □

For the resulting lower bound on the code length ℓ, one thus obtains

$$\ell \sim \frac{c \log_2 n}{1 - \frac{1}{2}h(r)\ln 2 + O(r^2)}. \tag{6.48}$$

So also in the dilution noise model, the first order term in the denominator scales as $h(r)$ rather than r, as one might have guessed from the results of [AS12].

Threshold without gaps. For threshold group testing with $u = l + 1$ (as described in Section 6.2.1) we now consider two different cases for u: $u = \Theta(c)$ and $u = o(c)$. In both cases, the capacity follows directly from Lemma 3.7, and we can obtain accurate asymptotics for p in both cases. The first case is sometimes referred to in the literature as majority group testing [ADL11a, ADL11b, ADL13].

Proposition 6.7. *For the threshold group testing model with $u = \ell + 1$, the joint capacity is $\frac{1}{c}$, and the corresponding maximizing value of p is:*

$$u = \Theta(c): \quad p^j_{thr}[\theta^{(u)}_{thr}] = \frac{1}{c}(u + \xi) \qquad (|\xi| \leqslant 1) \tag{6.49}$$

$$u = o(c): \quad p^j_{thr}[\theta^{(u)}_{thr}] = \frac{1}{c}\left(u - \frac{1}{3} + O\left(\frac{1}{u}\right)\right). \tag{6.50}$$

Proof. From Lemma 3.7 it follows that the capacity is $\frac{1}{c}$ for all u, and that the optimal value of p satisfies $a = \frac{1}{2}$. Writing out a, we have

$$a = \sum_{z=0}^{u-1} \binom{c}{z} p^z (1-p)^{c-z} = \frac{1}{2}. \tag{6.51}$$

The fact that $a = \frac{1}{2}$ roughly means that u is the median of the binomial distribution with c trials and probability of success p. Since the median of a binomial distribution is one of the two integers closest to cp [KB80], it follows that $|u - cp| \leqslant 1$ leading to the result for the case $u = \Theta(c)$.

For the case $u = o(c)$, note that $p = O(\frac{1}{c})$, so $(1-p)^z = 1 - O(p)$ for $z < u$. So we can expand a around $c = \infty$ as:

$$a = (1-p)^c \sum_{z=0}^{u-1} \binom{c}{z} p^z + O\left(\frac{1}{c}\right). \tag{6.52}$$

Since the solution is in the range $p = \Theta(\frac{1}{c})$, let us write $p = \frac{\alpha}{c}$ for some constant α. A Taylor expansion around $c = \infty$ of the binomial coefficients then gives us

$$a = e^{-\alpha} \sum_{z=0}^{u-1} \frac{\alpha^z}{z!} + O\left(\frac{1}{c}\right). \tag{6.53}$$

The condition that $a = \frac{1}{2}$ means that asymptotically, $u - 1$ is the median of the Poisson distribution with parameter $\lambda = \alpha$. Using results about the median of the Poisson distribution [Cho94], we obtain

$$\alpha = u - \frac{1}{3} + O\left(\frac{1}{u}\right). \tag{6.54}$$

Substituting this back into p, we get the result. □

Note that for $u = 1$ and $c \to \infty$, the above approximation says $p \approx \frac{0.67}{c}$, when in reality the asymptotic optimum lies at $p \sim \frac{\ln 2}{c} \approx \frac{0.69}{c}$, showing that already for small values of u the term $u - \frac{1}{3}$ is quite accurate.

Threshold with gaps. For threshold group testing with gaps, let us again consider the two models described in Section 6.2.1: the coin-flip gap model and the interleaving gap model. For both models, we can again interpolate between results obtained earlier in this section to obtain estimates for $C^j(\theta_{\text{coin}}^{(l,u)})$ and $C^j(\theta_{\text{int}}^{(l,u)})$ for various l and u, and verify our intuition numerically (see Figure 6.1). In both plots, from Proposition 6.7 it follows that the diagonals have value $c \cdot C^j(\theta) = 1$, while the upper left corner in Figure 6.1d converges to $\log_2(5/4) \approx 0.32$ (Proposition 3.9) and the upper left corner of Figure 6.1f scales as c^{-1} and converges to 0 (Proposition 3.3). In the left graph, even for small gaps we see that the capacity quickly decreases and approaches the coin-flip capacity. In the right graph, we see that the capacity decreases more gradually as the gap size increases. These density plots are again drawn for $c = 25$.

6.3 — Non-adaptive decoding schemes

6.3.1 – Simple decoders. For group testing, we will consider three models: the classical (noiseless) model and the models with additive noise and dilution noise. Other models where the probability of a positive test result only depends on the tally Z (such as the threshold group testing models considered in the previous sections) may be analyzed in a similar fashion.

We can again expand the expressions of Theorem 4.1 around $c = \infty$ for the optimal values of p from the previous section, but with the added parameter r the resulting formulas are quite a mess. If we also let $\gamma \to 0$ then we can use Theorem 4.3 to obtain the following simpler expressions for simple decoding:

$$\ell(\theta_{\text{all1}}) = \frac{c \ln n}{\ln(2)^2} \left[1 + O\left(\sqrt{\gamma} + \frac{1}{c}\right)\right], \tag{6.55}$$

$$\ell(\theta_{\text{add}}) = \frac{c \ln n}{\ln(2)^2 - r \ln 2 + O(r^2)} \left[1 + O\left(\sqrt{\gamma} + \frac{1}{c}\right)\right], \tag{6.56}$$

$$\ell(\theta_{\text{dil}}) = \frac{c \ln n}{\ln(2)^2 - O(r \ln r)} \left[1 + O\left(\sqrt{\gamma} + \frac{1}{c}\right)\right]. \tag{6.57}$$

To obtain more detailed expressions for ℓ, one could combine Theorems 4.1 and 4.3 with the results from the previous section. For the classical model, working out the details regarding the score function, we obtain the following result.

Corollary 6.8. *For the classical group testing model, the simple decoder for the optimal value* $p = p^s_{all1} \approx \frac{\ln 2}{c}$ *is given by*[2]

$$g(x, y, p^s_{all1}) = g(x, y) = \begin{cases} +1 & \text{if } (x, y) = (0, 0); \\ -1 + O\left(\frac{1}{c}\right) & \text{if } (x, y) = (0, 1); \\ -\infty & \text{if } (x, y) = (1, 0); \\ +c & \text{if } (x, y) = (1, 1). \end{cases} \tag{6.58}$$

[2]For convenience we have scaled g by a factor $(c \ln 2)$, and so also η should be scaled by a factor $(c \ln 2)$.

Using this decoder in combination with the parameters η and ℓ of Theorem 4.1, we obtain a simple group testing algorithm with an optimal asymptotic number of group tests of

$$\ell \sim \frac{c \ln n}{\ln(2)^2} \approx 2.08c \ln n \approx 1.44c \log_2 n. \tag{6.59}$$

This asymptotically improves upon results of [CCJS11, CJSA12] which proposed an algorithm with an asymptotic code length of $\ell \sim ec \ln n \approx 2.72c \ln n$. This other algorithm does have a guarantee of never falsely identifying a non-defective item as defective (whereas our proposed decoder does not have this guarantee), but the price they pay for fixing $\varepsilon_0 = 0$ is a significantly higher asymptotic number of tests required to find the defectives.

6.3.2 – Joint decoders. Similar to the above, for group testing models we can also use Theorem 4.4 to obtain exact expressions for ℓ in terms of $\theta, p, c, n, \varepsilon_0, \varepsilon_1$ with provable error bounds. For the optimal values of p, we may use Theorem 4.6 to obtain the following refined expressions for the required asymptotic code lengths for joint decoding:

$$\ell(\theta_{\text{all1}}) = c \log_2 n \left[1 + O\left(\sqrt{\gamma} + \frac{1}{c}\right)\right], \tag{6.60}$$

$$\ell(\theta_{\text{add}}) = \frac{c \log_2 n}{1 - \frac{1}{2}h(r) + O(r^2)} \left[1 + O\left(\sqrt{\gamma} + \frac{1}{c}\right)\right], \tag{6.61}$$

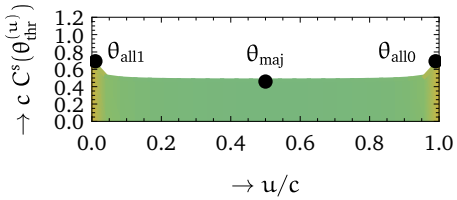
$$\ell(\theta_{\text{dil}}) = \frac{c \log_2 n}{1 - \frac{\ln 2}{2}h(r) + O(r^2)} \left[1 + O\left(\sqrt{\gamma} + \frac{1}{c}\right)\right]. \tag{6.62}$$

Note that as described in Theorem 4.5 the score function for the classical group testing model is equivalent to simply checking whether some subset of c items matches the test results, i.e. whether these would indeed have been the test results, had this subset been the set of defectives. With high probability, only the correct set of defectives passes this test.

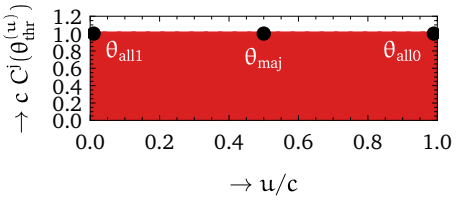
6.4 — Sequential decoding schemes

Let us finally briefly revisit sequential decoding schemes, applied to group testing. As described in Chapter 5, both the sequential Tardos scheme and Wald's sequential probability ratio test procedure work well against different attacks, and the biggest difference between the two solutions appears for the all-1 attack, which happens to be equivalent to the classical group testing model. In that case, as we saw in Figures 5.2e and 5.2f, the sequential Tardos scheme solution is equivalent to non-adaptive group testing (i.e. does not offer any advantage over non-adaptive testing) while Wald's scheme does offer a slightly different and more effective solution. Still, both solutions require an asymptotic number of tests equal to the number of tests required in non-adaptive group testing.

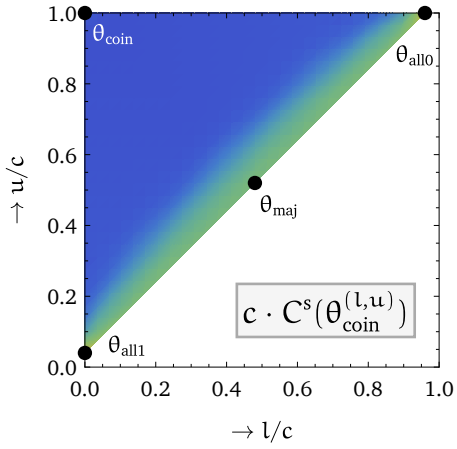
For noisy group testing models with a small amount of noise r, the situation is mostly the same, and the sequential Tardos solution hardly offers any improvement at all over the non-adaptive scheme; the accusation threshold will have a slope of $-\omega(1)$ (as a function of r) rather than $-\infty$, and so it becomes an almost-vertical line rather than a vertical line as in Figure 5.2e. So also for noisy group testing, Wald's SPRT should be preferred over the sequential Tardos scheme solution.



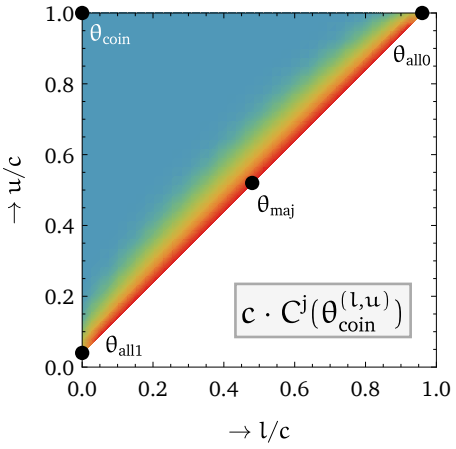
(a) Simple capacity in the gapless model



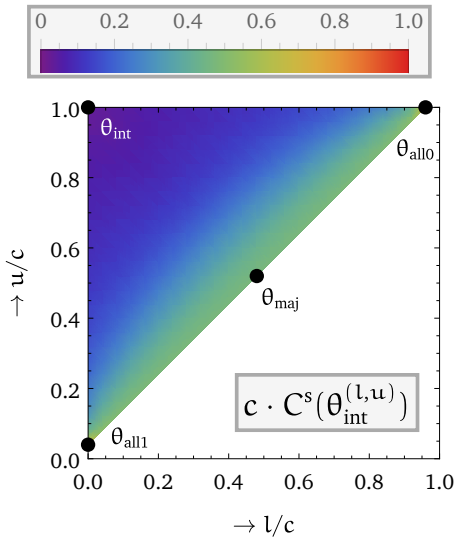
(b) Joint capacity in the gapless model



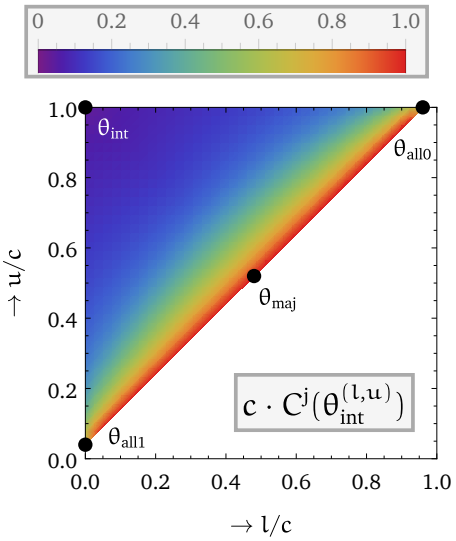
(c) Simple capacity in the coin-flip gap model



(d) Joint capacity in the coin-flip gap model



(e) Simple capacity in the interleaving gap model



(f) Joint capacity in the interleaving gap model

Figure 6.1: The simple (left) and joint capacities (right) for various threshold group testing models. The corners correspond to the all-1, all-0 and coin-flip/interleaving attack, and the centers of the graphs correspond to the majority voting attack in fingerprinting.

CHAPTER 7

Conclusions and open problems

Conclusions

To conclude, we briefly take another look at the research questions posed in Chapter 1, and how these questions were (partially) answered in the first part of this thesis.

Q1. Can the symmetric Tardos scheme be further improved?

In Chapter 2 we saw that the optimal distribution functions in the symmetric Tardos scheme, previously derived in [NHWI07, NFH+09], converge to the well-known arcsine distribution as the number of colluders increases. Together with results of e.g. [LdW14] this implies that the asymptotic code length $\ell \sim \frac{1}{2}\pi^2 c^2 \ln n$ for the Tardos scheme with the symmetric score function of Škorić–Katzenbeisser–Celik [ŠKC08] is already optimal, and cannot be further improved only by performing a different analysis or by using other decoders. As this asymptotic code length is larger than the information-theoretic lower bound $\ell \geqslant 2c^2 \ln n$ for large c and n, we concluded that the only way to get closer to this bound is to use other score functions.

Q2. How difficult is fingerprinting for fixed pirate strategies?

In Chapter 3 we then revisited the results of Huang and Moulin [HM12b], and extended their results to the setting of known pirate strategies. For various attacks commonly considered in the literature, we showed that a code length of $\ell = \Theta(c \ln n)$ suffices if the distributor knows in advance that this attack is used. Surprisingly, we also established that for joint decoding, a tracer can also achieve a code length $\ell \ll 2c^2 \ln n$ when defending against the interleaving attack. This contradicts previous results of Huang and Moulin who previously analyzed both the interleaving attack and arbitrary attacks, and the source of this contradiction was found to be an assumption of Huang and Moulin on the tracer's capabilities. This assumption turned out to be too restrictive: if the distributor is allowed to use any bias distribution function, then he can defend against the interleaving attack with a shorter code length than previously deemed possible by Huang and Moulin.

In Chapter 4 we then studied decoders for fixed pirate strategies known to the distributor, and saw that defending against known attacks is again much easier than defending against arbitrary attacks; not only are the code lengths much shorter in many cases, the proof that the Neyman–Pearson-based simple log-likelihood decoders are capacity-achieving for their respective attacks was straightforward. Overall, the results of Chapters 3 and 4 showed that defending against known attacks is significantly easier than

defending against arbitrary attacks, both in designing the scheme and in the code lengths needed to find the colluders.

Q3. Can the insight for fixed pirate strategies be used for arbitrary attacks?

In Chapters 3 and 4 we also discussed how these results affect the arbitrary-attacks setting. In particular, as the interleaving attack is commonly considered the strongest pirate strategy for large collusions, and as we saw that defending against the interleaving attack is not easier (with a simple decoder) than defending against arbitrary attacks, we focused on the decoder designed against the interleaving attack. For simple decoding, we saw that this decoder is asymptotically essentially equivalent to the decoder of Oosterwijk–Škorić–Doumen [OŠD15], implying that the log-likelihood decoder designed against the interleaving attack is capacity-achieving for arbitrary attacks. Moreover, we argued that with this new decoder we can eliminate the *cutoffs* on the bias distribution, simplifying the practical deployment of these encoders and decoders in practice, and we expect this log-likelihood decoder to work better in practice than the Lagrange-optimized decoder of [OŠD15]. Overall, the proposed decoder may be considered a strong and practical new candidate for defending against arbitrary attacks.

Q4. How much does adaptivity help in tracing collusions?

In Chapter 5 we then moved on to sequential settings, where the tracer is more powerful as the pirate copy is detected in real-time and the tracer is allowed to disconnect users at any point in time. Practical applications of this scenario include live broadcasts of sports events which are directly rebroadcast online by pirates. In these weakly adaptive settings, we saw that the "dynamic Tardos scheme" previously proposed in [LDR$^+$13] is actually just a variant of Wald's celebrated sequential probability ratio test (SPRT) procedure from the 1940s. Looking closely at the related literature, we saw that an immediate consequence of this connection is that with sequential schemes it is impossible to achieve significantly better asymptotic code lengths than in non-adaptive settings and both Wald's scheme and the solution from [LDR$^+$13] are essentially optimal, but that for various reasons outlined in Chapter 5, Wald's scheme may be preferred over the solution presented in [LDR$^+$13].

Q5. Do results in fingerprinting have applications in different fields?

In Chapter 6 we finally saw how all the previous results apply to the area of group testing, which almost exactly corresponds to fingerprinting where the pirate strategy is fixed as the all-1 attack. This again highlights why exploring easier settings of known pirate strategies (as in Chapters 3 and 4) may be a useful exercise, even if there may not be a direct application in fingerprinting. Among others we showed that an asymptotic code length of $\ell \sim (c \ln n)/(\log 2)^2$ is both necessary and sufficient for simple decoding in group testing, which is a factor $1/\ln 2$ higher than with joint decoders, and which improves upon results of e.g. [CJSA14]. We also showed how the code length optimally scales with the noise parameter r in various noisy group testing scenarios, and how the non-adaptive and sequential decoding schemes from Chapters 4 and 5 can be used in group testing as well.

Open problems

By searching for answers to the above research questions, new questions also appeared, and below we state the important questions that we did not answer and which may be a topic for future work.

Q6. What is the joint fingerprinting capacity for arbitrary attacks?

Arguably the biggest open question that was raised by the work in the first part of this thesis is: what is the joint fingerprinting capacity (for large c) for the setting of unknown pirate strategies? This question was previously answered by Huang and Moulin [HM12b], but in Chapter 11 we saw that one of their assumptions on the tracer's capabilities seems to be too restrictive; without this assumption, the tracer may actually be able to defend against arbitrary attacks with a code length $\ell \ll 2c^2 \ln n$ (but with a code length $\ell \geqslant (c^2 \ln n)/\beta$ with $\beta \approx 0.84$ given in Proposition 3.8). If the pirate strategy is fixed as the interleaving attack, then the claimed saddle-point solution of the fingerprinting game of Huang and Moulin is not a saddle-point at all.

Related to analyzing the joint fingerprinting capacity is studying whether the proposed distribution function has any practical merit for the tracer. If we let F_0 denote the arcsine distribution function and F_1 denote the optimized joint encoder for the interleaving attack (where p is always the same value), then clearly using F_0 as the encoder allows the tracer to do reasonably well against arbitrary attacks, while using F_1 may allow the colluders to use a different, unexpected attack, such that tracing the collusion with these extreme values of p becomes hard or impossible. A practical alternative for the tracer may be to use the encoder $F_\lambda \equiv \lambda F_0 + (1-\lambda)F_1$ with $\lambda \in [0,1]$, so that he can defend effectively against both the interleaving attack and against other, weaker attacks. As for defending against various weaker attacks we only need $\ell = \Theta(c \log n) \ll \Theta(c^2 \ln n)$ positions with the right choice of p, one could even mix various distribution functions $F_0, F_1, \ldots$ and choose weights such that one can e.g. defend against all attacks considered in Chapter 3 with a code length of $\ell \sim (c^2 \ln n)/\beta$. Can one devise a pirate strategy which defeats such an encoding strategy?

Q7. What is the best joint decoder for joint fingerprinting games?

In Chapter 4 we also discussed joint decoders, and argued that these are likely to be optimal as well. However, we were not able to prove that these decoders achieve asymptotic code lengths matching the joint capacities. Can it be proved that these joint decoders are capacity-achieving? Or are these decoders not optimal? Should perhaps a different tracing algorithm be used to decide which users to accuse, rather than fixing a threshold for the joint scores and accusing all tuples of users above this threshold? Moreover, for the uninformed setting we know that joint decoding may not necessarily lead to a decrease in the asymptotic code length (the joint capacity may be equivalent to $1/(2c^2 \ln 2)$), but even then an important question in practice remains: how much can one gain with joint decoders for finite values of c and n compared to simple decoders?

Q8. How much does full adaptivity help compared to sequential settings?

In Chapter 5 we discussed how sequential decoding may lead to faster tracing and allows us to catch the whole coalition, rather than part of it. Sequential decoding however assumes that the code is fixed and cannot be adjusted over time. Can fully adaptive schemes,

where the code is also generated on-the-fly, lead to even better schemes? A result of Fiat and Tassa [FT99, FT01] already showed that adaptive schemes may perform much better in certain settings, and the question remains what the impact of full adaptivity is on the required code lengths in the bias-based fingerprinting framework. Establishing upper and lower bounds on adaptive decoding remains an important problem in collusion-resistant fingerprinting.

Q9. How well do the asymptotic analyses reflect practical scenarios?

Finally, although the asymptotics for large c and n conveniently lead to simplifications in the formulas and allow us to describe how these schemes scale as both c and n increase, in practice it is commonly hard to say exactly how these results should be interpreted. The asymptotic formulas for the code lengths and scheme parameters commonly contain order terms with hidden constants, and so for finite parameters it is not clear what exactly the distributor should do to obtain the best performance. Non-asymptotic analyses and practical, numeric investigations of these schemes for finite c and n are just as useful (if not even more useful) as the asymptotic analyses considered in this thesis.

PART II

FINDING NEARBY VECTORS IN LATTICE SIEVING

CHAPTER 8

Sieving for shortest vectors in lattices

8.1 — Problem description

Lattices. Given a set $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\} \subset \mathbb{Z}^d$ of linearly independent, integral d-dimensional vectors, the *lattice* spanned by B is defined as

$$\mathcal{L}(B) = \left\{ \sum_{i=1}^{d} \lambda_i \mathbf{b}_i : \lambda_1, \ldots, \lambda_d \in \mathbb{Z} \right\}. \tag{8.1}$$

In other words, the lattice consists of all integer linear combinations of the basis vectors. A standard example of a lattice is $\mathbb{Z}^d$, generated by the d unit vectors $B = \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d\}$. Note that a basis of a lattice is by no means unique: the lattices spanned by $B = \{(1,0),(0,1)\}$ and $B' = \{(8,3),(5,2)\}$ are both equal to $\mathcal{L}(B) = \mathcal{L}(B') = \mathbb{Z}^2$. More generally, multiplying a basis of a lattice by an integral unimodular matrix (with determinant ± 1) always leads to another basis of the same lattice. For more on lattices, see e.g. [LvdPdW12].

The shortest vector problem (SVP). Given a basis B of a lattice, the *shortest vector problem* (SVP) asks to find a shortest non-zero vector in this lattice, i.e., find a vector $\mathbf{s} \in \mathcal{L}(B)$ such that $\|\mathbf{s}\| = \lambda_1(\mathcal{L}) = \min_{\mathbf{0} \neq \boldsymbol{v} \in \mathcal{L}(B)} \|\boldsymbol{v}\|$, where $\|\boldsymbol{v}\| = (\sum_{i=1}^{d} v_i^2)^{1/2}$ denotes the Euclidean norm[1]. In the example of the lattice $\mathbb{Z}^d$, it can easily be verified that the shortest non-zero vectors are $\pm \boldsymbol{e}_i$ for $i = 1, \ldots, d$. Given an arbitrary basis of an arbitrary lattice in high dimensions, finding a shortest vector is known to be NP-hard under randomized reductions [Ajt98, Mic98]. Even for approximate SVP with constant approximation factors $\alpha > 1$ (SVP_α), where one is tasked to find a lattice vector of norm at most $\alpha \cdot \lambda_1(\mathcal{L})$, it is known that this problem remains NP-hard [Kho04a, Kho04b].

Lattice-based cryptography. One of the main reasons why people have studied the hardness of finding short vectors in lattices is *lattice-based cryptography* [MR09, Reg06, vdP11]. After Shor's breakthrough work on quantum algorithms for e.g. factoring large numbers [Sho94, Sho97], which showed that most of the currently deployed cryptographic primitives can be easily broken with a large-scale quantum computer, the cryptographic community realized that building new cryptographic techniques relying on a different set of "hard problems" may be wise. Various alternatives to the number-theoretic primitives like Diffie–Hellman [DH76] and RSA [RSA78] were proposed since, and one of these lines of research considers basing cryptographic primitives on hard lattice problems like the shortest vector problem. After the pioneering work of Ajtai [Ajt96, AD97,

[1] The problem can be defined for arbitrary norms, but we will restrict our attention to the Euclidean norm.

Ajt98,Ajt99], the potential of lattices for cryptography was soon realized by others, which over the years led to exotic lattice-based cryptographic primitives like fully homomorphic encryption [Gen09] and multilinear maps [GGH13], efficient basic primitives like NTRU [HPS98, HHGPW10] and LWE-based schemes [LPR10, LPR13, Reg05, Reg10], and a growing belief that lattice cryptography may be secure against quantum attacks; despite various efforts, no one has yet found a way to solve hard lattice problems in polynomial time with quantum algorithms. As a result, lattice cryptography is often considered one of the main candidates for "post-quantum cryptography" [BBD09].

Computational hardness of finding short vectors. While the NP-hardness of SVP guarantees (assuming P $\neq$ NP) that solving the shortest vector problem is hard in high dimensions, for practical applications one has to be more precise, as parameters have to be chosen. Choosing large parameters may lead to a higher level of security, but may also make encrypting or decrypting a message slower. Small parameters are preferred, but one has to be careful that the underlying hard lattice problem remains computationally hard. Understanding the exact computational hardness of problems like SVP is crucial for accurately choosing parameters in practice [LP11, RS10, vdPS13]. Note that the complexity of SVP is not only interesting for finding *shortest* vectors in moderate dimensions: Schnorr and Euchner showed how an SVP-oracle in low dimensions can also be used as a tool inside the lattice basis reduction algorithm BKZ [Sch87, SE94] (a generalization of the celebrated LLL algorithm [LLL82, NV10]) to find short (rather than *shortest*) lattice vectors in high dimensions. Most lattice-based cryptographic schemes are broken if a sufficiently short vector is found, and the current state-of-the-art for finding short vectors in high dimensions is using BKZ with an efficient SVP oracle inside [CN11, GNR10, vdPS13].

Algorithms for finding shortest vectors. Currently the four main methodologies for solving SVP are enumeration [FP85, Kan83, Poh81], sieving [AKS01b, AKS01a], constructing the Voronoi cell of the lattice [AEVZ02, MV10a, Vou11], and a recent method based on discrete Gaussian sampling [ADRSD15]. Enumeration has a low space complexity, but a time complexity superexponential in the dimension d, which is suboptimal as the other methods all run in single exponential ($2^{\Theta(d)}$) time. Drawbacks of the other methods are that their space complexities are $2^{\Theta(d)}$ as well, and that the hidden constants in the exponents are relatively big. Enumeration (with extreme pruning [GNR10]) is commonly considered the most practical method for solving SVP in moderate dimensions [MW15].

Sieving algorithms. On the other hand, these newer SVP methods are less explored than enumeration, and recent improvements in sieving have considerably narrowed the gap with enumeration. Whereas the original work of Ajtai–Kumar–Sivakumar [AKS01b] showed only that sieving can solve SVP in time and space $2^{\Theta(d)}$, it was later shown that sieving can provably solve SVP in time $2^{2.465d+o(d)}$ and space $2^{1.233d+o(d)}$ [HPS11, NV08, PS09]. Heuristic analyses further suggest that with sieving one can solve SVP in time $2^{0.415d+o(d)}$ and space $2^{0.208d+o(d)}$ [MV10b, NV08], or optimizing for time, in time $2^{0.378d+o(d)}$ and space $2^{0.293d+o(d)}$ [BGJ14, WLTB11, ZPH13]. Various papers have further studied how to speed up sieving in practice [BNvdP14, FBB+14, IKMT14, LMvdP15, MTB14, MODB14, MS11, Sch11, Sch13], and currently the highest dimension in which sieving was used to solve SVP is 116 for arbitrary lattices [SG15], and 128 for cyclic (ideal) lattices [BNvdP14, IKMT14, PS15], where the additional structure in these lattices was exploited to obtain a polynomial improvement in the time and space complexities.

8.2 — The sieving framework

Sieving algorithms, introduced by Ajtai–Kumar–Sivakumar [AKS01b, AKS01a], attempt to solve the shortest vector problem as follows: sample a long list $L = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n\}$ of random lattice vectors, and consider all pairwise differences $\boldsymbol{w}_i - \boldsymbol{w}_j$ within this list. Note that $\boldsymbol{w}_i, \boldsymbol{w}_j \in \mathcal{L}$ implies that $\boldsymbol{w}_i - \boldsymbol{w}_j \in \mathcal{L}$. Many of these difference vectors are longer than both $\boldsymbol{w}_i$ and $\boldsymbol{w}_j$, but as we started by sampling a very long list of points, we hope that we will also find some difference vectors $\boldsymbol{w}_i - \boldsymbol{w}_j$ which are short, e.g. shorter than $\boldsymbol{w}_i, \boldsymbol{w}_j$. As finding shorter vectors means we are making progress, we will keep the short vectors we find, and repeat the procedure of looking at difference vectors after that. By starting with a sufficiently long initial list of lattice vectors, we hope that we will keep making progress, finding shorter vectors as we go. Ultimately we hope to saturate the space of short lattice vectors with our list of vectors, so that after many iterations, at least one of the difference vectors $\boldsymbol{w}_i - \boldsymbol{w}_j$ is a shortest non-zero lattice vector.

Within the sieving literature there are many different algorithms, and similar to e.g. lattice basis reduction algorithms, for which provable bounds on the quality of the output basis seem far off from the actual output quality, analyses of provable sieving algorithms seem to be very loose. Provable bounds suggest the quality of sieving is very poor: even with the best provable sieving algorithm of Pujol and Stehlé [PS09], showing that SVP can provably be solved in time $2^{2.465d+o(d)}$ and space $2^{1.233d+o(d)}$, one would not be able to solve SVP in dimensions higher than 40. Sieving algorithms seem to perform better in practice though, and one line of research on *heuristic* sieving focuses on analyzing the complexity of sieving under some heuristic assumptions. Naturally these assumptions need to be validated in practice, but experiments suggest that these assumptions are sound, and allow us to predict the performance of sieving more accurately than with provable bounds. For assessing the computational hardness of solving SVP, studying these heuristic algorithms may be more relevant than studying provable sieving methods.

8.2.1 – The Nguyễn–Vidick sieve. The first paper to investigate the complexity of lattice sieving under such heuristic assumptions was the paper of Nguyễn and Vidick [NV08], studying the following algorithm. First, we sample a list L of n lattice vectors using e.g. Klein's algorithm [Kle00], which roughly samples vectors from a discrete Gaussian over the lattice with a large variance (i.e. with a relatively high probability of sampling long vectors). The parameter n is to be chosen later. Then, we apply the sieve described in Algorithm 8.1 to the list L to obtain a list L' of similar size, but with shorter lattice vectors. After applying the sieve we set $L \leftarrow L'$ and we repeat the procedure as many times as needed to either find a shortest vector in the list, or deplete the list due to the shrinking radius of vectors that we store in our list. In that case, we started with a list that was too small, and we need to start over with a longer list.

As described above, the sieve that maps L to L' only looks at difference vectors $\boldsymbol{v} - \boldsymbol{w}$ for $\boldsymbol{v}, \boldsymbol{w} \in L$, and stores the short difference vectors it finds in the new list L'. However, instead of looking at all difference vectors, it keeps track of a list of *centers* $C \subset L$, which each cover part of the space. Then, for each vector in the list, we only check the difference vectors with these center vectors for short vectors. If a short vector is found, we keep it, and if this vector is far away from all center vectors, we add it to the list of centers to cover a part of the space which was not yet covered by the other centers.

To analyze their sieve algorithm and prove bounds on the time and space complexities,

Algorithm 8.1 The Nguyễn–Vidick sieve algorithm (sieving step)

Require: An input list L of $(4/3)^{d/2+o(d)}$ vectors, and a parameter $R := \max_{\boldsymbol{v}\in L} \|\boldsymbol{v}\|$
Ensure: The output list L' has $(4/3)^{d/2+o(d)}$ vectors of norm at most $\gamma \cdot R$
1: Initialize an empty list L' and initialize $C \leftarrow \{\mathbf{0}\}$
2: **for each** $\boldsymbol{v} \in L$ **do**
3: **if** $\exists \boldsymbol{w} \in C : \|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma \cdot R$ **then**
4: Add $\boldsymbol{v} - \boldsymbol{w}$ to the list L' and continue the outer loop over $\boldsymbol{v} \in L$
5: **else**
6: Add $\boldsymbol{v}$ to the centers C

Nguyễn and Vidick used (a slightly stronger version of) the following assumption.

Assumption 8.1. *The angle $\Theta(\boldsymbol{v}, \boldsymbol{w})$ between two list vectors $\boldsymbol{v}, \boldsymbol{w} \in L$ follows the same distribution as the distribution of angles $\Theta(\boldsymbol{v}, \boldsymbol{w})$ between vectors $\boldsymbol{v}, \boldsymbol{w} \in S^{d-1}$ drawn at random from the unit sphere.*

Using this heuristic assumption, Nguyễn and Vidick showed that an initial list of size $n = (4/3)^{d/2+o(d)} \approx 2^{0.208d+o(d)}$ suffices to find a shortest vector for $\gamma \to 1$ as $d \to \infty$ [NV08]. Since the time complexity is dominated by comparing almost every pair of vectors in L in each sieving step (the number of centers is asymptotically equivalent to the total number of vectors in the list), this leads to a time complexity quadratic in n. Overall, this means that under Assumption 8.1, the Nguyễn–Vidick sieve provably solves SVP in time $\tilde{O}(n^2) \approx 2^{0.415d+o(d)}$ and space $\tilde{O}(n) \approx 2^{0.208d+o(d)}$. For validation of this heuristic assumption, see e.g. the original paper [NV08], which reports the running times and number of vectors used for lattices of dimensions 30–50.

8.2.2 – The GaussSieve. Two years after Nguyễn and Vidick's heuristic sieve, Micciancio and Voulgaris proposed another heuristic sieving algorithm, which seems to perform better in practice, but for which to date no bounds are known on the time complexity. A simplified version of this GaussSieve algorithm of Micciancio and Voulgaris is described in Algorithm 8.2. Instead of starting with a long list, and shrinking it as we proceed, this algorithm starts with a short list of vectors, and iteratively builds a longer and longer list of lattice vectors, occasionally reducing the lengths of list vectors in the process, until at some point this list L contains a shortest vector. Reductions are similar to the Nguyễn–Vidick sieve, where differences between pairs of list vectors are considered to find shorter vectors. New vectors to be added to the list are first taken from the stack of vectors which have been temporarily removed from the list, while if the stack is empty we sample new vectors from a discrete Gaussian over the lattice, using e.g. Klein's sampler [Kle00].

In the GaussSieve, the reductions in Lines 5 and 6 follow the rule:

$$\text{Reduce } \boldsymbol{w}_1 \text{ with } \boldsymbol{w}_2 : \quad \text{if } \|\boldsymbol{w}_1 \pm \boldsymbol{w}_2\| < \|\boldsymbol{w}_1\| \text{ then } \boldsymbol{w}_1 \leftarrow \boldsymbol{w}_1 \pm \boldsymbol{w}_2. \tag{8.2}$$

Throughout the execution of the algorithm, the list L is always pairwise reduced w.r.t. (8.2), i.e., $\|\boldsymbol{w}_1 \pm \boldsymbol{w}_2\| \geqslant \max\{\|\boldsymbol{w}_1\|, \|\boldsymbol{w}_2\|\}$ for all $\boldsymbol{w}_1, \boldsymbol{w}_2 \in L$. This implies that two list vectors $\boldsymbol{w}_1, \boldsymbol{w}_2 \in L$ always have an angle of at least $60°$; otherwise one of them would have been used to reduce the other before being added to the list. Since all angles between list vectors are always at least $60°$, the size of L is bounded by the *kissing constant*

Algorithm 8.2 The GaussSieve algorithm

```
1: Initialize an empty list L and an empty stack S
2: repeat
3:     Get a vector v from the stack (or sample a new one)
4:     for each w ∈ L do
5:         Reduce v with w
6:         Reduce w with v
7:         if w has changed then
8:             Remove w from the list L
9:             Add w to the stack S (unless w = 0)
10:    if v has changed then
11:        Add v to the stack S (unless v = 0)
12:    else
13:        Add v to the list L
14: until v is a shortest vector
```

in dimension d: the maximum number of vectors in $\mathbb{R}^d$ that can be constructed such that any two vectors have an angle of at least $60°$. Bounds and conjectures on the kissing constant in high dimensions lead us to believe that the size of the list L will not exceed $2^{0.208d+o(d)}$ [CS99]; if the GaussSieve algorithm structurally encounters longer lists of lattice vectors, then this algorithm could be used to generate lists of vectors in $\mathbb{R}^d$ of length exceeding the best known, long-standing asymptotic lower bound on the kissing constant, which is unlikely. Experiments with the GaussSieve [MV10b] validate that the space complexity is close to $2^{0.208d+o(d)}$ in moderate dimensions.

While the space complexity of the GaussSieve is well understood, there are no proven bounds on the time complexity. One might guess that the time complexity is quadratic in n: at any point in time, each pair of vectors $\boldsymbol{w}_1, \boldsymbol{w}_2 \in L$ was compared at least once to see if one of them could reduce the other. The algorithm further seems to display a similar asymptotic behavior as the NV-sieve in practice [MV10b, NV08], for which the time complexity is heuristically known to be quadratic in n. One might therefore conjecture that the GaussSieve also has a time complexity $\tilde{O}(n^2) \approx 2^{0.415d+o(d)}$, matching experiments with the GaussSieve in high dimensions [Kle14].

8.3 — Searching for nearby vectors

Looking at both sieving algorithms, there are various ways the algorithm can be modified and potentially improved (e.g. the reductions could be done differently), but we will focus on one particular part of these algorithms: the search for nearby vectors for reduction. In the Nguyễn–Vidick sieve, this corresponds to the search in Line 3 for a vector $\boldsymbol{w} \in C$ such that $\|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma R$ (which intuitively means that $\boldsymbol{w}$ is close to $\boldsymbol{v}$), and in the GaussSieve this corresponds to the search for vectors $\boldsymbol{w} \in L$ which lead to a reduction in Lines 5 and 6. Note that e.g. in the Nguyễn–Vidick sieve we are satisfied if we find $\tilde{O}(n)$ difference vectors shorter than γR to add to L'. If we could instantly find the nearby vectors to a given vector $\boldsymbol{v}$, we would be able to reduce the time complexity to $\tilde{O}(n) \approx 2^{0.208d+o(d)}$. Studying faster algorithms for finding nearby vectors in high-dimensional spaces may significantly improve the practicability of sieving.

8.4 — Research questions and outline

With the above previous work in mind, we can formulate various research questions which are ultimately aimed at further reducing the asymptotic time complexity of (heuristically) solving SVP with sieving. Below we present the research questions which will be addressed in the upcoming chapters, and how they will be answered in each chapter.

Q1. Can leveled sieving be further improved?

A recent line of research on heuristic sieving [WLTB11, ZPH13] concerns *leveled sieving*. These works showed how to improve the time complexity to approximately $2^{0.378d+o(d)}$ using three levels of centers in the Nguyễn–Vidick sieve. The time complexity seems to decrease as the number of levels increases, so a natural question is: what is the complexity of four-level sieving or even higher-level sieving? This question is addressed in Chapter 9.

Q2. Can techniques from nearest-neighbor literature be used to speed up sieving?

We then turn our attention to other techniques to speed up the search procedure in sieving, with the main focus being on locality-sensitive hashing (LSH). Similar to leveled sieving, this method relies on partitioning the space into regions, and only searching for nearby vectors within regions. We consider existing LSH methods from the literature in Chapters 10 and 11 and analyze how these methods affect the asymptotics of sieving.

Q3. Can existing NNS techniques be improved (and applied to sieving)?

Then, following up on the previous question, we may ask: can we improve upon NNS methods to achieve even larger speed-ups for sieving? Although one existing LSH method is already known to be optimal within the LSH framework, the hidden constants in that construction are known to be large, and so finding a method which has better order terms may offer a significant improvement in practice. This is done in Chapter 12.

Q4. Can new NNS primitives be designed specifically for settings like sieving?

Whereas lower bounds on NNS and LSH are known which say that existing methods are essentially optimal, there is an important caveat to these results; existing schemes are only known to be optimal in sparse settings where the data set has a size $n = 2^{o(d)}$ subexponential in the dimension, while in sieving we are in the dense setting of $n = 2^{\Theta(d)}$. Can we improve upon LSH for these dense settings? Or can we perhaps step outside the LSH framework to further improve upon the search complexity for dense settings? This question is addressed in Chapter 13.

Q5. How do quantum algorithms affect the asymptotic complexities of sieving?

Finally, lattice cryptography is often advertised as "post-quantum cryptography," i.e. resistant against quantum attacks. This means that there are no known methods to solve e.g. SVP in subexponential time with quantum algorithms, but it does not mean that quantum algorithms may not offer substantial speed-ups to existing methods like sieving. In Chapter 14 we describe the potential impact of quantum algorithms on sieving, and how this affects post-quantum parameter selections for lattice cryptography.

CHAPTER 9

Limitations of leveled sieving

9.1 — Overview

Context. After Ajtai–Kumar–Sivakumar's groundbreaking work on sieving [AKS01b], Nguyễn and Vidick's follow-up work on practical sieving [NV08], and Micciancio and Voulgaris' introduction of the even more practical GaussSieve algorithm [MV10b], several follow-up works focused on finding further improvements to sieving that would make sieving even faster. This included making various practical tweaks and devising optimized (parallel) implementations, leading to polynomial speed-ups [BNvdP14, FBB+14, HPS11, IKMT14, MTB14, MODB14, MS11, Sch11, Sch13], and two lines of research showed how an exponential speed-up can be obtained: the overlattice approach of Becker–Gama–Joux [BGJ14] obtained an asymptotic time complexity of $2^{0.3774d+o(d)}$, and the leveled sieving approach of Wang–Liu–Tian–Bi [WLTB11] and later Zhang–Pan–Hu [ZPH13] showed how to obtain a time complexity of $2^{0.3778d+o(d)}$ with 3-level sieving. These methods both offered a trade-off between space and time, but the similarity seems to end there; Becker–Gama–Joux went into a different direction than both the NV-sieve and the GaussSieve, while the leveled sieving approach stayed close to Nguyễn and Vidick's original approach.

In the leveled sieving framework, one could consider the original algorithm of Nguyễn and Vidick to be the 1-level sieve, and Wang–Liu–Tian–Bi then showed how the 2-level sieve can offer a trade-off between space and time: using exponentially more space, the algorithm achieves a better asymptotic time complexity for solving SVP in high dimensions. The idea of using multiple levels could easily be generalized, and Zhang–Pan–Hu later studied the 3-level sieve, analyzing the asymptotic complexities and showing that yet again, the parameters can be tuned so that more space is used but asymptotically the algorithm finds a shortest vector faster in high dimensions.

Higher-level sieving. A natural question to ask after discovering these results is: what happens when even more levels of sieving are used? One can tell from the analysis in the papers [WLTB11, ZPH13] that analyzing a 4-level sieve or even higher-level sieving will probably be tedious, but the answer may be of significant interest; in 2014 Becker–Gama–Joux set the record for the fastest heuristic SVP algorithm with the overlattice sieve, achieving a time complexity of $2^{0.3774d+o(d)}$, while the 3-level sieve was only just behind with an asymptotic complexity of $2^{0.3778d+o(d)}$ using slightly less space.

*This chapter is based on some previously unpublished notes.

Could a 4-level sieve beat this record and become the fastest algorithm to date for solving SVP?

In this chapter we take another look at the leveled sieving approach, and in particular we revisit the (numerically) optimized parameters and complexities of 2- and 3-level sieving. Whereas the original papers only described numerically-optimized parameters leading to the best complexities, here we describe analytic expressions for the parameters and complexities found by Wang–Liu–Tian–Bi [WLTB11] and Zhang–Pan–Hu [ZPH13]. Looking at the optimal parameters and complexities of the 1-, 2-, and 3-level sieves, we further observe that a pattern seems to exist, and by extrapolating this pattern to higher levels we conjecture what the complexities of higher-level sieving may be.

Relation with the overlattice sieve. Surprisingly, we further establish that the optimized complexities of the 1-, 2-, and 3-level sieves (as well as the time/memory trade-offs of 2- and 3-level sieving) all lie on the same time/memory trade-off curve of Becker–Gama–Joux's seemingly unrelated overlattice sieve algorithm. The conjectured pattern suggests that all higher-level sieves also lie on the exact same trade-off curve, and we therefore conjecture that the optimal time complexity of the 4-level sieve is exactly equal to the complexity of the overlattice sieve, i.e. a time complexity of $2^{0.3774d+o(d)}$ and a space complexity of $2^{0.2925d+o(d)}$.

While this relation with the (time/memory trade-off of the) overlattice sieve raises several new questions, which at this point we are unable to answer, the conjectured pattern also provides us with some (conjectured) answers. If this pattern is correct, then (i) the 4-level sieve will not be faster than the sieve of Becker–Gama–Joux; and (ii) higher-level sieving does not further improve the asymptotic time complexity. As a result, leveled sieving seems to be limited by the same time/memory trade-off curve of overlattice sieving, and to achieve a better heuristic time complexity for sieving, different methods are needed.

Outline. The remainder of this chapter is structured as follows. In Section 9.2 we first recall the Nguyễn–Vidick sieve, and we state the result regarding the optimized complexities slightly differently for the sake of highlighting a pattern later. Sections 9.3 and 9.4 describe the 2- and 3-level sieves, and we show that the optimized parameters and complexities originally found through numerical optimization can be explained analytically. Section 9.5 discusses the conjectured pattern for higher-level sieving, and the mysterious relation with the overlattice sieve.

9.2 — The 1-level sieve of Nguyễn and Vidick

Let us first recall the original sieve algorithm of Nguyễn and Vidick, which may also be considered the 1-level sieve as we will see later. The algorithm is described in Algorithm 9.1. Here we have slightly rephrased the sieving step in the context of leveled sieving, where the parameter γ is now called γ_1 and the list of centers C is called C_1.

For this algorithm, as described in Chapter 8, the best (heuristic) asymptotic time complexity is obtained by letting $\gamma_1 \to 1$ as $d \to \infty$, leading to the following result. A proof of this result can be found in [NV08]. Below we again state the result somewhat differently than before; the reasons for this will become clear in the next sections.

Theorem 9.1. *[NV08] The optimal parameter choice γ_1 for the 1-level sieve, minimizing*

Algorithm 9.1 The 1-level sieve

Require: An input list L of $(4/3)^{d/2+o(d)}$ vectors of norm at most R
Ensure: The output list L' has $(4/3)^{d/2+o(d)}$ vectors of norm at most $\gamma_1 \cdot R$
1: Initialize $L' \leftarrow \emptyset$ and $C_1 \leftarrow \{\mathbf{0}\}$
2: **for each** $\boldsymbol{v} \in L$ **do**
3: **if** $\exists \boldsymbol{w}_1 \in C_1 : \|\boldsymbol{v} - \boldsymbol{w}_1\| \leqslant \gamma_1 \cdot R$ **then**
4: Add $\boldsymbol{v} - \boldsymbol{w}_1$ to the list L'
5: **else**
6: Add $\boldsymbol{v}$ to the list C_1

the overall time complexity, is given by $(\gamma_1) \to (1)$. *Let* $x = \sqrt{4/3} \approx 1.155$ *be a root of*

$$p_1(x) = x^2 - 4x^2 + 4. \qquad (= -3x^2 + 4) \tag{9.1}$$

Then the optimal time and space complexities are $x^{2d+o(d)}$ *and* $x^{d+o(d)}$ *respectively, i.e., the time and space complexities are* $2^{c_{\text{time}} d+o(d)}$ *and* $2^{c_{\text{space}} d+o(d)}$ *with*

$$c_{\text{time}} = 2\log_2(x) \approx 0.4150, \qquad c_{\text{space}} = 1\log_2(x) \approx 0.2075. \tag{9.2}$$

9.3 — The 2-level sieve of Wang–Liu–Tian–Bi

In 2011, Wang–Liu–Tian–Bi [WLTB11] proposed a generalization of the 1-level sieve, where two levels of centers C_1, C_2 are used. The algorithm is described in Algorithm 9.2. This algorithm has two parameters γ_1, γ_2 corresponding to the radii of the balls in the two levels of centers C_1 and C_2, and as $\gamma_1 > 1 > \gamma_2$ is chosen larger than in the original 1-level sieve, each of the vectors in C_1 will now cover a larger part of the space. This means that the list C_1 will be shorter than in the 1-level sieve, as fewer points will be needed to cover the spherical shell of radii $(\gamma_1 R, R)$, but also that vectors inside a ball of radius $\gamma_1 \cdot R$ around a vector $\boldsymbol{w}_1 \in C_1$ may now be longer than before; a vector $\boldsymbol{v} - \boldsymbol{w}_1$ for a nearby vector $\boldsymbol{w}_1$ may no longer be sufficiently short to add this vector to L'.

To overcome the problem of using a larger radius γ_1 for the outer list of centers, the algorithm uses another 1-level sieve inside each of the balls around the center vectors $\boldsymbol{w}_1 \in C_1$. So instead of adding $\boldsymbol{v} - \boldsymbol{w}_1$ to L', we now look for nearby center vectors $\boldsymbol{w}_2 \in C_2^{(\boldsymbol{w}_1)}$ which are at distance at most $\gamma_2 \cdot R$ from $\boldsymbol{v}$, where as in the 1-level sieve we will take $\gamma_2 \approx 1$. In other words, the outer level of centers C_1 defines a partitioning of the spherical shell as the union of balls (intersected with this spherical shell), and within each region we perform a 1-level sieve as before with parameter γ_2. This idea of partitioning the space to reduce the search space is quite commonly used in nearest-neighbor searching as well, and the following chapters will essentially deal with the same technique, but where the partitions of the space are chosen slightly differently[1].

For this algorithm, the following theorem shows which *exact* (rather than numerical) values of γ_1, γ_2 lead to the best time complexity, and how the resulting time and space complexities can be expressed analytically rather than numerically.

[1] In fact, the relation with locality-sensitive hashing does not quite end here, as there appear to be big similarities between the leveled sieving approach discussed here and the leveled locality-sensitive hashing approach discussed in [AINR14, Section 4]. Further exploring these similarities and differences, and potential ways to improve either method is left as an open problem.

Algorithm 9.2 The 2-level sieve

Require: An input list L of $(4/3)^{d/2+o(d)}$ vectors of norm at most R
Ensure: The output list L' has $(4/3)^{d/2+o(d)}$ vectors of norm at most $\gamma_2 \cdot R$
1: Initialize $L' \leftarrow \emptyset$ and $C_1 \leftarrow \emptyset$
2: **for each** $\boldsymbol{v} \in L$ **do**
3: **if** $\exists \boldsymbol{w}_1 \in C_1 : \|\boldsymbol{v} - \boldsymbol{w}_1\| \leqslant \gamma_1 \cdot R$ **then**
4: **if** $\exists \boldsymbol{w}_2 \in C_2^{(\boldsymbol{w}_1)} : \|\boldsymbol{v} - \boldsymbol{w}_2\| \leqslant \gamma_2 \cdot R$ **then**
5: Add $\boldsymbol{v} - \boldsymbol{w}_2$ to the list L'
6: **else**
7: Add $\boldsymbol{v}$ to the list $C_2^{(\boldsymbol{w}_1)}$
8: **else**
9: Add $\boldsymbol{v}$ to the list C_1
10: Initialize $C_2^{(\boldsymbol{v})} \leftarrow \{\boldsymbol{v}\}$

Theorem 9.2. *The optimal parameter choice (γ_1, γ_2) for the 2-level sieve, minimizing the overall time complexity, is given by $(\gamma_1, \gamma_2) \to (x, 1)$ where $x \approx 1.093$ is a root of*

$$p_2(x) = x^6 - 4x^4 + 4. \tag{9.3}$$

Furthermore, the optimal time and space complexities are $x^{3d+o(d)}$ and $x^{2d+o(d)}$ respectively, i.e., the time and space complexities are $2^{c_{\text{time}}\, d+o(d)}$ and $2^{c_{\text{space}}\, d+o(d)}$ with

$$c_{\text{time}} = 3\log_2(x) \approx 0.3836, \qquad c_{\text{space}} = 2\log_2(x) \approx 0.2557. \tag{9.4}$$

Proof. We will rely on previous analysis and notation of Wang–Liu–Tian–Bi, and we will assume the reader is somewhat familiar with the original paper [WLTB11]. The parameters γ_1, γ_2 must satisfy $\gamma_2 < 1 < \gamma_1 < \sqrt{2}\cdot\gamma_2$, and omitting sub-exponential terms, there are constants $n_1 = (c_{\mathcal{H}_1})^d$ and $n_2 = (c_{\mathcal{H}_2}/d_{\min})^d$ describing the costs of the algorithm: $n_1 = |C_1|$ is the total number of outer centers used in the algorithm, and $n_2 = |C_2^{(\boldsymbol{w}_1)}|$ counts the number of inner centers in one of the lists of centers associated to a vector $\boldsymbol{w}_1 \in C_1$. The constants $c_{\mathcal{H}_1}, c_{\mathcal{H}_2}, d_{\min}$ can be shown to be functions of γ_1 and γ_2 as follows, as demonstrated in [WLTB11]:

$$c_{\mathcal{H}_1} = \frac{1}{\gamma_1\sqrt{1-\frac{1}{4}\gamma_1^2}}, \qquad c_{\mathcal{H}_2} = \frac{\gamma_1}{\gamma_2}\sqrt{1-\frac{\gamma_1^2}{4\gamma_2^2}}, \qquad d_{\min} = \gamma_2\sqrt{1-\frac{\gamma_2^2 c_{\mathcal{H}_1}^2}{4}}. \tag{9.5}$$

The overall time and space complexities of this algorithm (ignoring sub-exponential factors) can be described as $n_1 n_2 (n_1 + n_2)$ and $n_1 n_2$ respectively: we need to store $n_1 \cdot n_2$ center vectors, and for each of these vectors we need to perform a search over C_1 and (potentially) another search over a list $C_2^{(\boldsymbol{w}_1)}$.

First, as previously observed in [WLTB11] we should fix $\gamma_2 \to 1$ as that leads to the smallest value n_2 (and n_1 does not depend on γ_2). For the remaining parameter γ_1, we observe that as n_1 is decreasing with γ_1 and n_2 is increasing with γ_2 at roughly comparable rates, the optimal time complexity is obtained when $n_1 \simeq n_2$, i.e., n_1 and n_2

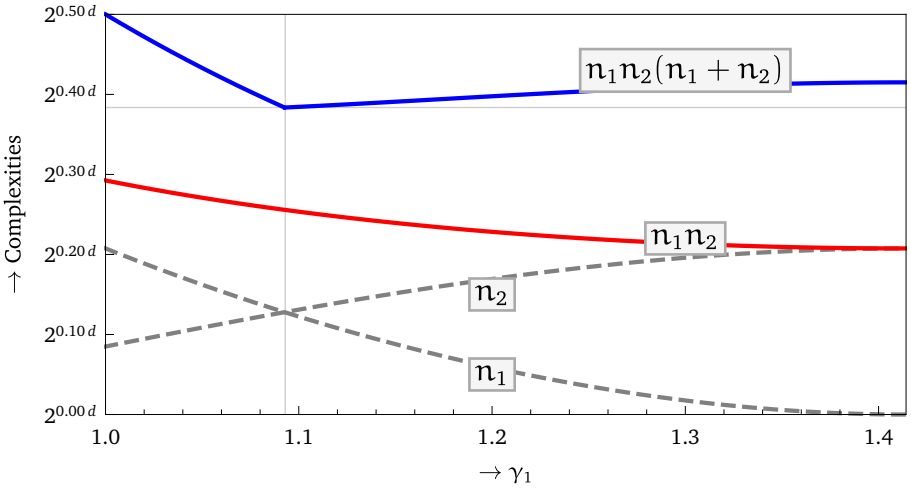


Figure 9.1: The list sizes n_1 and n_2 (gray, dashed), the time complexity $n_1n_2(n_1+n_2)$ (blue) and the space complexity n_1n_2 (red) as functions of γ_1, for the optimal choice $\gamma_2 \to 1$.

are asymptotically equivalent up to subexponential terms. See e.g. Figure 9.1 illustrating the complexities as a function of γ_1 for $\gamma_2 \to 1$, with the optimum lying at $\gamma_1 \approx 1.09$.

Since the optimal time complexity lies at a point where $n_1(\gamma_1,\gamma_2) \simeq n_2(\gamma_1,\gamma_2)$, and we know that taking $\gamma_2 \to 1$ is optimal, we only need to find the value of $\gamma_1 \approx 1.1$ for which these expressions are asymptotically equivalent. We will show that for $\gamma_1 = x$ (where x is a root of $p_2(x)$) these values are asymptotically equivalent. It can then be easily verified that exactly one root x of $p_2(x)$ lies in the required interval $(1,\sqrt{2})$.

Since $n_1 = (c_{\mathcal{H}_1})^d$ and $n_2 = (c_{\mathcal{H}_2}/d_{\min})^d$, to show that $n_1 \simeq n_2$ we need to show that $c_{\mathcal{H}_1} = c_{\mathcal{H}_2}/d_{\min}$, or equivalently $d_{\min} = c_{\mathcal{H}_2}/c_{\mathcal{H}_1}$. Note that for $\gamma_2 = 1$, $c_{\mathcal{H}_1}$ is equal to $1/c_{\mathcal{H}_2}$, so this is equivalent to showing $d_{\min} = c^2_{\mathcal{H}_2}$. Writing out the expressions for $d_{\min}$ and $c_{\mathcal{H}_2}$ with $(\gamma_1,\gamma_2) = (x,1)$, this translates to the condition:

$$(d_{\min} =)\ \sqrt{1-\frac{c^2_{\mathcal{H}_1}}{4}} = x^2\left(1-\frac{x^2}{4}\right)\ \ (= c^2_{\mathcal{H}_2}). \tag{9.6}$$

Multiplying both sides by 4 and taking squares (both sides are positive), we obtain:

$$16 - 4c^2_{\mathcal{H}_1} = x^4\left(4-x^2\right)^2. \tag{9.7}$$

Substituting the expression for $c_{\mathcal{H}_1}$ and working out the left hand side, we obtain

$$(16 - 4c^2_{\mathcal{H}_1} =)\ \ 16 - \frac{16}{x^2(4-x^2)} = x^4\left(4-x^2\right)^2. \tag{9.8}$$

Multiplying by $x^2(4-x^2)$ and expanding the polynomials on both sides, we finally get

$$64x^2 - 16x^4 - 16 = 64x^6 - 48x^8 + 12x^{10} - x^{12}. \tag{9.9}$$

Algorithm 9.3 The 3-level sieve

Require: An input list L of $(4/3)^{d/2+o(d)}$ vectors of norm at most R
Ensure: The output list L' has $(4/3)^{d/2+o(d)}$ vectors of norm at most $\gamma_3 \cdot R$

```
Initialize L' ← ∅ and C_1 ← ∅
for each v ∈ L do
    if ∃w_1 ∈ C_1 : ‖v − w_1‖ ⩽ γ_1 · R then
        if ∃w_2 ∈ C_2^(w_1) : ‖v − w_2‖ ⩽ γ_2 · R then
            if ∃w_3 ∈ C_3^(w_2) : ‖v − w_3‖ ⩽ γ_3 · R then
                Add v − w_3 to the list L'
            else
                Add v to the list C_3^(w_2)
        else
            Add v to the list C_2^(w_1)
            Initialize C_3^(v) ← {v}
    else
        Add v to the list C_1
        Initialize C_2^(v) ← {v}
```

Bringing all terms to one side, and factoring the resulting polynomial, we get

$$(x^3 - 4x - 2)(x^3 - 4x + 2)p_2(x) = 0. \tag{9.10}$$

Since $p_2(x) = 0$ by definition of x, it follows that $n_1 \simeq n_2$ for this choice of γ_1. Note that the other factors $x^3 - 4x \pm 2$ do not have a root $x \in [1, \sqrt{2}]$, i.e., the given value x is the unique solution to $n_1 \simeq n_2$ in the interval $[1, \sqrt{2}]$.

Finally, in the optimal point $(\gamma_1, \gamma_2) = (x, 1)$ we have $n_1 \simeq n_2$ and so the total time and space complexities are of the order $n_1 n_2 (n_1 + n_2) \simeq n_1^3$ and $n_1 n_2 \simeq n_1^2$ respectively. Since $n_1 = (c_{\mathcal{H}_1})^d$, let us take a closer look at $c_{\mathcal{H}_1}$:

$$c_{\mathcal{H}_1} = \frac{1}{x\sqrt{1 - \frac{x^2}{4}}} = \sqrt{\frac{4}{4x^2 - x^4}} = \sqrt{\frac{4x^2}{(-x^6 + 4x^4 - 4) + 4}} = \sqrt{\frac{4x^2}{4}} = x. \tag{9.11}$$

So the time and space complexities are $n_1^3 \simeq x^{3d}$ and $n_1^2 \simeq x^{2d}$ respectively. □

Note that the time exponent $c_{\text{time}} \approx 0.3836$ described in the original paper [WLTB11] is actually exactly equal to $3/2$ times the space complexity exponent $c_{\text{space}} \approx 0.2557$ of the 2-level sieve. We further observe that the given time and space exponents match the approximate solutions described in [WLTB11] found through numerical optimization.

9.4 — The 3-level sieve of Zhang–Pan–Hu

We next turn our attention to 3-level sieving, which has previously been studied by Zhang–Pan–Hu [ZPH13]. This algorithm uses three levels of centers, three parameters $\gamma_3 < 1 < \gamma_2 < \gamma_1$, and is described in Algorithm 9.3. It naturally extends the 1- and 2-level sieves to a sieve using three levels of centers.

In this case we have three parameters $(\gamma_1, \gamma_2, \gamma_3)$, and after eliminating one of them by observing that $\gamma_3 \to 1$ is again optimal for minimizing the asymptotic complexities, we are left with a complex two-dimensional optimization problem. To restrict the search space, we make the following natural assumption on the optimal parameter choice $(\gamma_1, \gamma_2, \gamma_3)$ minimizing the asymptotic time complexity:

Assumption 9.3. *The optimal ratio $\frac{\gamma_{i+1}}{\gamma_i}$ between successive radii does not depend on i.*

In other words, for the 3-level sieve this assumption implies that $\gamma_3/\gamma_2 = \gamma_2/\gamma_1$. We argue that this is a natural assumption as fixing this ratio roughly means that the consecutive searches for nearby center vectors are equally time- and space-consuming. This means that the costs of the searches in different levels are intuitively roughly balanced, and so the overall time complexity is expected to be smallest under this assumption. We cannot prove that this assumption is not too restrictive though, but without this assumption we cannot prove which parameter choice is optimal for 3-level sieving. Note that the numerically-optimized parameters found by Zhang–Pan–Hu do satisfy this assumption, and the parameters we find with this assumption match the parameters obtained in the original paper.

Under this assumption, note that as $\gamma_3 \to 1$ is again optimal, it follows that $\gamma_2 = \sqrt{\gamma_1}$ and we are left with a one-dimensional optimization problem over γ_1, the solution of which is given by the following theorem. Again, the numerical constants found by Zhang–Pan–Hu can be expressed as (functions of) a root of a certain polynomial. In this case, the optimized time complexity exponent is exactly a factor $\frac{4}{3}$ higher than the corresponding space complexity exponent.

Theorem 9.4. *Under Assumption 9.3, the optimal parameter choice $(\gamma_1, \gamma_2, \gamma_3)$ for the 3-level sieve, minimizing the asymptotic time complexity, is $(\gamma_1, \gamma_2, \gamma_3) = (x^2, x, 1)$ where $x \approx 1.067$ is a root of*

$$p_3(x) = x^{10} - 4x^6 + 4. \tag{9.12}$$

Furthermore, the optimal time and space complexities are $x^{4d+o(d)}$ and $x^{3d+o(d)}$ respectively, i.e., the time and space complexities are $2^{c_{\text{time}} d + o(d)}$ and $2^{c_{\text{space}} d + o(d)}$ with

$$c_{\text{time}} = 4\log_2(x) \approx 0.3778, \qquad c_{\text{space}} = 3\log_2(x) \approx 0.2833. \tag{9.13}$$

Proof. We will again rely on the analysis and notation of the 3-level sieve of Zhang–Pan–Hu [ZPH13]. The three radii $\gamma_1, \gamma_2, \gamma_3$ must satisfy $0.88 < \gamma_3 < 1 < \gamma_2 < \gamma_1 < \sqrt{2} \cdot \gamma_3$, and ignoring sub-exponential factors, there are constants $n_1 = (c_{\mathcal{H}_1})^d$, $n_2 = (c_{\mathcal{H}_2}/d_{\min})^d$, and $n_3 = (d_{\max}/r_{\min})^d$ describing the costs of the algorithm, i.e., describing the size of a list of centers at each of the three levels. The constants $c_{\mathcal{H}_1}$, $c_{\mathcal{H}_2}$, $c_{\mathcal{H}_3}$, $d_{\min}$, $d_{\max}$, $r_{\min}$ are given below, and can also be found in [ZPH13][2].

$$c_{\mathcal{H}_1} = \frac{1}{\gamma_1\sqrt{1-\frac{1}{4}\gamma_1^2}}, \quad c_{\mathcal{H}_2} = \frac{\gamma_1}{\gamma_3}\sqrt{1-\frac{\gamma_1^2}{4\gamma_3^2}}, \quad c_{\mathcal{H}_3} = \gamma_2^2\left(1-\frac{\gamma_2^2 c_{\mathcal{H}_1}^2}{4}\right), \tag{9.14}$$

[2]The authors of [ZPH13] have confirmed that in the formula for $d_{\max}$ in their paper, a term γ_3 was missing in the denominator of the third term inside the square root due to a typo.

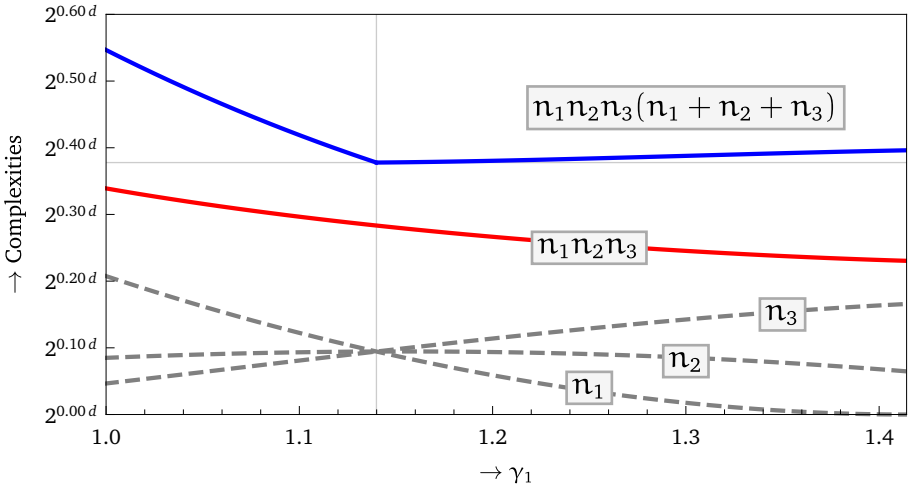


Figure 9.2: The list sizes n_1, n_2, n_3 (gray, dashed), the time complexity $n_1n_2n_3(n_1+n_2+n_3)$ (blue) and the space complexity $n_1n_2n_3$ (red) as functions of γ_1, for the optimal choice $\gamma_3 \to 1$ and under the assumption that $\gamma_2 = \sqrt{\gamma_1}$.

$$r_{\min} = \gamma_3\sqrt{1-\frac{\gamma_3^2}{4c_{\mathcal{H}_3}}}\,, \quad d_{\min} = \gamma_2\sqrt{1-\frac{\gamma_2^2c_{\mathcal{H}_1}^2}{4}}\,, \tag{9.15}$$

$$d_{\max} = \sqrt{1-\left(\frac{\gamma_3^2-\gamma_1^2+1}{2\gamma_3}\right)^2-\left(\frac{\gamma_3^2-\gamma_2^2+1}{2\gamma_3^2c_{\mathcal{H}_2}^2}-\frac{(2\gamma_3^2-\gamma_1^2)\,(\gamma_3^2-\gamma_1^2+1)}{4\gamma_3^4c_{\mathcal{H}_2}^2}\right)^2}\,. \tag{9.16}$$

Up to subexponential factors, the overall time and space complexities of the 3-level sieve can now be expressed as $n_1n_2n_3(n_1+n_2+n_3)$ and $n_1n_2n_3$ respectively; for each of the n_1 centers $\boldsymbol{w}_1 \in C_1$, we have a list of roughly n_2 centers $\boldsymbol{w}_2 \in C_2^{(\boldsymbol{w}_1)}$, and for each of these centers we have a list of n_3 centers $\boldsymbol{w}_3 \in C_3^{(\boldsymbol{w}_2)}$, leading to a space complexity of $n_1n_2n_3$. For the time complexity, for each vector we need to search for nearby vectors in the three consecutive layers of centers (cost $n_1+n_2+n_3$), and we need to perform these searches for each of the $n_1n_2n_3$ vectors in L.

First, note that again $\gamma_3 \to 1$ is optimal, as n_1 and n_2 do not depend on γ_3 and n_3 is decreasing with γ_3. Furthermore, from Assumption 9.3 we know that $\gamma_2^2 = \gamma_1$ and we only have to perform a one-dimensional optimization over γ_1. Similar to the 2-level sieve, one can verify that n_1 is decreasing with γ_1, n_3 is increasing with γ_1, and n_2 has a maximum for $\gamma_1 \approx 1.154$; see e.g. Figure 9.2. All three curves intersect at one value γ_1, and at this point the minimum time complexity is attained. We will show that for the given parameter choice, n_1 and n_2 are asymptotically the same, leading to the best time complexity. Proving that n_3 is also the same can be done analogously.

To show that n_1 and n_2 are equal for the given parameters $(\gamma_1, \gamma_2, \gamma_3)$, we need to show that $c_{\mathcal{H}_1} = c_{\mathcal{H}_2}/d_{\min}$, or $d_{\min} = c_{\mathcal{H}_2}/c_{\mathcal{H}_1}$. Note that for $\gamma_3 = 1$ we have $c_{\mathcal{H}_2} = 1/c_{\mathcal{H}_1}$, so this is equivalent to showing $d_{\min} = c_{\mathcal{H}_2}^2$. Writing out these expressions

with $(\gamma_1, \gamma_2, \gamma_3) = (x^2, x, 1)$, we obtain:

$$(d_{\min} =)\ x\sqrt{1 - \frac{x^2 c_{\mathcal{H}_1}^2}{4}} = x^4\left(1 - \frac{x^4}{4}\right)\ \ (= c_{\mathcal{H}_2}^2). \tag{9.17}$$

Taking squares, again noting that both sides are positive, this is equivalent to

$$x^2\left(1 - \frac{x^2}{x^4(4-x^4)}\right) = x^8\left(1 - \frac{x^4}{4}\right)^2. \tag{9.18}$$

Simplifying this expression, getting rid of all fractions, expanding all polynomials, and bringing all terms to one side, we finally obtain

$$(x^5 - 4x + 2)(x^5 - 4x - 2)p_3(x) = 0. \tag{9.19}$$

As x is a root of p_3 and lies in the interval $[1, \sqrt[4]{2}]$, this proves that indeed $n_1 \simeq n_2$ for the given parameters. Note that again the other factors $x^5 - 4x \pm 2$ do not have any roots in this interval, so the given value x is the unique solution to $n_1 \simeq n_2$.

Finally, in the point $(\gamma_1, \gamma_2, \gamma_3) = (x^2, x, 1)$ we have $n_1 \simeq n_2 \simeq n_3$ and the total time and space complexities are given by $n_1 n_2 n_3 (n_1 + n_2 + n_3) \simeq n_1^4$ and $n_1 n_2 n_3 \simeq n_1^3$. Since $n_1 = (c_{\mathcal{H}_1})^d$, we again take a closer look at $c_{\mathcal{H}_1}$:

$$c_{\mathcal{H}_1} = \frac{1}{x^2\sqrt{1 - \frac{x^4}{4}}} = \sqrt{\frac{4}{4x^4 - x^8}} = \sqrt{\frac{4x^2}{(-x^{10} + 4x^6 - 4) + 4}} = \sqrt{\frac{4x^2}{4}} = x. \tag{9.20}$$

So the overall space complexity is $n_1^3 \simeq x^{3d}$ and the time complexity is $n_1^4 \simeq x^{4d}$. □

Note that not only the optimized time complexity for the 3-level sieve is attained under Assumption 9.3: also the optimal trade-off between the time and space complexities, by varying $\gamma_1, \gamma_2, \gamma_3$, is obtained by taking $(\gamma_1, \gamma_2, \gamma_3) = (\alpha^2, \alpha, 1)$ and varying $\alpha \in [1, \sqrt[4]{2}]$, as can be verified by plotting the complexities for general $(\gamma_1, \gamma_2, \gamma_3)$. Other choices $(\gamma_1, \gamma_2, \gamma_3)$ are all worse, as they lie above and to the right of the curve sketched in Figure 9.3. The same figure also shows that the trade-offs of 2-level and 3-level sieving overlap for part of the curve.

9.5 — High-level sieving

The approach of introducing multiple layers of centers can easily be extended to higher levels $k \geqslant 4$ as well. Extrapolating from the previous algorithms, a k-level sieve would take as input k parameters $\gamma_1, \ldots, \gamma_k$, defining the radii of balls in different levels of centers, and the resulting algorithm would be as described in Algorithm 9.4.

As analyzing arbitrary-level sieves in a similar fashion as the 1-, 2-, and 3-level sieves of [NV08, WLTB11, ZPH13] will lead to terribly complicated expressions and optimizations, we will not perform this analysis, and instead we will focus on how we can make an educated guess regarding the complexities of higher-level sieving, based on the previous results. In particular, looking at the results of Theorems 9.1, 9.2, 9.4, there seems to be a pattern in the optimized parameters and complexities of k-level sieving, which leads us to the following conjecture.

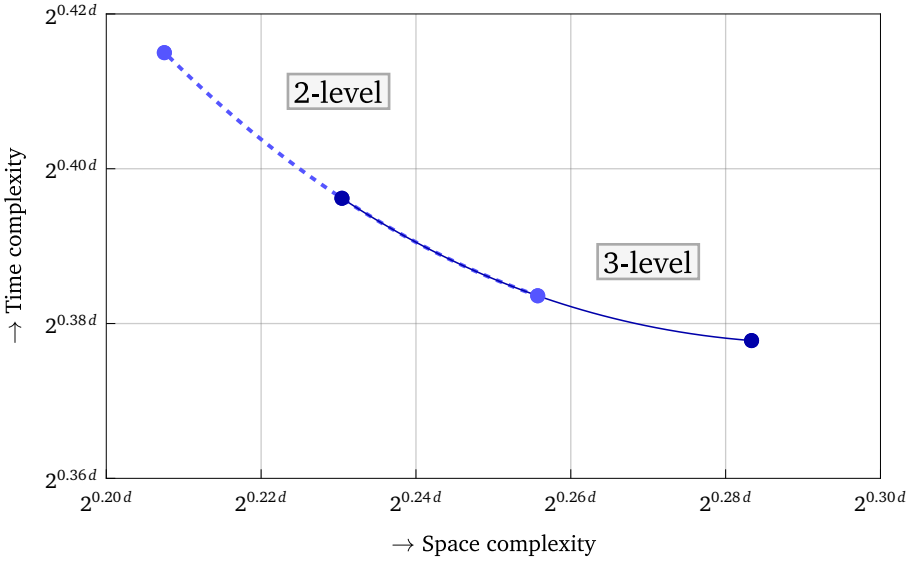


Figure 9.3: The time/memory trade-offs for 2-level sieving with parameters $(\gamma_1, \gamma_2) = (\alpha, 1)$ for $\alpha \in [x, \sqrt{2}]$ where x is a root of $p_2(x)$ (dashed thick curve) and for 3-level sieving with parameters $(\gamma_1, \gamma_2, \gamma_3) = (\alpha^2, \alpha, 1)$ for $\alpha \in [x, \sqrt[4]{2}]$ where x is a root of $p_3(x)$ (solid thin curve).

Algorithm 9.4 The k-level sieve

Require: An input list L of $(4/3)^{d/2+o(d)}$ vectors of norm at most R
Ensure: The output list L' has $(4/3)^{d/2+o(d)}$ vectors of norm at most $\gamma_k \cdot R$

Initialize $L' \leftarrow \emptyset$ and $C_1 \leftarrow \{\mathbf{0}\}$
for each $\boldsymbol{v} \in L$ **do**
  if $\exists \boldsymbol{w}_1 \in C_1 : \|\boldsymbol{v} - \boldsymbol{w}_1\| \leqslant \gamma_1 \cdot R$ **then**
    if $\exists \boldsymbol{w}_2 \in C_2^{(\boldsymbol{w}_1)} : \|\boldsymbol{v} - \boldsymbol{w}_2\| \leqslant \gamma_2 \cdot R$ **then**
      [...]
      if $\exists \boldsymbol{w}_k \in C_k^{(\boldsymbol{w}_{k-1})} : \|\boldsymbol{v} - \boldsymbol{w}_k\| \leqslant \gamma_k \cdot R$ **then**
        Add $\boldsymbol{v} - \boldsymbol{w}_k$ to the list L'
      else
        Add $\boldsymbol{v}$ to the list $C_k^{(\boldsymbol{w}_{k-1})}$
      [...]
    else
      Add $\boldsymbol{v}$ to the list $C_2^{(\boldsymbol{w}_1)}$
      Initialize $C_3^{(\boldsymbol{v})} \leftarrow \{\boldsymbol{v}\}$
  else
    Add $\boldsymbol{v}$ to the list C_1
    Initialize $C_2^{(\boldsymbol{v})} \leftarrow \{\boldsymbol{v}\}$

Algorithm	**Parameters**	**Polynomial**	**Complexities**		**Exponents**	
Name	Variables = Values	$(x : p(x) = 0)$	Time	Space	c_{time}	c_{space}
1-level sieve	$(\gamma_1) = (1)$	$x^2 - 4x^2 + 4$	x^{2n}	x^n	0.4150	0.2075
2-level sieve	$(\gamma_1, \gamma_2) = (x, 1)$	$x^6 - 4x^4 + 4$	x^{3n}	x^{2n}	0.3836	0.2557
3-level sieve	$(\gamma_1, \gamma_2, \gamma_3) = (x^2, x, 1)$	$x^{10} - 4x^6 + 4$	x^{4n}	x^{3n}	0.37780	0.2833
4-level sieve	$(\gamma_1, \ldots, \gamma_4) = (x^3, \ldots, 1)$	$x^{14} - 4x^8 + 4$	x^{5n}	x^{4n}	0.37783	0.3023
5-level sieve	$(\gamma_1, \ldots, \gamma_5) = (x^4, \ldots, 1)$	$x^{18} - 4x^{10} + 4$	x^{6n}	x^{5n}	0.3797	0.3164
...						

Table 9.1: An overview of the optimal parameter choices (minimizing the time complexity) and the associated complexities for $k = 1, 2, 3$, and how these results may extend to higher-level sieving for $k = 4, 5, \ldots$. The bottom part of the table is based on Conjecture 9.5.

Conjecture 9.5. *For $k \geqslant 4$, a "sensible" parameter choice $(\gamma_1, \ldots, \gamma_k)$ for the k-level sieve is $(\gamma_1, \ldots, \gamma_k) = (x^{k-1}, \ldots, x, 1)$ where $x \in [1, 2^{1/(2k-2)}]$ is a root of*

$$p_k(x) = x^{4k-2} - 4x^{2k} + 4. \tag{9.21}$$

With this choice, the time and space complexities are $x^{(k+1)n+o(n)}$ and $x^{kn+o(n)}$ respectively, i.e., the time and space complexities are $2^{c_{\text{time}}n+o(n)}$ and $2^{c_{\text{space}}n+o(n)}$ with

$$c_{\text{time}} = (k+1)\log_2(x), \qquad c_{\text{space}} = k\log_2(x). \tag{9.22}$$

Note that these polynomials indeed have a root in the given interval by Bolzano's theorem, as $p_k(1) = 1$ and $p_k(2^{1/(2k-2)}) = 4(1 - 2^{1/(k-1)}) < 0$ for all k. Assuming this conjecture holds, and the given parameters indeed lead to the given complexities, then for 4- and 5-level sieving we would get the parameters and complexities described in Table 9.1.

Note that the use of the word "sensible" above (rather than "optimal") comes from the fact that, similar to 2- and 3-level sieving, one can always obtain a time/memory trade-off by taking $(\gamma_1, \ldots, \gamma_k) = (\alpha^{k-1}, \ldots, 1)$ with $x < \alpha < 2^{1/(2k-2)}$. As the curves in Figure 9.3 are decreasing, for 2- and 3-level sieving the endpoint of the curve is also optimal, minimizing the time complexity. For 4-level sieving however, note that the "sensible" parameter choice leads to a time complexity which is slightly worse than for 3-level sieving (with an even worse space complexity). We therefore believe that it is possible to choose a parameter $\alpha \in [x, \sqrt[6]{2}]$ where x is a root of $p_4(x)$, such that the time complexity is less than for 3-level sieving and less than given by the sensible choice above.

Asymptotics. Let us further consider what happens in the limiting case of large k in Conjecture 9.5. Letting $y = x^k$, we see that the polynomial $p_k(y)$ is of the form $p_k(y) = y^{4-o(1)} - 4y^2 + 4$ where $o(1) \to 0$ as $k \to \infty$, while the complexity exponents satisfy $c_{\text{space}} = y$ and $c_{\text{time}} = y(1 + o(1))$. For large k this means that y will start looking more and more like a root of the following polynomial:

$$p_\infty(y) = y^4 - 4y^2 + 4 = (y^2 - 2)^2. \tag{9.23}$$

This polynomial clearly has a single positive real solution at $y = \sqrt{2}$. For large k, this means that the "sensible" time and space exponents are conjectured to converge to $c_{\text{time}} = c_{\text{space}} = \frac{1}{2}$ with $(\gamma_1, \ldots, \gamma_k) = (x^{k-1}, \ldots, x, 1)$ where $x \approx 2^{1/(2k-2)}$.

Intuitively, note that in the limiting case of large k with a fixed ratio γ_{i+1}/γ_i, we will use parameters $\gamma_k \approx 1$ and $\gamma_1 \approx y = \sqrt{2}$, and the radii get slightly smaller and smaller in every of the k levels. As $\gamma_1 \approx \sqrt{2}$, we first partition the spherical shell in regions, by drawing balls of radius $\sqrt{2} - o(1)$ around certain outer centers C_1. Note that a ball of radius $\sqrt{2}$ would cover exactly half of the spherical shell, and so using a parameter $\gamma_1 = \sqrt{2} - o(1)$ means that the number of centers C_1 needed will be slightly subexponential for large k. Then, with each new layer, we reduce the radius of the balls by a constant factor $\gamma_{i+1}/\gamma_i = x$. For large k, this factor x is very close to 1, and we will only need subexponentially many centers in each layer for large k. The number of centers used in each layer will thus be subexponential for large k, but as the total number of regions is $n_1 \cdots n_k$ and k also increases, we are eventually apparently left with a total list of size $2^{d/2+o(d)}$, assuming the conjecture is true. Understanding where the exponent $d/2 + o(d)$ comes from is left as an open problem.

As the time complexity will only be slightly larger than the space complexity, as it is of the order $n_1 \cdots n_k(n_1 + \cdots + n_k)$, the time complexity for large k will then also be $2^{d/2+o(d)}$. Interestingly, assuming the conjecture is true, for large k this implies that with our leveled data structure, we can essentially find a vector at distance at most $\gamma_k \cdot R < R$ from a target vector $\boldsymbol{v}$ in subexponential time: the cost of a single traversal of the entire tree of centers costs $n_1 + \cdots + n_k$, and as the size of each list n_i is subexponential, the total cost for one search is $2^{o(d)}$.

Relation with Becker–Gama–Joux's overlattice sieve. Although as argued above the "sensible" parameter choice of Conjecture 9.5 does not quite make sense for $k \geqslant 4$, as the time and memory both increase, if we forget about optimality for now we can simply plot these (conjectured) points in the space/time trade-off grid as illustrated in Figure 9.4. Besides the conjectured high-level sieving complexities, this plot also includes the 1-, 2-, and 3-level points minimizing the time complexity, and the overlattice sieving trade-off curve [BGJ14, Figure 3]. Note that Becker–Gama–Joux's time/memory trade-off was obtained with a different algorithm which seems completely unrelated to leveled sieving.

At first sight, it appears that all (conjectured) k-level sieve points lie exactly on the overlattice trade-off curve. The following theorem shows that indeed, these points all lie on the given curve.

Theorem 9.6. *Let $x \in [1, 2^{1/(2k-2)}]$ be a root of $p_k(x) = x^{4k-2} - 4x^{2k} + 4$ and let $c_{\text{time}} = (k+1)\log_2(x)$ and $c_{\text{space}} = k\log_2(x)$. Then this point lies on the overlattice trade-off curve [BGJ14, Figure 3], and corresponds to choosing $\alpha = \frac{2}{x^k}\sqrt{x^{2k}-1}$ and $\beta = x^k$ in Becker–Gama–Joux's algorithm.*

Proof. The time complexity of Becker–Gama–Joux's algorithm (which we will not cover here) is given by $(\beta^2/\alpha)^d = 2^{d\log_2(\beta^2/\alpha)}$ and the space complexity is $\beta^d = 2^{d\log_2\beta}$, where the parameters α and β must satisfy the following relation for large d:

$$\beta\sqrt{1 - \frac{\alpha^2}{4}} \geqslant 1 + o(1). \tag{9.24}$$

In other words, the time and space exponents of their algorithm are $c_{\text{time}} = \log_2(\beta^2/\alpha)$ and $c_{\text{space}} = \log_2\beta$ respectively. The optimal trade-off curve [BGJ14, Figure 3] corresponds to replacing the inequality above by an equality and taking the limit of $d \to \infty$,

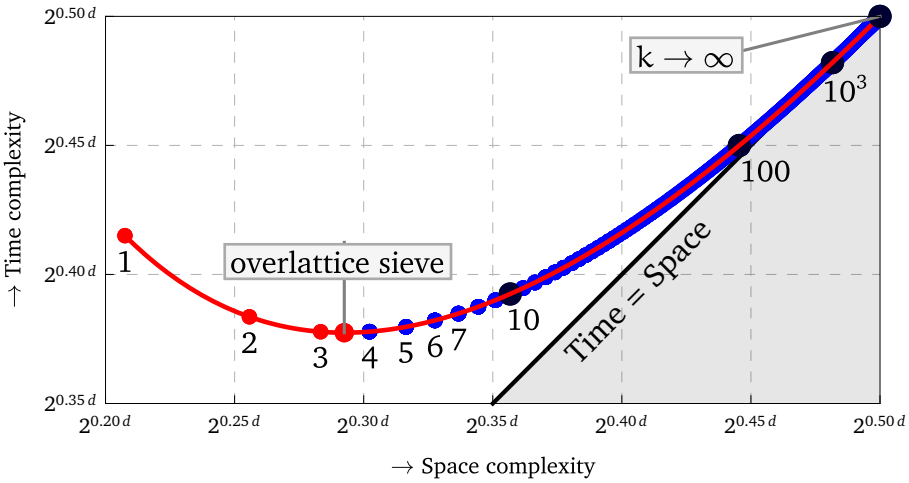


Figure 9.4: The optimal k-level sieve points for $k = 1, 2, 3$ (small red points), the optimal complexity of the overlattice sieve (big red point), the trade-off of the overlattice sieve [BGJ14] by varying $\alpha \in [1, \sqrt{2}]$ (red curve), and the conjectured "sensible" k-level sieve complexities of Conjecture 9.5 for $k \geqslant 4$ (blue). The left starting point corresponds to $k = 1$ in k-level sieving, which in the Becker–Gama–Joux curve corresponds to $\alpha = 1$ and $\beta = \sqrt{4/3}$. The limiting case $k \to \infty$ in leveled sieving is conjectured to correspond to the top right corner of the graph, which in the Becker–Gama–Joux curve corresponds to $\alpha = \beta = \sqrt{2}$. Note that the red curve also overlaps with both trade-off curves of Figure 9.3.

so that the $o(1)$-term disappears. Isolating α from (9.24) (with the inequality replaced by an equality) leads to $\alpha = \frac{2}{\beta}\sqrt{\beta^2 - 1}$. Note that substituting $\beta = x^k$ indeed leads to the given value for α. To prove that the k-level points lie on this curve, what remains is to prove that:

$$(c_{\text{time}}(\text{k-level})) \qquad (k+1)\log_2(x) \stackrel{?}{=} \log_2(\beta^2/\alpha), \qquad (c_{\text{time}}(\text{overlattices})) \tag{9.25}$$

$$(c_{\text{space}}(\text{k-level})) \qquad k\log_2(x) \stackrel{?}{=} \log_2(\beta). \qquad (c_{\text{space}}(\text{overlattices})) \tag{9.26}$$

The second equality follows immediately from the choice $\beta = x^k$. For the first inequality, we substitute the given expressions for α and β in the right hand side to obtain

$$\log_2(x^{k+1}) \stackrel{?}{=} \log_2\left(\frac{x^{3k}}{2\sqrt{x^{2k}-1}}\right). \tag{9.27}$$

Removing the logarithms, and taking squares, this is equivalent to

$$x^{2k+2} \stackrel{?}{=} \frac{x^{6k}}{4(x^{2k}-1)}. \tag{9.28}$$

Multiplying by $4(x^{2k}-1)/x^{2k+2}$ and bringing all terms to one side, this is equivalent to $p_k(x) \stackrel{?}{=} 0$. As x is a root of this polynomial, this is clearly true. □

Limitations of leveled sieving. To come back to the initial question of whether 4-level sieving beats the overlattice algorithm, and what the complexity would be, as explained before one can obtain a suitable trade-off between the time and memory by taking

$(\gamma_1, \ldots, \gamma_k) = (\alpha^{k-1}, \ldots, 1)$ and choosing $\alpha \geqslant x$ where x is a root of $p_k(x)$. In Figure 9.3 we saw how this means we can interpolate between e.g. the complexities of 1-, 2-, and 3-level sieving. These trade-offs also lie exactly on the Becker–Gama–Joux curve, and so we naturally conjecture that for the 4-level sieve we can obtain an arbitrary point on the Becker–Gama–Joux trade-off curve between the optimized point for 3-level sieving and the extreme point for 4-level sieving, and in particular we can achieve the same optimized time complexity as Becker–Gama–Joux.

Conjecture 9.7. *There exists a parameter choice* $(\gamma_1, \gamma_2, \gamma_3, \gamma_4) = (\alpha^3, \alpha^2, \alpha, 1)$ *with* $\alpha \in (x, 2^{1/6})$*, where* $x \approx 1.05377$ *is a root of* $p_4(x)$*, such that the* 4*-level sieve achieves the same optimized complexities as Becker–Gama–Joux, and this is the best time complexity that can be achieved with* 4*-level sieving. In other words, minimizing the time complexity in* 4*-level sieving leads to a time complexity* $2^{c_{\text{time}} d + o(d)}$ *and a space complexity* $2^{c_{\text{space}} d + o(d)}$ *with*

$$c_{\text{time}} = \tfrac{1}{2}\log_2(\tfrac{27}{16}) \approx 0.3774, \qquad c_{\text{space}} = \tfrac{1}{2}\log_2(\tfrac{3}{2}) \approx 0.2925. \tag{9.29}$$

The increasing curve and the increased time and space complexities of Conjecture 9.5 also suggest that this is the best that can be achieved with leveled sieving overall.

Conjecture 9.8. *With higher-level sieving it is not possible to achieve a better asymptotic time complexity than the conjectured* $(c_{\text{space}}, c_{\text{time}}) \approx (0.2925, 0.3774)$ *of the* 4*-level sieve.*

To conclude, the analysis in this chapter raises many new interesting questions, such as: Why are the complexities of leveled sieving and the overlattice sieve related, even though the algorithms seem unrelated? Why does the (conjectured) limit of $k \to \infty$ correspond to time and space complexities of $2^{d/2+o(d)}$? How should one choose α in Conjecture 9.7 to obtain the optimal time complexity? How does the multi-level sieving approach discussed here relate to the multi-level approach in data-dependent locality-sensitive hashing in [AINR14, Ngu14, Raz14]? Fortunately the conjectured pattern of leveled sieving also leads to at least one (conjectured) answer to our initial question: it is unlikely that higher-level sieving will lead to a better asymptotic time complexity than the overlattice sieve of Becker–Gama–Joux, and so to obtain a better complexity it is probably necessary to use different techniques.

CHAPTER 10

Hyperplane locality-sensitive hashing

10.1 — Overview

Context. In the previous chapter we saw that leveled sieving with multiple levels of center vectors will probably not lead to an improved time complexity compared to Becker–Gama–Joux's overlattice sieve [BGJ14]. We also saw that two-level sieving could be viewed as using a (data-dependent) space partitioning method for finding nearby vectors faster, similar in spirit to nearest-neighbor search techniques like locality-sensitive hashing (LSH). Investigating whether other (existing) methods from nearest-neighbor searching lead to speed-ups for sieving is a natural follow-up question, and this chapter as well as the next will address what are arguably the most practical method (this chapter) and the asymptotically best method (Chapter 11) for solving the nearest neighbor search problem in the context of sieving.

Hyperplane locality-sensitive hashing. While the idea of leveled sieving of partitioning the space and only searching within these regions for nearby vectors was very ingenious indeed, as mentioned before this idea was not quite new. Many methods were already known for partitioning the space, in such a way that nearby vectors are more likely to end up in the same region than distant vectors. In this chapter we will investigate the celebrated hyperplane locality-sensitive hash method of Charikar [Cha02], where the regions are defined as intersections of random half-spaces, and we will show that with this method we can heuristically solve SVP with sieving in time and space both bounded by $2^{0.3366d+o(d)}$. Tuning the parameters differently, we further obtain a continuous trade-off between the space and time complexities as illustrated in Figure 10.1. This result applies to both the Nguyễn–Vidick sieve and the GaussSieve, while for the Nguyễn–Vidick sieve we can further use a trick first applied in [BGJ15] to turn the trade-off into a speed-up, leading to a time complexity of $2^{0.3366d+o(d)}$ and a space complexity of $2^{0.2075d+o(d)}$. The complexities in this point are indicated by the leftmost blue point in Figure 10.1.

The same idea of turning the trade-off into a speed-up unfortunately does not seem to apply to the more practical GaussSieve. To overcome the increase in the memory complexity we show that with probing we can significantly limit the increase in the memory, at only a small loss in the time complexity. Practical experiments with the GaussSieve with hyperplane LSH (the HashSieve) validate our heuristic analyses, and show that (i) already in low dimensions, this algorithm outperforms the most practical sieving algorithm to date, the GaussSieve algorithm [MV10b]; and (ii) as expected, the increase in

*This chapter is based on results from [Laa15d, MLB15].

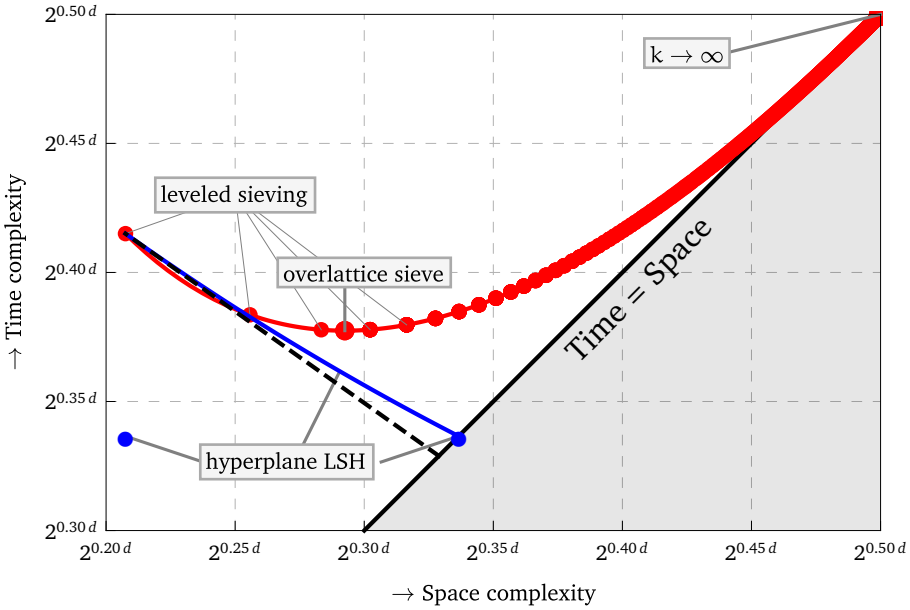


Figure 10.1: The heuristic space-time trade-off of overlattice sieving and leveled sieving (the red points and curve, cf. Chapter 9), and the heuristic trade-off between the space and time complexities obtained with hyperplane LSH (the blue curve). The dashed, black line shows the estimate for the space-time trade-off of our algorithm obtained by assuming that all reduced vectors are orthogonal (cf. Proposition 10.3). The leftmost blue point follows from Theorem 10.10 and only applies to the Nguyên–Vidick sieve, and not to the GaussSieve.

the space complexity is significantly smaller than one might guess from only looking at the leading exponent of the asymptotic space complexity of the GaussSieve-based HashSieve.

Outline. The remainder of this chapter is organized as follows. In Section 10.2 we describe the technique of locality-sensitive hashing, and in Section 10.3 we describe the specific instantiation of this general framework with Charikar's hyperplane locality-sensitive hash family [Cha02]. Section 10.4 describes how to apply this technique to the Nguyễn–Vidick sieve algorithm to obtain an exponential speed-up for solving SVP. Section 10.5 describes a practical alternative by applying the same techniques to the GaussSieve, leading to the HashSieve algorithm, and describes the technique of probing. In the same section we finally describe experiments performed with the GaussSieve and the HashSieve, showing that the HashSieve is already faster than the GaussSieve in moderate dimensions (i.e. the cross-over point between the GaussSieve and the HashSieve seems to lie around dimensions $30 - 40$).

10.2 — The locality-sensitive hashing (LSH) framework

10.2.1 – Locality-sensitive hash families. The near(est) neighbor problem is the following [IM98]: Given a list of d-dimensional vectors of cardinality n, e.g., given a list $L = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n\} \subset \mathbb{R}^d$, preprocess L in such a way that given a target vector $\boldsymbol{v} \notin L$, we can efficiently find an element $\boldsymbol{w} \in L$ close(st) to $\boldsymbol{v}$. While for low dimensions d there exist ways to answer these queries in time sub-linear or even logarithmic in the list size n,

for high dimensions it generally seems hard to do better than with a naive brute-force list search of time $O(n)$. This inability to efficiently store and query lists of high-dimensional data is sometimes referred to as the "curse of dimensionality" [IM98].

Fortunately, if we know that the list L has a certain structure, or if there is a significant gap between what is meant by "nearby" and "far away," then there are ways to preprocess L such that queries can be answered in time sub-linear in n. One of the most well-known methods for this is locality-sensitive hashing (LSH), which relies on the use of locality-sensitive hash functions [IM98]. These are functions h which map d-dimensional vectors $\boldsymbol{v}$ to low-dimensional *sketches* $h(\boldsymbol{v})$, such that vectors which are nearby in $\mathbb{R}^d$ have a high probability of having the same sketch and vectors which are far apart have a low probability of having the same image under h. Formalizing this property leads to the following definition of a locality-sensitive hash family $\mathcal{H}$. Here D is a similarity measure[1] on $\mathbb{R}^d$, and U is commonly a finite subset of $\mathbb{N}$.

Definition 10.1. *[IM98] A family $\mathcal{H} = \{h : \mathbb{R}^d \to U\}$ is called (r_1, r_2, p_1, p_2)-sensitive for similarity measure D if for any $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^d$:*

- *if $D(\boldsymbol{v}, \boldsymbol{w}) < r_1$ then $\mathbb{P}_{h \in \mathcal{H}}[h(\boldsymbol{v}) = h(\boldsymbol{w})] \geqslant p_1$;*
- *if $D(\boldsymbol{v}, \boldsymbol{w}) > r_2$ then $\mathbb{P}_{h \in \mathcal{H}}[h(\boldsymbol{v}) = h(\boldsymbol{w})] \leqslant p_2$.*

Note that if there exists an LSH family $\mathcal{H}$ which is (r_1, r_2, p_1, p_2)-sensitive with $p_1 \gg p_2$, then (without actually computing pairwise distances) we can use $\mathcal{H}$ to distinguish between vectors which are at most r_1 away from $\boldsymbol{v}$, and vectors which are at least r_2 away from $\boldsymbol{v}$ with non-negligible probability.

10.2.2 – Amplification. In general it is not known whether efficiently computable (r_1, r_2, p_1, p_2)-sensitive hash families even exist for the ideal setting of $r_1 \approx r_2$ and $p_1 \approx 1$ and $p_2 \approx 0$. Instead, one commonly first constructs an (r_1, r_2, p_1, p_2)-sensitive hash family $\mathcal{H}$ with $p_1 \approx p_2$, and then uses several AND- and OR-compositions to turn it into an (r_1, r_2, p_1', p_2')-sensitive hash family $\mathcal{H}'$ with $p_2' < p_2 < p_1 < p_1'$, thereby amplifying the gap between p_1 and p_2.

AND-composition. Given an (r_1, r_2, p_1, p_2)-sensitive hash family $\mathcal{H}$, we can construct an (r_1, r_2, p_1^k, p_2^k)-sensitive hash family $\mathcal{H}'$ by taking a bijective function $\alpha : U^k \to U'$ and k functions $h_1, \ldots, h_k \in \mathcal{H}$ and defining $h \in \mathcal{H}'$ as $h(\boldsymbol{v}) = \alpha(h_1(\boldsymbol{v}), \ldots, h_k(\boldsymbol{v}))$. This increases the relative gap between p_1 and p_2 but decreases their absolute values.

OR-composition. Given an (r_1, r_2, p_1, p_2)-sensitive hash family $\mathcal{H}$, we can construct an $(r_1, r_2, 1-(1-p_1)^t, 1-(1-p_2)^t)$-sensitive hash family $\mathcal{H}'$ by taking $h_1, \ldots, h_t \in \mathcal{H}$, and defining $h \in \mathcal{H}'$ by the relation $h(\boldsymbol{v}) = h(\boldsymbol{w})$ iff $h_i(\boldsymbol{v}) = h_i(\boldsymbol{w})$ for some $i \in \{1, \ldots, t\}$. This compensates the decrease of the absolute values of the probabilities.

Combining a k-wise AND-composition with a t-wise OR-composition, we can turn an (r_1, r_2, p_1, p_2)-sensitive hash family $\mathcal{H}$ into an (r_1, r_2, p_1^*, p_2^*)-sensitive hash family $\mathcal{H}'$ with $p^* = 1-(1-p^k)^t$ for $p = p_1, p_2$. Note that for $p_1 > p_2$ we can always find values k and t such that $p_1^* \approx 1$ and $p_2^* \approx 0$. For ease of notation, we will write $p^* = 1-(1-p^k)^t$ for arbitrary p, k, t.

[1] A similarity measure D may informally be thought of as a "slightly relaxed" metric, which may not satisfy all properties associated to metrics; see e.g. [IM98] for details.

10.2.3 – Finding nearest neighbors. To use these hash families to find nearest neighbors, we can use the following method first described in [IM98]. First, choose $t \cdot k$ random hash functions $h_{i,j} \in \mathcal{H}$, and use the AND-composition to combine k of them at a time to build t different hash functions $h_1, \dots, h_t$. Then, given the list L, build t different hash tables $T_1, \dots, T_t$, where for each hash table T_i we insert $\boldsymbol{w}$ into the bucket labeled $h_i(\boldsymbol{w})$. Finally, given the target vector $\boldsymbol{v}$, compute its t images $h_i(\boldsymbol{v})$, gather all the candidate vectors that collide with $\boldsymbol{v}$ in at least one of these hash tables (an OR-composition), and search this list of candidates for the nearest neighbor.

Clearly, the quality of this algorithm for finding nearest neighbors depends on the quality of the underlying hash family and on the parameters k and t. Larger k and t amplify the gap between the probabilities of finding nearby and faraway vectors as candidates, but this comes at the cost of having to compute many hashes (both during the preprocessing phase and in the querying phase) and having to store many hash tables, each containing all vectors from L. The following lemma shows how to balance k and t such that the overall query time complexity of finding near(est) neighbors is minimized.

Lemma 10.2. *[IM98] Suppose there exists a (r_1, r_2, p_1, p_2)-sensitive hash family $\mathcal{H}$. Let*

$$\rho = \frac{\log p_1}{\log p_2}, \qquad k = \frac{\log n}{\log(1/p_2)}, \qquad t = O(n^\rho). \tag{10.1}$$

Then, for any $\boldsymbol{v} \in \mathbb{R}^d$, with high probability we can find an element $\boldsymbol{w}^* \in L$ with $D(\boldsymbol{v}, \boldsymbol{w}^*) \leqslant r_2$ or correctly conclude that no element $\boldsymbol{w}^* \in L$ with $D(\boldsymbol{v}, \boldsymbol{w}^*) \leqslant r_1$ exists, with the following costs:

1. *Time for preprocessing the list:* $O(n^{1+\rho} \log_{1/p_2} n)$.
2. *Space complexity of the preprocessed data:* $O(n^{1+\rho})$.
3. *Time for answering a query $\boldsymbol{v}$:* $O(n^\rho)$.
   a) *Hash evaluations of the query vector $\boldsymbol{v}$:* $t = O(n^\rho)$.
   b) *Candidates to compare to the query vector $\boldsymbol{v}$:* $O(n^\rho)$.

Remark 10.1. Although Lemma 10.2 only shows how to choose k and t to minimize the time complexity, we can generally tune k and t to use slightly more time and less space. In a sense this algorithm can be seen as a generalization of the naive brute-force search method, as $k = 0$ and $t = 1$ corresponds to checking the whole list in linear time with linear space. Note that the main costs of the algorithm are determined by the value of ρ, which is therefore often considered the central parameter of interest in LSH literature. The goal is to design $\mathcal{H}$ so that ρ is as small as possible.

10.3 — Hyperplane locality-sensitive hashing

Let us now consider an actual hash family for the similarity measure D that we are interested in. As argued in the next section, what seems a more natural choice for D than the Euclidean distance is the *angular distance* or *cosine similarity*, defined on $\mathbb{R}^d$ as

$$D(\boldsymbol{v}, \boldsymbol{w}) = \theta(\boldsymbol{v}, \boldsymbol{w}) = \arccos\left(\frac{\boldsymbol{v}^T \boldsymbol{w}}{\|\boldsymbol{v}\| \cdot \|\boldsymbol{w}\|}\right). \tag{10.2}$$

In other words, the "distance" between two vectors is given by their common angle. With this similarity measure, two vectors are considered nearby if their common angle is small,

and far apart if their angle is large. In a sense, this is similar to the Euclidean norm: if two vectors have similar Euclidean norms, then their distance is large if and only if their angular distance is large. For this similarity measure D, the following hash family $\mathcal{H}$ was described in [Cha02]:

$$\mathcal{H} = \{h_{\mathbf{a}} : \mathbf{a} \in \mathbb{R}^d, \|\mathbf{a}\| = 1\}, \qquad h_{\mathbf{a}}(\mathbf{v}) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \mathbf{a}^T\mathbf{v} \geqslant 0; \\ 0 & \text{if } \mathbf{a}^T\mathbf{v} < 0. \end{cases} \tag{10.3}$$

Intuitively, the vector $\mathbf{a}$ defines a hyperplane (for which $\mathbf{a}$ is a normal vector), and $h_{\mathbf{a}}$ maps the two half-spaces separated by this hyperplane to different bits.

To see why this is potentially a suitable locality-sensitive hash family for the angular distance, consider two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^d$. These two vectors lie on a two-dimensional plane passing through the origin, and with probability 1 a random 8hash vector $\mathbf{a}$ does not lie on this plane (for $d > 2$). This means that the hyperplane defined by $\mathbf{a}$ intersects this plane in some line ℓ. Since $\mathbf{a}$ is taken uniformly at random from the unit sphere, the line ℓ has a uniformly random 'direction' in the plane, and maps $\mathbf{v}$ and $\mathbf{w}$ to different hash values iff ℓ separates $\mathbf{v}$ and $\mathbf{w}$ in the plane. Therefore the probability that $h(\mathbf{v}) \neq h(\mathbf{w})$ is directly proportional to their common angle $\theta(\mathbf{v}, \mathbf{w})$ as follows [Cha02]:

$$\mathbb{P}_{h_{\mathbf{a}} \in \mathcal{H}}\big[h_{\mathbf{a}}(\mathbf{v}) = h_{\mathbf{a}}(\mathbf{w})\big] = 1 - \frac{\theta(\mathbf{v}, \mathbf{w})}{\pi}. \tag{10.4}$$

For instance, if $\theta(\mathbf{v}, \mathbf{w}) = 0$ then $\mathbb{P}_{h_{\mathbf{a}} \in \mathcal{H}}[h_{\mathbf{a}}(\mathbf{v}) = h_{\mathbf{a}}(\mathbf{w})] = 1$, and if $\theta(\mathbf{v}, \mathbf{w}) = \pi$ then $\mathbb{P}_{h_{\mathbf{a}} \in \mathcal{H}}[h_{\mathbf{a}}(\mathbf{v}) = h_{\mathbf{a}}(\mathbf{w})] = 0$. For any two angles $\theta_1 < \theta_2$, the family $\mathcal{H}$ is $(\theta_1, \theta_2, 1 - \frac{\theta_1}{\pi}, 1 - \frac{\theta_2}{\pi})$-sensitive. In particular, Charikar's hyperplane hash family is $(\frac{\pi}{3}, \frac{\pi}{2}, \frac{2}{3}, \frac{1}{2})$-sensitive.

To illustrate LSH and in particular the hyperplane LSH method described above, Figure 10.2 shows how hyperplane hashing might work in a two-dimensional setting ($d = 2$) where we have a list $L = \{\mathbf{w}_1, \ldots, \mathbf{w}_{10}\}$ and a query vector $\mathbf{v}$, and we used $k = 2$ hyperplanes in each of t hash tables to find nearby vectors. Preprocessing would consist of computing and storing each of the list vectors in their corresponding hash buckets, which involves k inner product computations for each vector for each hash table. Answering a query can be done by computing a target vector's hash buckets in each hash table and searching the vectors in these hash buckets for a nearest neighbor.

10.4 — The Nguyễn–Vidick sieve with hyperplane LSH

Let us now describe how locality-sensitive hashing can be used to speed up sieving, and in particular how we can speed up the NV-sieve of Nguyễn and Vidick [NV08] using hyperplane LSH. The same ideas can also be applied to the GaussSieve [MV10b], as outlined in Section 10.5.

10.4.1 – The Nguyễn–Vidick sieve algorithm. First, recall that the sieving algorithm of Nguyễn and Vidick [NV08] starts with a long list L of reasonably long, randomly sampled lattice vectors $\mathbf{v}$ sampled from a discrete Gaussian over the lattice, using e.g. Klein's algorithm [Kle00], and then repeatedly applies a sieve to it to split each list L into a list C of centers and a new list L′ of vectors whose norms are at least a geometric factor $\gamma < 1$ smaller than the maximum norm of the vectors in L. After repeatedly applying this sieve,

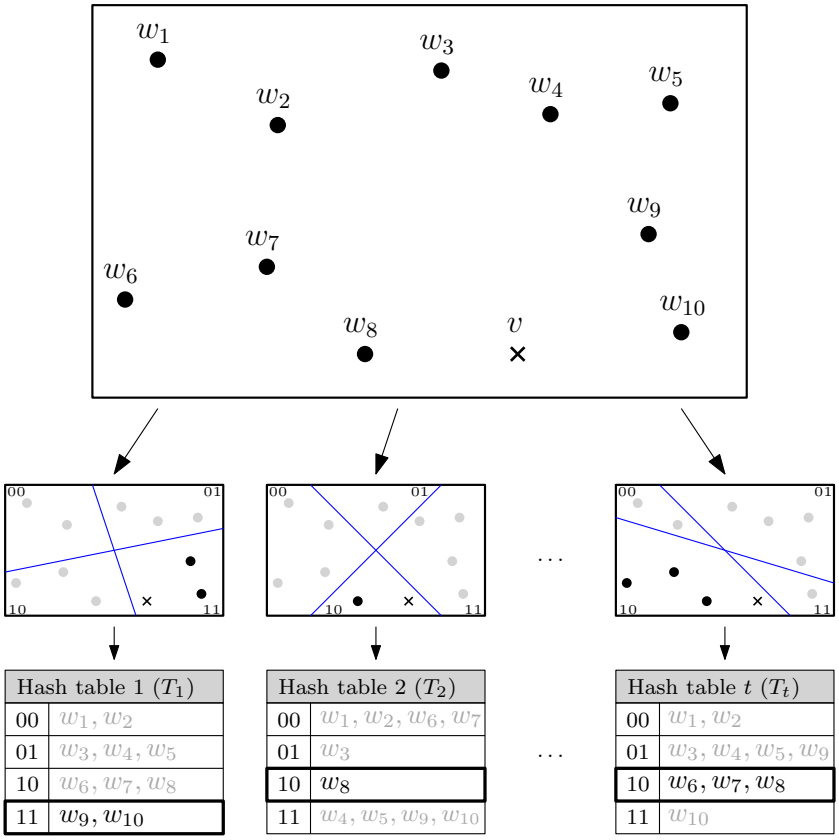


Figure 10.2: An example of hyperplane LSH, using $k = 2$ hyperplanes and $2^k = 4$ buckets in each hash table. Given 10 list vectors $L = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{10}\}$ and a target vector $\boldsymbol{v}$, for each of the t hash tables we first compute $\boldsymbol{v}$'s hash value (i.e. compute the region in which it lies), look up vectors which have the same hash value, and compare $\boldsymbol{v}$ to those vectors. Based on these three hash tables, we will find $C = \{\boldsymbol{w}_6, \boldsymbol{w}_7, \boldsymbol{w}_8, \boldsymbol{w}_9, \boldsymbol{w}_{10}\}$ as the set of candidate near neighbors for $\boldsymbol{v}$.

discarding L and C and continuing with $L \leftarrow L'$ we eventually hope to be left with a short list of very short lattice vectors, which contains the shortest vector.

A slightly modified and simplified version of the sieve that maps L onto two sets L' and C is described in Algorithm 10.2. Here we have replaced the on-the-fly generation of the set of centers in the original algorithm by a predetermined random selection of list points to be used as centers. Note that Nguyễn and Vidick's analysis is essentially based on this modified algorithm rather than the on-the-fly generation described in [NV08, Algorithm 5] and Chapters 9 and 10, and for reducing the space complexity later on it is essential that we select the set of centers in advance.

In Lines 7–10 of Algorithm 10.1, the Nguyễn–Vidick sieve essentially solves the following search problem through a brute-force linear search:

$$\text{Find an element } \boldsymbol{w} \in C \text{ such that } \|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma R. \tag{10.5}$$

To obtain the estimate $2^{0.415d+o(d)}$ for the time complexity and $2^{0.208d+o(d)}$ for the space

Algorithm 10.1 Nguyễn and Vidick's lattice sieve (without hyperplane LSH)

Require: An input list L of $(4/3)^{d/2+o(d)}$ vectors of norm at most R
Ensure: The output list L' has $(4/3)^{d/2+o(d)}$ vectors of norm at most $\gamma \cdot R$
1: Initialize an empty list L'
2: Sample $C \subset L \cap \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| \geqslant \gamma \cdot R\}$ of size $\mathrm{poly}(d) \cdot (4/3)^{d/2}$
3: **for each** $\boldsymbol{v} \in L \setminus C$ **do**
4: **if** $\|\boldsymbol{v}\| \leqslant \gamma R$ **then**
5: Add $\boldsymbol{v}$ to the list L'
6: **else**
7: **for each** $\boldsymbol{w} \in C$ **do**
8: **if** $\|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma \cdot R$ **then**
9: Add $\boldsymbol{v} - \boldsymbol{w}$ to the list L'
10: Continue the loop over "$\boldsymbol{v} \in L \setminus C$"

complexity, recall that Nguyễn and Vidick further let γ approach 1 in their analysis. This means that all vectors with a length significantly shorter than $\gamma \cdot R \approx R$ are automatically added to L' in Line 5, and the bottleneck of the time complexity comes from those vectors $\boldsymbol{v}$ with $\gamma R < \|\boldsymbol{v}\| \leqslant R$, i.e., the vectors $\boldsymbol{v}$ lying in a thin spherical shell of thickness $(1-\gamma)R$ around the origin. For those vectors $\boldsymbol{v}$ we have $\gamma R \approx \|\boldsymbol{v}\| \approx R$, and for vectors $\boldsymbol{w} \in C_{m+1}$ we also know that $\gamma R \approx \|\boldsymbol{w}\| \approx R$. This implies that the reduction method described in (10.5) for $\gamma \to 1$ is essentially equivalent to the following angular reduction step:

$$\text{Find an element } \boldsymbol{w} \in C \text{ such that } \theta(\boldsymbol{v}, \boldsymbol{w}) \leqslant 60^\circ. \tag{10.6}$$

In the limiting case of $\gamma \to 1$, the problems (10.5) and (10.6) are essentially equivalent.

10.4.2 – The (NV-)HashSieve algorithm. Considering the angular notion of reduction of (10.6), we can clearly see the connection with nearest neighbor searching for the angular distance or cosine similarity, and how we can fit in angular or hyperplane LSH. Replacing the brute-force list search over $\boldsymbol{w} \in C$ in the original algorithm with the technique of hyperplane locality-sensitive hashing, we obtain Algorithm 10.2. Blue lines in Algorithm 10.2 indicate modifications to the original algorithm. Note that the setup costs of locality-sensitive hashing (building the hash tables) are only paid once, rather than once for each search.

10.4.3 – Relation with leveled sieving. Overall, the crucial modification we introduce is that by using hash tables and looking up vectors to reduce the target vector with in these hash tables, we make the search space smaller; instead of comparing a new vector to *all* vectors in C, we only compare a vector to a much smaller subset of candidates $\mathcal{C} \subset C$, which mostly contains good, nearby candidates for reduction, and does not contain many of the distant vectors in C to $\boldsymbol{v}$. This is very similar to leveled sieving (cf. Chapter 9), where the search space of candidate nearby vectors was reduced by partitioning the space into regions, and for each vector storing in which region it lies. In those algorithms, two nearby vectors in adjacent regions are not considered for reductions, which means one needs more vectors to saturate the space (a higher space complexity) but less time to search the list of candidates for nearby vectors (a lower time complexity).

Algorithm 10.2 Nguyễn and Vidick's lattice sieve (with hyperplane LSH)

Require: An input list L of $(4/3)^{d/2+o(d)}$ vectors of norm at most R
Ensure: The output list L' has $(4/3)^{d/2+o(d)}$ vectors of norm at most $\gamma \cdot R$

Initialize an empty list L_{m+1}
Sample $C \subset L \cap \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| \geqslant \gamma \cdot R\}$ of size $\mathrm{poly}(d) \cdot (4/3)^{d/2}$
Initialize k empty hash tables T_i and sample $k \cdot t$ random hash functions $h_{i,j} \in \mathcal{H}$
for each $\boldsymbol{w} \in C$ **do**
    Add $\boldsymbol{w}$ to the buckets $T_i[h_i(\boldsymbol{w})]$ in the t hash tables T_i
for each $\boldsymbol{v} \in L \setminus C$ **do**
    if $\|\boldsymbol{v}\| \leqslant \gamma R$ **then**
        Add $\boldsymbol{v}$ to the list L'
    else
        Obtain the set of candidates $\mathcal{C} = \bigcup_{i=1}^{t} T_i[h_i(\boldsymbol{v})]$
        for each $\boldsymbol{w} \in \mathcal{C}$ **do**
            if $\|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma \cdot R$ **then**
                Add $\boldsymbol{v} - \boldsymbol{w}$ to the list L'
                Continue the loop over "$\boldsymbol{v} \in L \setminus C$"

Two key differences between leveled sieving and hyperplane LSH are the way the partitions of $\mathbb{R}^d$ are chosen (using giant balls in leveled sieving, similar to the Euclidean LSH method of [AI06], versus using random hyperplanes here), and whereas LSH guarantees that nearby vectors are found with high (constant) probability using (exponentially) many hash tables, leveled sieving only uses one "hash table" and pays for this with a larger list size.

10.4.4 – High-dimensional intuition. To estimate the complexity of the hyperplane LSH-based lattice sieve, we will again make use of Assumption 8.1, which says that the angle $\Theta(\boldsymbol{v}, \boldsymbol{w})$ between list vectors $\boldsymbol{v}, \boldsymbol{w}$ follows the same distribution as the distribution of angles $\Theta(\boldsymbol{v}, \boldsymbol{w})$ obtained by drawing $\boldsymbol{v}, \boldsymbol{w} \in S^{d-1}$ at random from the unit sphere.

Note that under this assumption, in high dimensions angles close to 90° are much more likely to occur between pairs of vectors than smaller angles. So one might guess that for two vectors $\boldsymbol{v}, \boldsymbol{w} \in L$, with high probability their angle is very close to 90°. On the other hand, nearby vectors $\boldsymbol{w} \in L$ that can reduce our target vector $\boldsymbol{v}$ always have an angle less than 60° with $\boldsymbol{v}$, and by similar arguments we expect this angle to always be close to 60° and not much less than this. Under the extreme assumption that all angles between vectors $\boldsymbol{v}, \boldsymbol{w}$ that do not satisfy the condition $\|\boldsymbol{v} \pm \boldsymbol{w}\| \leqslant \gamma \cdot R$ are *exactly* 90° (and angles of nearby pairs of vectors are at most 60°), we obtain the following preliminary estimate for the costs of this algorithm. Note that this preliminary estimate does not make use of Assumption 8.1.

Proposition 10.3. *Assuming that non-reducing vectors are always pairwise orthogonal, the NV-sieve with hyperplane LSH with parameters* $k = 0.2075d + o(d)$ *and* $t = 2^{0.1214d+o(d)}$ *heuristically solves SVP in time and space* $2^{0.3289d+o(d)}$. *By varying the values* k *and* t, *we further obtain the trade-off between the space and time complexities indicated by the straight dashed line in Figure 10.1.*

Proof. If all 'random angles' are 90°, then we can simply let $\theta_1 = \frac{\pi}{3}$ and $\theta_2 = \frac{\pi}{2}$ and use the hash family described in Section 10.3 with $p_1 = \frac{2}{3}$ and $p_2 = \frac{1}{2}$. Applying Lemma 10.2, where $\rho = \frac{\log(1/p_1)}{\log(1/p_2)} = \log_2(\frac{3}{2}) \approx 0.585$, we can perform a single search in time $n^\rho = 2^{0.1214d+o(d)}$ using $t = 2^{0.1214d+o(d)}$ hash tables. Since we need to perform these searches $\tilde{O}(n)$ times, and we need to repeat the whole sieving procedure $\mathrm{poly}(d)$ times, the time complexity is of the order $\tilde{O}(n^{1+\rho}) = 2^{0.3289d+o(d)}$. The space complexity is dominated by having to store all n vectors in $t = \tilde{O}(n^\rho)$ hash tables, leading to a space complexity of $\tilde{O}(n^{1+\rho}) = 2^{0.3289d+o(d)}$ as well. □

10.4.5 – Solving SVP in time and space $2^{0.3366d+o(d)}$. Of course, in practice not all pairwise angles between 'reduced' vectors are actually 90°, and one should carefully analyze what is the real probability that a vector $\boldsymbol{w}$ whose angle with $\boldsymbol{v}$ is more than 60°, is found as a candidate due to a collision in one of the hash tables. The following theorem follows from this analysis and shows how to choose the parameters to optimize the asymptotic time complexity.

Theorem 10.4. *The Nguyễn–Vidick sieve with hyperplane LSH with parameters*

$$k = 0.2206d + o(d), \qquad t = 2^{0.1290d+o(d)}, \tag{10.7}$$

and $\gamma \to 1$ heuristically solves SVP in time and space $2^{0.3366d+o(d)}$. Tuning k and t differently, we further obtain the trade-off indicated by the solid blue line in Figure 10.1.

Note that the optimized values in Theorem 10.4 and Proposition 10.3, and the associated curves in Figure 10.1 are very similar. In other words, the simple estimate based on the intuition that in high dimensions "everything is orthogonal" is not far off. Also note that as there are $2^k \gg n$ hash buckets in each table, the space complexity would be slightly higher ($2^{0.3496d+o(d)}$) if hash tables are naively stored as arrays. To achieve the minimum asymptotic time and space complexities, one should only store non-empty buckets in memory as asymptotically most buckets are empty.

To prove the claims in Theorem 10.4, we will show how to choose a sequence of parameters $\{(k_d, t_d)\}_{d\in\mathbb{N}}$ such that for large d, the following holds:

1. The average probability that a reducing vector $\boldsymbol{w}$ collides with $\boldsymbol{v}$ in at least one of the t hash tables is at least constant in d:
$$p_1^* = \mathbb{P}_{h_{i,j}\in\mathcal{H}}[\boldsymbol{v}, \boldsymbol{w} \text{ collide} \mid \theta(\boldsymbol{v}, \boldsymbol{w}) \leqslant \tfrac{\pi}{3}] \geqslant 1 - \varepsilon. \qquad (0 < \varepsilon \neq \varepsilon(d)) \tag{10.8}$$
2. The average probability that a non-reducing vector $\boldsymbol{w}$ collides with $\boldsymbol{v}$ in at least one of the t hash tables is exponentially small:
$$p_2^* = \mathbb{P}_{h_{i,j}\in cH}[\boldsymbol{v}, \boldsymbol{w} \text{ collide} \mid \theta(\boldsymbol{v}, \boldsymbol{w}) > \tfrac{\pi}{3}] \leqslant n^{-0.3782+o(1)}. \tag{10.9}$$
3. The number of hash tables grows as $t = n^{0.6218+o(1)}$.

This would imply that for each search, the number of candidate vectors is of the order $n \cdot n^{-0.3782} = n^{0.6218}$ and the number of hash computations is also $t = n^{0.6218+o(1)}$. Overall, we heuristically expect to iterate searching the list $\tilde{O}(n)$ times, so after substituting $n = 2^{0.2075d+o(d)}$ this leads to the following asymptotic time and space complexities:

- Time (hashing): $\tilde{O}(n \cdot t) = 2^{0.3366d+o(d)}$.
- Time (searching): $\tilde{O}(n^2 \cdot p_2^*) = 2^{0.3366d+o(d)}$.
- Space: $\tilde{O}(n \cdot t) = 2^{0.3366d+o(d)}$.

In the following three subsections we will prove Equations (10.8) and (10.9).

10.4.6 – Nearby vectors collide with constant probability. We first prove that reducing vectors often collide in at least one of the hash tables, given that k is a suitable function of t. In the next subsection we then show how p_2^* scales as a function of k and t and how to choose k and t to minimize the overall time complexity. Finally we describe how to obtain the trade-off between the space and time complexities of Figure 10.1 by choosing k and t slightly differently.

Lemma 10.5. *Let $\varepsilon > 0$ and let $k = \log_{3/2}(t) - \log_{3/2}(\ln 1/\varepsilon)$. Then the probability that reducing vectors collide in at least one of the hash tables is at least $1 - \varepsilon$.*

Proof. The probability that a reducing vector $\boldsymbol{w}$ is a candidate vector, given the angle $\Theta = \Theta(\boldsymbol{v}, \boldsymbol{w}) \in (0, \frac{\pi}{3})$, is

$$p_1^* = \mathbb{E}_{\Theta \in (0,\frac{\pi}{3})}\left[p^*(\Theta)\right] = \mathbb{E}_{\Theta \in (0,\frac{\pi}{3})}\left[1 - \left(1 - \left(1 - \frac{\Theta}{\pi}\right)^k\right)^t\right], \tag{10.10}$$

where the angle Θ is a random variable with a certain distribution on $(0, \frac{\pi}{3})$. Since the argument on the right hand side is strictly decreasing in Θ, we can obtain a lower bound by substituting $\Theta = \frac{\pi}{3}$. Using the bound $1 - x < e^{-x}$ which holds for all x, we obtain:

$$p_1^* \geqslant 1 - \left(1 - \frac{\ln(1/\varepsilon)}{t}\right)^t \geqslant 1 - \exp(-\ln(1/\varepsilon)) = 1 - \varepsilon. \tag{10.11}$$

This completes the proof. □

Observe that Lemma 10.5 again does not make use of Assumption 8.1. We will only use this assumption for analyzing the collision probabilities for distant vectors below.

10.4.7 – Distant vectors collide with low probability. Proving that distant vectors at angle more than 60° do not often lead to hash collisions is more involved. We need to average the probability of a collision over all possible angles between $\boldsymbol{v}$ and $\boldsymbol{w}$, given that $\boldsymbol{v}$ and $\boldsymbol{w}$ cannot reduce one another, where we thus have to take the density of angles Θ into account. Since it is not so easy to compute the exact distribution of angles that may occur between list vectors throughout the algorithm, we will use Assumption 8.1. To obtain a tight bound on the average probability of a "useless hash collision" we will further use the following lemma about the surface area of hyperspheres. This formula can be found in e.g. [CS99, p. 10, Eq. (19)].

Lemma 10.6. *The hypersurface area or volume of the d-dimensional hypersphere of radius R, defined as $S^{d-1}(R) = \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| = R\}$, is equal to*

$$A_d(R) = \frac{2\pi^{d/2}}{\Gamma(d/2)} R^{d-1}. \tag{10.12}$$

Assumption 8.1 together with the above lemma allow us to derive the density of angles $f(\theta)$ between non-reducing vectors explicitly as follows.

Lemma 10.7. *If Assumption 8.1 holds, then the probability density function* $f(\theta)$ *of angles between non-reducing vectors satisfies*

$$f(\theta) = \sqrt{\frac{2d}{\pi}}\,(\sin\theta)^{d-2}\,[1+o(1)] = 2^{\log_2(\sin\theta)\,d+o(d)}. \tag{10.13}$$

Proof. Suppose without loss of generality that $\boldsymbol{v} = (1,0,\ldots,0)$ is fixed. To derive the density at a given angle θ, we basically need to know the fraction of points in $S^{d-1}(R)$ that have this angle with $\boldsymbol{v}$. Note that if the angle is fixed at θ, then the first coordinate of $\boldsymbol{w}$ is $\cos\theta$ and so the remaining coordinates of $\boldsymbol{w}$ must satisfy

$$w_2^2 + \cdots + w_d^2 = 1 - \cos^2\theta = \sin^2\theta. \tag{10.14}$$

This equation defines a $(d-1)$-dimensional hypersphere with radius $\sin\theta$, whose volume follows from Lemma 10.6. Dividing by the total volume of $S^{d-1}(R)$, the density function f satisfies

$$f(\theta) = \frac{1}{M}A_{d-1}(\sin\theta), \qquad \text{where } M = \int_{\pi/3}^{\pi/2} A_{d-1}(\sin\phi)\,d\phi. \tag{10.15}$$

For the normalizing constant M, note that integrating the given expression from 0 to $\frac{\pi}{2}$ would give us exactly half the hypersurface area of the d-dimensional hypersphere of radius 1, and the contribution of angles less than $\frac{\pi}{3}$ is negligible for large d. Writing out the expressions for $A_{d-1}(\sin\theta)$ and $M \sim A_d(1)$, we therefore obtain

$$f(\theta) = \frac{A_{d-1}(\sin\theta)}{(1-o(1))\frac{1}{2}A_d(1)} = \frac{2}{\sqrt{\pi}}\cdot\frac{\Gamma(\frac{d}{2})}{\Gamma(\frac{d-1}{2})}\cdot(\sin\theta)^{d-2}\,[1+o(1)]\,. \tag{10.16}$$

Noting that $\Gamma(x+\frac{1}{2}) \sim \sqrt{x}\cdot\Gamma(x)$ for large x, the result follows. □

We are now ready to prove the main result, showing that bad collisions occur with exponentially small probability. We first prove a general result relating the probability of a bad collision to the parameter t, and then show how to choose t to balance the time and space complexities. Here we write $\gamma_1 = \frac{1}{2}\log_2(\frac{4}{3}) \approx 0.2075$, $\gamma_2 = \log_2(\frac{3}{2}) \approx 0.5850$, and we write $n = 2^{c_n\cdot d}$ and $t = 2^{c_t\cdot d}$ for certain constants c_n and c_t.

Lemma 10.8. *Let* $c_n \geqslant \gamma_1$. *Then, if Assumption 8.1 holds, for large* d *the probability of bad collisions is bounded by*

$$p_2^* = \mathbb{P}_{\{h_{i,j}\}\subset\mathcal{H}}(\boldsymbol{v},\boldsymbol{w}\ \textit{collide} \mid \theta(\boldsymbol{v},\boldsymbol{w}) > \tfrac{\pi}{3}) \leqslant n^{-\alpha+o(1)}, \tag{10.17}$$

where $\alpha \in (0,1)$ *is defined as*[2]

$$\alpha = \frac{-1}{c_n}\left[c_t + \max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})}\left\{\log_2(\sin\theta) + \frac{c_t}{\gamma_2}\log_2\left(1-\frac{\theta}{\pi}\right)\right\}\right]. \tag{10.18}$$

[2]The inequality $\alpha > 0$ can be derived by splitting the maximum in (10.18) in two parts, and solving each separately. Strictly speaking the inequality $\alpha < 1$ is an *assumption* on the other parameters, which is ultimately tangential to the result; for all parameters of interest, the expression for α in (10.18) will not be larger than 1.

Proof. First, if we know the angle $\theta \in (\frac{\pi}{3}, \frac{\pi}{2})$ between two non-reducing vectors, then the probability of a collision is $p^*(\theta) = 1 - (1 - (1 - \frac{\theta}{\pi})^k)^t$. Letting $f(\theta)$ denote the density of angles θ on Ω, we have

$$p_2^* = \mathbb{E}_{\Theta \in (\frac{\pi}{3}, \frac{\pi}{2})} [p^*(\Theta)] = \int_{\pi/3}^{\pi/2} f(\theta) p^*(\theta) d\theta. \tag{10.19}$$

Substituting $p^*(\theta) = 1 - (1 - (1 - \frac{\theta}{\pi})^k)^t$ and substituting the expression of Lemma 10.7 for $f(\theta)$, we obtain

$$p_2^* = \sqrt{\frac{2d}{\pi}} \int_{\pi/3}^{\pi/2} (\sin\theta)^{d-2} [1 + o(1)] \left[1 - \left(1 - \left(1 - \frac{\theta}{\pi} \right)^k \right)^t \right] d\theta. \tag{10.20}$$

Next, note that for $\theta \gg \frac{\pi}{3}$ we have $t \ll (1 - \frac{\theta}{\pi})^{-k}$ and so $(1 - \left(1 - \frac{\theta}{\pi}\right)^k)^t \approx 1 - t\left(1 - \frac{\theta}{\pi}\right)^k$. In that case, we can simplify the expression between square brackets to $t \cdot \left(1 - \frac{\theta}{\pi}\right)^k$. However, the integration range includes $\frac{\pi}{3}$ as well, so to be careful we will divide the range of integration into two disjoint intervals $[\frac{\pi}{3}, \frac{\pi}{3} + \delta]$ and $[\frac{\pi}{3} + \delta, \frac{\pi}{2}]$, where $\delta = O(\sqrt{d})$:

$$p_2^* = \underbrace{\int_{\pi/3}^{\pi/3+\delta} f(\theta) p^*(\theta) d\theta}_{I_1} + \underbrace{\int_{\pi/3+\delta}^{\pi/2} f(\theta) p^*(\theta) d\theta}_{I_2}. \tag{10.21}$$

Bounding I_1. Using $f(\theta) \leqslant f(\frac{\pi}{3} + \delta) = \text{poly}(d) \cdot \sin^{d-2}\left(\frac{\pi}{3} + \delta\right)$ and $p^*(\theta) \leqslant p^*(\frac{\pi}{3}) = 1 - (1 - \frac{\ln(1/\varepsilon)}{t})^t \approx 1 - \varepsilon$, we obtain

$$I_1 \leqslant \text{poly}(d) \cdot \delta(1 - \varepsilon) \sin^{d-2}\left(\frac{\pi}{3} + \delta\right). \tag{10.22}$$

A Taylor expansion around $\theta = \frac{\pi}{3}$ of $\sin\theta$ tells us that $\sin(\frac{\pi}{3} + \delta) = \frac{1}{2}\sqrt{3}\,[1 + O(\delta)]$, which for constant $\varepsilon \in (0, 1)$ leads to

$$I_1 \leqslant 2^{-\gamma_1 d + o(d)} (1 + O(\delta))^d = 2^{-\gamma_1 d + o(d)}. \tag{10.23}$$

Bounding I_2. For I_2, this choice of δ is sufficient to make the approximation $(1 - \left(1 - \frac{\theta}{\pi}\right)^k)^t \approx 1 - t\left(1 - \frac{\theta}{\pi}\right)^k$ work. So for I_2 we obtain the simplified expression

$$I_2 \leqslant \text{poly}(d) \cdot t \cdot \int_{\pi/3+\delta}^{\pi/2} (\sin\theta)^{d-2} \left(1 - \frac{\theta}{\pi}\right)^k d\theta \tag{10.24}$$

$$\leqslant \int_{\pi/3}^{\pi/2} 2^{d \log_2(\sin\theta) + k \log_2(1 - \frac{\theta}{\pi}) + c_t d + o(d)} d\theta. \tag{10.25}$$

Note that the integrand is exponential in d (assuming k is at most linear d) and the exponent is a continuous, differentiable function of θ. So the asymptotic behavior of the integral is the same as the asymptotic behavior of its maximum value:

$$\log_2 I_2 \leqslant c_t d + \max_{\theta \in (\frac{\pi}{3}, \frac{\pi}{2})} \left\{ d \log_2(\sin\theta) + k \log_2\left(1 - \frac{\theta}{\pi}\right) \right\} + o(d). \tag{10.26}$$

Bounding $p_2^* = I_1 + I_2$. Combining the results from (10.23) and (10.26), and substituting the expression for k from Lemma 10.5 (assuming $\varepsilon > 0$ is fixed), we have

$$\frac{\log_2 p_2^*}{d} \leqslant \max\left\{-\gamma_1,\ c_t + \max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})}\left\{\log_2(\sin\theta) + \frac{c_t}{\gamma_2}\log_2\left(1-\frac{\theta}{\pi}\right)\right\}\right\} + o(1). \tag{10.27}$$

In the end, we would like to prove that $p_2^* \leqslant n^{-\alpha}$, or equivalently $\frac{1}{d}\log_2 p_2^* \leqslant -\alpha c_n$. To complete the proof, it therefore suffices to prove the following two inequalities:

$$-\gamma_1 \leqslant -\alpha c_n + o(1). \tag{10.28}$$

$$c_t + \max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})}\left\{\log_2(\sin\theta) + \frac{c_t}{\gamma_2}\log_2\left(1-\frac{\theta}{\pi}\right)\right\} \leqslant -\alpha c_n + o(1). \tag{10.29}$$

The first inequality follows from the assumptions $n = 2^{c_n d + o(d)}$ with $c_n \geqslant \gamma_1$ and $\alpha < 1$. For the second inequality, we isolate α to obtain

$$\alpha \leqslant \frac{-1}{c_n}\left[c_t + \max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})}\left\{\log_2(\sin\theta) + \frac{c_t}{\gamma_2}\log_2\left(1-\frac{\theta}{\pi}\right)\right\}\right] + o(1). \tag{10.30}$$

Since we want α to be as large as possible, so that p_2^* is as small as possible, we set α equal to its upper bound, leading to the result. □

10.4.8 – Balancing the parameters. To return to actual consequences for the complexity of the scheme, recall that the overall time and space complexities are given by:

- Time (hashing): $\tilde{O}(n \cdot t) = 2^{(c_n + c_t)d + o(d)}$.
- Time (searching): $\tilde{O}(n^2 \cdot p_2^*) = 2^{(c_n + (1-\alpha)c_n)d + o(d)}$.
- Time (overall): $2^{(c_n + \max\{c_t, (1-\alpha)c_n\})d + o(d)}$.
- Space: $\tilde{O}(n \cdot t) = 2^{(c_n + c_t)d + o(d)}$.

Writing the overall time complexity as $2^{c_{\text{time}} d + o(d)}$ and the asymptotic space complexity as $2^{c_{\text{space}} d + o(d)}$, this means

$$c_{\text{time}} = c_n + \max\{c_t, (1-\alpha)c_n\}, \qquad c_{\text{space}} = c_n + c_t. \tag{10.31}$$

Also recall that as we will still find nearby vectors with high probability if they exist similar to the Nguyễn–Vidick sieve, we expect the required list size to be of the order $n = (4/3)^{d/2 + o(d)}/(1-\varepsilon)$, which for constant $\varepsilon > 0$ means that $c_d = \gamma_1$. To balance the asymptotic time complexities for hashing and searching, so that the time and space complexities are the same and the time complexity is minimized, we solve $(1-\alpha)\gamma_1 = c_t$ numerically for c_t, where we note that α is implicitly a function of c_t as well. The following corollary describes the result of this optimization, where θ^* denotes the angle at which the upper bound for α is attained in Lemma 10.8.

Corollary 10.9. *Taking* $c_t \approx 0.129043$ *leads to:*

$$\theta^* \approx 0.458921\pi, \quad \alpha \approx 0.378163, \quad c_{\text{time}} \approx 0.336562, \quad c_{\text{space}} \approx 0.336562.$$

In other words, using $t \approx 2^{0.129043 d}$ *hash tables and a hash length of* $k \approx 0.220600 d$*, the heuristic time and space complexities of the algorithm are balanced at* $2^{0.336562 d + o(d)}$.

Algorithm 10.3 Nguyễn and Vidick's lattice sieve (with hyperplane LSH, space-efficient)

Require: An input list L of $(4/3)^{d/2+o(d)}$ vectors of norm at most R
Ensure: The output list L' has $(4/3)^{d/2+o(d)}$ vectors of norm at most $\gamma \cdot R$
Initialize an empty list L'
Sample $C \subset L \cap \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| \geqslant \gamma \cdot R\}$ of size $\mathrm{poly}(d) \cdot (4/3)^{d/2}$
for each $i \in \{1, \dots, t\}$ **do**
    Initialize an empty hash table T and sample k random hash functions $h_{i,j} \in \mathcal{H}$
    for each $\boldsymbol{w} \in C$ **do**
        Add $\boldsymbol{w}$ to the hash table T, to bucket $T[h_i(\boldsymbol{w})]$
    for each $\boldsymbol{v} \in L \setminus C$ **do**
        if $\|\boldsymbol{v}\| \leqslant \gamma R$ **then**
            Add $\boldsymbol{v}$ to the list L'
        else
            Obtain the (partial) set of candidates $\mathcal{C} = T[h_i(\boldsymbol{v})]$
            for each $\boldsymbol{w} \in \mathcal{C}$ **do**
                if $\|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma R$ **then**
                    Add $\boldsymbol{v}$ to the list L'
                    Continue the loop over "$\boldsymbol{v} \in L \setminus C$"

10.4.9 – Trade-offs between space and time. Finally, setting $c_t = 0$ leads to the original Nguyễn–Vidick sieve, while $c_t \approx 0.129043$ minimizes the heuristic time complexity at the cost of more space. One can obtain a continuous time-memory trade-off between the Nguyễn–Vidick sieve and the new algorithm by considering values $c_t \in (0, 0.129043)$. Numerically evaluating the resulting time and space complexities for this range of values of c_t leads to the blue curve shown in Figure 10.1.

10.4.10 – Solving SVP in time $2^{0.3366d+o(d)}$ and space $2^{0.2075d+o(d)}$. For the Nguyễn–Vidick sieve [NV08], we can actually process the hash tables sequentially and eliminate the need of storing exponentially many hash tables in memory at the same time. The simple but crucial modification that we can make to this algorithm, in similar fashion to the sequential filtering idea described in [BGJ15, MO15], is that we process the hash tables one by one; we first construct the first hash table, add all vectors in C to this hash table, and look for short difference vectors to add to L'. By then deleting this hash table from memory and building a new hash table (repeating this $t = 2^{0.13d+o(d)}$ times) we keep adding more and more vectors to L' until finally we will again have found the exact same set of short difference vectors for the next iteration. In this case however we never stored all hash tables in memory at the same time, and the memory increase compared to the NV-sieve is asymptotically negligible. This modification leads to Algorithm 10.3, and the previous discussion also immediately leads to the following result.

Theorem 10.10. *The space-efficient Nguyễn–Vidick sieve with hyperplane LSH with*

$$k = 0.2206d + o(d), \qquad t = 2^{0.1290d+o(d)}, \tag{10.32}$$

and $\gamma \to 1$ heuristically solves SVP in time $2^{0.3366d+o(d)}$ and space $2^{0.2075d+o(d)}$. These complexities are indicated by the leftmost blue point in Figure 10.1.

Algorithm 10.4 The GaussSieve algorithm (without hyperplane LSH)

```
1:  Initialize an empty list L and an empty stack S
2:  repeat
3:      Get a vector v from the stack (or sample a new one if S = ∅)
4:      for each w ∈ L do
5:          Reduce v with w
6:          Reduce w with v
7:          if w has changed then
8:              Remove w from the list L
9:              Add w to the stack S (unless w = 0)
10:     if v has changed then
11:         Add v to the stack S (unless v = 0)
12:     else
13:         Add v to the list L
14: until v is a shortest vector
```

Note that this choice of parameters k, t still balances the costs of computing hashes and comparing vectors; the fact that the blue point in Figure 10.1 does not lie on the "Time = Space"-diagonal does not mean we can reduce the time complexity by increasing t.

10.5 — The GaussSieve with hyperplane LSH

Let us next describe how hyperplane LSH can also be used to speed up the GaussSieve of Micciancio and Voulgaris [MV10b]. This algorithm is our main focus for practical applications, since it seems to be the fastest and most space-efficient sieving algorithm to date, which is further motivated by the extensive attention it has received in recent years [BNvdP14, FBB+14, IKMT14, Kle14, MODB14, MTB14, MS11, Sch11, Sch13] and by the fact that the current highest sieving records in the SVP challenge database were obtained using (a modification of) the GaussSieve [Kle14, SG15].

10.5.1 – The GaussSieve algorithm. A simplified version of the GaussSieve algorithm of Micciancio and Voulgaris is described in Algorithm 10.4. The algorithm iteratively builds a longer and longer list L of lattice vectors, occasionally reducing the lengths of list vectors in the process, until at some point this list L contains a shortest vector. Vectors are again sampled from a discrete Gaussian over the lattice, using e.g. the sampling algorithm of Klein [Kle00, MV10b], or popped from the stack if the stack is not empty. If list vectors are modified or newly sampled vectors are reduced, they are pushed to the stack.

In the GaussSieve, the reductions in Lines 5 and 6 follow the rule:

$$\text{Reduce } \mathbf{u}_1 \text{ with } \mathbf{u}_2 : \quad \text{if } \|\mathbf{u}_1 \pm \mathbf{u}_2\| < \|\mathbf{u}_1\| \text{ then } \mathbf{u}_1 \leftarrow \mathbf{u}_1 \pm \mathbf{u}_2. \tag{10.33}$$

Throughout the execution of the algorithm, the list L is always pairwise reduced w.r.t. (10.33), i.e., $\|\mathbf{w}_1 \pm \mathbf{w}_2\| \geqslant \max\{\|\mathbf{w}_1\|, \|\mathbf{w}_2\|\}$ for all $\mathbf{w}_1, \mathbf{w}_2 \in L$. This implies that two list vectors $\mathbf{w}_1, \mathbf{w}_2 \in L$ always have an angle of at least $60°$; otherwise one of them would have been used to reduce the other before being added to the list. Since all angles between list vectors are always at least $60°$, the size of L is bounded by the kissing constant

in dimension d: the maximum number of vectors in $\mathbb{R}^d$ one can find such that any two vectors have an angle of at least 60°. Bounds and conjectures on the kissing constant in high dimensions lead us to believe that asymptotically the size of the list L will not exceed $2^{0.2075d+o(d)}$ [CS99], which has been verified in various experiments with the GaussSieve (e.g. [BNvdP14, MODB14, MV10b]).

While the space complexity of the GaussSieve is reasonably well understood, there are no proven bounds on the time complexity of this algorithm. One might estimate that the time complexity is determined by the double loop over L: at any time each pair of vectors $\boldsymbol{w}_1, \boldsymbol{w}_2 \in L$ was compared at least once to see if one could reduce the other, so the time complexity is at least quadratic in |L|. The algorithm further seems to display a similar asymptotic behavior as the NV-sieve in experiments [MV10b, NV08], for which the asymptotic time complexity is heuristically known to be quadratic in |L|, i.e., of the order $2^{0.415d+o(d)}$. One might therefore conjecture that the GaussSieve also has a time complexity of $2^{0.415d+o(d)}$, which closely matches previous experiments with the GaussSieve in high dimensions [Kle14].

Since these conjectured bounds on the space and time complexities are only based on the fact that each pair of vectors $\boldsymbol{w}_1, \boldsymbol{w}_2 \in L$ has an angle of at least 60°, the same estimates apply to any reduction method that guarantees that angles between vectors in L are at least 60°. In particular, if we reduce vectors only if their angle is at most 60° using the following rule:

$$\text{Reduce } \mathbf{u}_1 \text{ with } \mathbf{u}_2 : \quad \text{if } \theta(\mathbf{u}_1, \pm\mathbf{u}_2) \leqslant 60^\circ \text{ and } \|\mathbf{u}_1\| \geqslant \|\mathbf{u}_2\| \text{ then } \mathbf{u}_1 \leftarrow \mathbf{u}_1 \mp \mathbf{u}_2, \tag{10.34}$$

then we expect the same heuristic bounds on the time and space complexities to apply. More precisely, the list size would again be bounded by $2^{0.208d+o(d)}$, and the time complexity may again be estimated to be of the order $2^{0.415d+o(d)}$. Basic experiments show that, although with this notion of reduction the list size increases, the increase in the list size appears to be sub-exponential in d.

10.5.2 – The HashSieve algorithm. Replacing the notion of reduction of (10.33) by the weaker one of (10.34), we can clearly see the connection with hyperplane hashing. Considering the GaussSieve with angular reductions, we are repeatedly sampling new target vectors $\boldsymbol{v}$ (with each time almost the same list L), and each time we are looking for vectors $\boldsymbol{w} \in L$ whose angle with $\boldsymbol{v}$ is at most 60°. Replacing the brute-force list search in the original algorithm with the technique of hyperplane locality-sensitive hashing, we obtain Algorithm 10.5. Blue lines in Algorithm 10.5 indicate modifications to the GaussSieve. Note that the setup costs of locality-sensitive hashing are again spread out over the various iterations; at each iteration we only update the parts of the hash tables that were affected by updating L.

Finally, there is no point in skipping potential reductions in Lines 7 and 8. So while for our intuition and for the theoretical motivation we may consider the case where the reductions are based on (10.34), in practice we will again reduce vectors based on (10.33).

10.5.3 – Reducing the space complexity with probing. For the Nguyễn-Vidick sieve, recall that we could process the hash tables sequentially rather than in parallel, to prevent getting an increased space complexity. For the GaussSieve the same trick does not seem to apply, and we only obtain the (conjectured) space/time trade-off of Theorem 10.4,

Algorithm 10.5 The GaussSieve algorithm (with hyperplane LSH) – The HashSieve

Initialize an empty list L and an empty stack S
Initialize t empty hash tables T_i and sample $k \cdot t$ random hash functions $h_{i,j} \in \mathcal{H}$
repeat
  Get a vector $\boldsymbol{v}$ from the stack (or sample a new one if $S = \emptyset$)
  Obtain the set of candidates $C = \bigcup_{i=1}^{t} T_i[h_i(\boldsymbol{v})]$
  for each $\boldsymbol{w} \in C$ **do**
    Reduce $\boldsymbol{v}$ with $\boldsymbol{w}$
    Reduce $\boldsymbol{w}$ with $\boldsymbol{v}$
    if $\boldsymbol{w}$ has changed **then**
      Remove $\boldsymbol{w}$ from the list L
      Remove $\boldsymbol{w}$ from all t hash tables T_i
      Add $\boldsymbol{w}$ to the stack S (unless $\boldsymbol{w} = \mathbf{0}$)
  if $\boldsymbol{v}$ has changed **then**
    Add $\boldsymbol{v}$ to the stack S (unless $\boldsymbol{v} = \mathbf{0}$)
  else
    Add $\boldsymbol{v}$ to the list L
    Add $\boldsymbol{v}$ to all t hash tables T_i
until $\boldsymbol{v}$ is a shortest vector

illustrated in Figure 10.1. As the practicability of sieving seems bounded both by the required amounts of time and memory, this trade-off may not offer much of an improvement overall. Being able to handle the increased memory complexity is crucial to making this method practical in higher dimensions.

To decrease the memory requirement of the hash tables, Panigrahy [Pan06] suggested that instead of using many hash tables and checking only one hash bucket in each table for candidate nearby vectors, one could also *probe* several hash buckets in each hash table for nearby vectors, and use fewer hash tables overall to get a similar quality for the list of candidates, using significantly less memory.

Construction. To illustrate how this method of probing might help in reducing the memory, consider the following modification to the HashSieve algorithm. In each hash table T_i, instead of only checking the bucket labeled $h_i(\boldsymbol{v}) \in \{0,1\}^k$ for candidates, we also check buckets labeled $h_i(\boldsymbol{v}) \oplus \boldsymbol{e}_j$ for all $j \in [k]$, where $\boldsymbol{e}_j$ is the jth unit vector in k dimensions and $\oplus$ represents addition in $\mathbb{Z}_2^k$ (bitwise XOR). In other words, we now consider a vector $\boldsymbol{w} \in L$ as a candidate for reduction if and only if it is separated from $\boldsymbol{v}$ by at most one of the random hyperplanes defined by the hash vectors $\boldsymbol{a}_{i,j}$. This means that in each table we now check $k+1$ hash buckets, instead of only one bucket: we do not only check the vectors with an exact match on all k hash values, but we also consider vectors with $k-1$ matches.

Example. To make this construction even more explicit, consider the following example. Suppose we use a hash length of $k = 5$ and we have a vector $\boldsymbol{v}$ with $h(\boldsymbol{v}) = (h_1(\boldsymbol{v}), h_2(\boldsymbol{v}), h_3(\boldsymbol{v}), h_4(\boldsymbol{v}), h_5(\boldsymbol{v})) = (0,0,1,1,1)$. Now the bucket most likely to contain nearby vectors is the bucket labeled $(0,0,1,1,1)$, which contains all vectors lying on the same side of five random hyperplanes as $\boldsymbol{v}$. However, also those buckets labeled

$(1,0,1,1,1)$, $(0,1,1,1,1)$, $(0,0,0,1,1)$, $(0,0,1,0,1)$, $(0,0,1,1,0)$ are quite likely to contain nearby vectors; these buckets contain vectors in adjacent hash regions with four equal hash values and only one different hash value, i.e. one hyperplane separating these vectors from our target vector $\boldsymbol{v}$. Checking these buckets as well may lead to finding more nearby vectors, while we still only check a small fraction of all vectors spread out over all the 32 hash buckets.

Analysis. To analyze the effect of this modification on the algorithm, let us again make the simplifying assumption that non-reduced vectors have an angle of at most $60°$ with $\boldsymbol{v}$, and reduced vectors have an angle of exactly $90°$ with $\boldsymbol{v}$. For non-reduced vectors, the probability that none of the hyperplanes separate $\boldsymbol{v}$ and $\boldsymbol{w}$ is $(\frac{2}{3})^k$, and the probability that *at most one* of the hyperplanes separates these vectors is $(\frac{2}{3})^k + k(\frac{2}{3})^{k-1}(\frac{1}{3})^1 = (\frac{2}{3})^k\left[1+\frac{k}{2}\right]$. Similarly, for reduced vectors the probability of two vectors landing in the same bucket is $(\frac{1}{2})^k$, and the probability of landing in buckets differing in at most one bit is $(\frac{1}{2})^k[1+k]$. Comparing the probabilities of finding given vectors with and without this technique of *probing* multiple buckets, we see that with probing:

- The probability of finding a nearby vector increases by a factor $1+\frac{k}{2}$.
- The probability of finding a distant vector increases by a factor $1+k$.

Note that an increase in the probability of finding nearby vectors in a hash table by a factor α roughly translates to a decrease of t by a factor α, which is motivated by the approximations $1-(1-\alpha p^k)^{t/\alpha} \approx \frac{t}{\alpha}\cdot\alpha p^k = t\cdot p^k \approx 1-(1-p^k)^t$. Therefore, using this technique of probing, we can use the same value k as before, but a smaller value $t_1 = \frac{t}{1+k/2}$ now suffices to still find nearby vectors with high probability. With these values of k and t_1, the probability of finding distant vectors increases by roughly a factor $\frac{1+k}{1+k/2} < 2$, thus leading to up to 2 times more comparisons overall, and an increase in the time complexity by a factor less than 2. But more importantly, the memory requirement of the hash tables decreases by a factor $1+\frac{k}{2} = O(d)$. To illustrate the possible impact of probing: in dimension 80 we have $k \approx 18$ by Theorem 10.4, and so probing might lead to a loss of a factor 2 in the time complexity, and a gain a factor 10 in the memory requirement.

Multiprobe. The procedure of probing adjacent buckets (buckets at Hamming distance 1) can trivially be generalized to considering all buckets with labels that differ from the hash value $h_i(\boldsymbol{v}) \in \{0,1\}^k$ in at most $0 \leqslant \ell \leqslant k$ bits. For $\ell = O(1)$ and large d, this implies a reduction in the number of hash tables (and the space complexity) by a factor $1+\frac{1}{2}k+\frac{1}{4}\binom{k}{2}+\cdots+\frac{1}{2^\ell}\binom{k}{\ell} = O(d^\ell)$ and an increase in the time complexity by less than a factor $2^\ell = O(1)$. For instance, in dimension 120, without probing we have $(k, t_0 = t) \approx (26, 45700)$; with one level of probing we have $(k, t_1) \approx (26, 3200)$; and with two levels of probing we have $(k, t_2) \approx (26, 450)$. Using two levels of probing, in dimension 120 the space complexity of the hash tables is reduced by a factor more than 100, at the cost of a factor less than 4 increase of the overall time complexity.

Besides using multiple levels of probing and brute-forcing all buckets at each level, one could also consider more sophisticated ways of choosing which buckets to check for candidates. If $\boldsymbol{a}_{i,j}^T\boldsymbol{v} \approx 0$ then it makes more sense to consider the bucket labeled $h_i(\boldsymbol{v}) \oplus \boldsymbol{e}_j$ than if $\boldsymbol{a}_{i,j}^T\boldsymbol{v} \gg 0$ or $\boldsymbol{a}_{i,j}^T\boldsymbol{v} \ll 0$. In other words, if a vector $\boldsymbol{v}$ lies close to the hyperplane defined by $\boldsymbol{a}_{i,j}$, it makes more sense to consider vectors on the other side of the hyperplane as well, than if $\boldsymbol{v}$ lies far away from the hyperplane. Algorithmically speaking,

given $\boldsymbol{v}$ and any bucket $\mathbf{b} \in \{0,1\}^k$, one could for instance compute the probability that a reducing vector is in this bucket labeled $\mathbf{b}$, and only check those buckets with the highest probabilities of containing nearby vectors. For more details, see e.g. [LJW+07].

10.5.4 – Experimental results in moderate dimensions. To verify the theoretical speed-ups, we implemented both the GaussSieve and the GaussSieve-based HashSieve to try to compare the asymptotic trends of these algorithms. For implementing the HashSieve, we note that we can use various simple tweaks to further improve the algorithm's performance. These include:

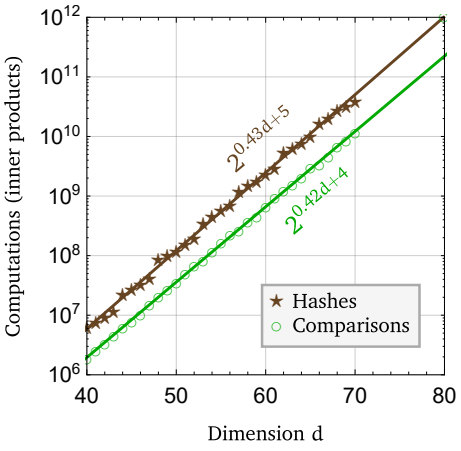
(a) With the HashSieve, maintaining a list L is no longer needed, as the hash tables implicitly keep track of which vectors are in the list.
(b) Instead of making a list of candidates, we go through the hash tables one by one, checking if collisions in this table lead to reductions. If a nearby vector is found early on, this may save up to $t \cdot k$ hash computations and table lookups.
(c) As $h_i(-\boldsymbol{v}) = -h_i(\boldsymbol{v})$ the hash of $-\boldsymbol{v}$ can be computed "for free" from $h_i(\boldsymbol{v})$.
(d) Instead of comparing $\pm\boldsymbol{v}$ to all candidate vectors $\boldsymbol{w}$, we only compare $+\boldsymbol{v}$ to the vectors in the bucket $h_i(\boldsymbol{v})$ and $-\boldsymbol{v}$ to the vectors in the bucket labeled $-h_i(\boldsymbol{v})$. This further reduces the number of comparisons by a factor 2 compared to the GaussSieve, where both comparisons are done for each potential reduction.
(e) For choosing vectors $\mathbf{a}_{i,j}$ to use for the hash functions h_i, there is no reason to assume that drawing these vectors from a specific, sufficiently large random subset of the unit sphere would lead to substantially different results. In particular, using sparse vectors $\mathbf{a}_{i,j}$ makes computing hash values significantly cheaper, while retaining the same performance [Ach01, Ach03, LHC06]. Our experiments indicate that even if all vectors $\mathbf{a}_{i,j}$ have only two equal non-zero entries, the algorithm still finds the shortest vector in (roughly) the same number of iterations as with random vectors $\mathbf{a}_{i,j}$.
(f) We should not store the actual vectors, but only pointers to vectors in each hash table T_i. This means that compared to the GaussSieve, the space complexity roughly increases from $O(n \cdot d)$ to $O(n \cdot d + n \cdot t)$ instead of to $O(n \cdot d \cdot t)$, i.e., an asymptotic increase of a factor t/d rather than t.

With these tweaks, we performed several experiments of finding shortest vectors using the lattices of the SVP challenge [SG15]. We generated lattice bases for different seeds and different dimensions using the SVP challenge generator, used NTL [Sho15] to preprocess the bases (LLL reduction [LLL82] with $\delta = 0.99$), and we then used our implementations of the GaussSieve and HashSieve to obtain these results. For the HashSieve we chose k and t by rounding the theoretical estimates of Theorem 10.4 to the nearest integers, i.e., $k = \lfloor 0.2206d \rceil$ and $t = \lfloor 2^{0.1290d} \rceil$ (see Figure 10.3a). Note that clearly there are ways to further speed up both the GaussSieve and the HashSieve, using e.g. better preprocessing, vectorized code, parallel implementations, optimized samplers, etc. The purpose of our experiments is only to obtain a fair comparison of the two algorithms and to estimate and compare the asymptotic behaviors of these algorithms. Details on a more optimized and parallelized implementation of the HashSieve are given in [MLB15].

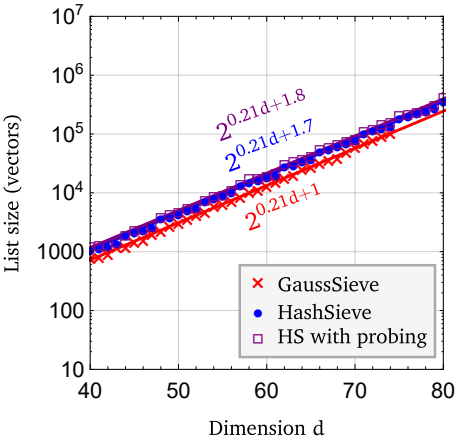
Computations. Figure 10.3b shows the number of inner products computed by the HashSieve for comparing vectors and for computing hashes. We have chosen k and t so that the total time for each of these operations is roughly balanced, and indeed this seems

Dimension (d)	40	45	50	55	60	65	70	75	80	85	90	95	100
Hash length (k)	9	10	11	12	13	14	15	17	18	19	20	21	22
Number of hash tables...													
...without probing (t)	36	56	87	137	214	334	523	817	1278	1999	3126	4888	7643
...with 1-level probing (t_1)	7	9	13	20	29	42	62	86	128	190	284	425	637
...with 2-level probing (t_2)	2	3	4	6	8	11	15	19	26	38	53	76	110

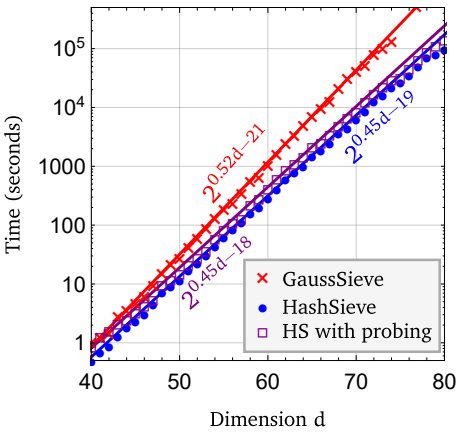
(a) Parameters in the HashSieve (the values of Theorem 10.4, rounded to the nearest integer), without probing (k, t), with one level of probing (k, t_1), and with two levels of probing (k, t_2).



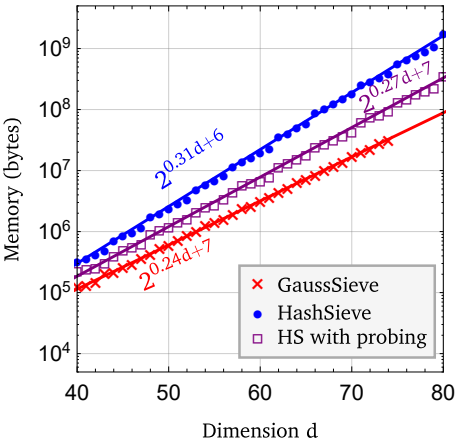
(b) HashSieve computations (without probing)



(c) List sizes



(d) Time complexities



(e) Space complexities

Figure 10.3: Experimental data obtained from applying the GaussSieve and HashSieve (with/without probing) to LLL-reduced random lattice bases. Markers indicate experimental data, lines and line labels represent least-squares fits of the data. Note that the step-wise behavior of some curves can be explained by the fact that the parameter k is small but integral, and increases by 1 only once every four to five dimensions.

to be the case. The total number of inner products for hashing seems to be a constant factor higher than the total number of inner products computed for comparing vectors, which may also be desirable, as hashing is cheaper than comparing vectors using sparse hash vectors. Tuning the parameters differently may slightly change this ratio.

List sizes. In the analysis, we assumed that if reductions are missed with a constant probability $\varepsilon = O(1)$, then the list size also increases by a factor approximately $1/(1-\varepsilon) = O(1)$. Figure 10.3c seems to support this intuition, as indeed the list sizes in the HashSieve seem to be a (small) constant factor larger than in the GaussSieve.

Time complexities. Figure 10.3d compares the timings of the GaussSieve and HashSieve on a single core of a Dell Optiplex 780, which has a processor speed of 2.66 GHz. Theoretically, we expect to achieve a speed-up of roughly $2^{0.078d}$ for each list search, and in practice we see that the asymptotic speed-up of the HashSieve over the GaussSieve is close to $2^{0.07d}$ using a least-squares fit.

Note that the coefficients in the least-squares fits for the time complexities of the GaussSieve and HashSieve are higher than theory suggests, which is in fact consistent with previous experiments in low dimensions [FBB+14, IKMT14, MTB14, MODB14, MV10b]. This phenomenon seems to be caused purely by the low dimensionality of our experiments. Figure 10.3d shows that in higher dimensions, the points start to deviate from the straight line, with a better scaling of the time complexity in higher dimensions. High-dimensional experiments of the GaussSieve ($80 \leqslant d \leqslant 116$) and the HashSieve ($86 \leqslant d \leqslant 96$) demonstrated that these algorithms start following the expected trends of $2^{0.42d+o(d)}$ (GaussSieve) and $2^{0.34d+o(d)}$ (HashSieve) as d gets larger [Kle14, MLB15]. In high dimensions we therefore expect the coefficient 0.3366 to be accurate for the GaussSieve-based HashSieve as well. For more details, see [MLB15].

Space complexities. Figure 10.3e illustrates the experimental space complexities of the tested algorithms for various dimensions. For the GaussSieve, the total space complexity is dominated by the memory required to store the list L. In our experiments we stored each vector coordinate in a register of 4 bytes, and since each vector has d coordinates, this leads to a total space complexity for the GaussSieve of roughly $4dn$ bytes. For the HashSieve the asymptotic space complexity is significantly higher, but recall that in our hash tables we only store pointers to vectors, which may also be only 4 bytes each. For the HashSieve, we estimate the total space complexity to be $4dn + 4tn \sim 4tn$ bytes, i.e., roughly a factor $\frac{t}{d} \approx 2^{0.1290d}/d$ higher than the space complexity of the GaussSieve. Using probing, the memory requirement is further reduced by a significant amount, at the cost of a small increase in the time complexity (Figure 10.3d).

10.5.5 – High-dimensional extrapolations. As explained at the start of this section, the experiments in Section 10.5.4 are aimed at verifying the heuristic analysis and at establishing trends which hold regardless of the amount of optimization of the code, the quality of preprocessing of the input basis, the amount of parallelization etc. However, the linear estimates in Figure 10.3 may not be accurate. For instance, the time complexities of the GaussSieve and HashSieve seem to scale better in higher dimensions; the time complexities may well be $2^{0.415d+o(d)}$ and $2^{0.337d+o(d)}$ respectively, but the contribution of the $o(d)$ only starts to fade away for large d. To get a better feeling of the actual time complexities in high dimensions, one would have to run these algorithms in higher di-

mensions. In recent work [MLB15] we showed that the HashSieve can be parallelized in a similar fashion as the GaussSieve [MTB14]. With better preprocessing, optimized code, and one level of probing, we were able to solve SVP in dimensions up to 107 in less than five days on one multicore machine. Based on experiments in dimensions 86 up to 96, they further estimated the time complexity to lie between $2^{0.32d-15}$ and $2^{0.33d-16}$, which is close to the theoretical estimate $2^{0.3366d+o(d)}$. So although the points in Figure 10.3d almost seem to lie on a line with a different leading constant, these leading constants should not be taken for granted for high-dimensional extrapolations; the theoretical estimate $2^{0.3366d+o(d)}$ seems more accurate.

Finally, let us try to estimate the highest practical dimension d in which the HashSieve may be able to solve SVP right now. The current highest dimension that was attacked using the GaussSieve is $d = 116$, for which 32GB RAM and about 2 core years were needed [Kle14]. Assuming the theoretical estimates for the GaussSieve ($2^{0.4150d+o(d)}$) and HashSieve ($2^{0.3366d+o(d)}$) are accurate, and assuming there is a constant overhead of approximately 2^2 of the HashSieve compared to the GaussSieve (based on the exponents in Figure 10.3d), we might estimate the time complexities of the GaussSieve and HashSieve to be $G(d) = 2^{0.4150d+C}$ and $H(d) = 2^{0.3366d+C+2}$ respectively for some constant C. To solve SVP in the same dimension $d = 116$, we therefore expect to use a factor $G(116)/H(116) \approx 137$ less time using the HashSieve, i.e. a total time complexity of only five core days on the same machine. With approximately two core years, we may further be able to solve SVP in dimension 138 using the HashSieve, which would place sieving near the very top of the SVP hall of fame [SG15]. This does not take into account the space complexity though, which at this point may have increased to several terabytes. Several levels of probing may significantly reduce the required amount of RAM, but further experiments have to be conducted to see how practical the HashSieve is in high dimensions. As in high dimensions the space requirement also becomes an issue, studying the memory-efficient NV-sieve-based HashSieve (with space complexity $2^{0.2075d+o(d)}$) may be an interesting topic for future work.

CHAPTER 11

Hypercone locality-sensitive hashing

11.1 — Overview

Context. In the previous chapter we saw that with the hyperplane hash family of Charikar [Cha02], it is possible to solve SVP heuristically in time $2^{0.3366d+o(d)}$, offering a substantial improvement over both leveled sieving and the overlattices approach (cf. Chapter 9). Although Charikar's random hyperplane method is often considered (one of) the most practical LSH method(s) known for the angular similarity measure, it is not clear if this is really the best method to use in high dimensions. For the high-dimensional regime of large d, there may exist LSH methods which, when combined with sieving, lead to an even better asymptotic time complexity.

To illustrate this, the best known asymptotic lower bound on the exponent ρ for locality-sensitive hashing with approximation factor c is $\rho \geqslant 1/(2c^2-1)$ [AR15b,Dub10]. Under the assumption that all distant vectors are orthogonal (cf. Proposition 10.3), this leads to $c = \sqrt{2}$ and a lower bound $\rho \geqslant \frac{1}{3} \approx 0.33$, while under the same assumption hyperplane LSH only achieves an exponent $\rho = \log_2(\frac{3}{2}) \approx 0.59$, which is significantly higher. Later work of Andoni–Indyk–Nguyễn–Razenshteyn [AI06, AI08, And09, AINR14, AR15a] showed that in certain settings this lower bound on ρ can actually be achieved for e.g. the Euclidean distance, and so a natural question to ask is: could other LSH methods lead to even faster sieving algorithms in high dimensions?

Hypercone locality-sensitive hashing. In this chapter we answer this question in the affirmative. With the hypercone or spherical LSH method of [AINR14, AR15a] we obtain heuristic time and space complexities for solving SVP of $2^{0.2972d+o(d)}$, significantly improving upon the hyperplane LSH method for sieving[1]. Tuning the scheme parameters differently, we obtain the asymptotic space/time trade-off depicted in Figure 11.1. We further show that a practical variant of this algorithm appears to be very similar to the 2-level sieve of Wang–Liu–Tian–Bi [WLTB11] discussed in Chapter 9.

As we will see later in this chapter, an important open problem that arises from these results is: can this hash family be made truly practical? Even though the asymptotic complexity is better than that of the hyperplane LSH-based HashSieve, the order terms

*This chapter is based on results from [LdW15].

[1] Analysis of the Euclidean LSH family of [AI06, AI08] in the context of sieving has shown that using this family would result in an asymptotic time complexity of $2^{0.3326d+o(d)}$ and a time/memory trade-off which are both strictly worse than the results obtained with hypercone LSH (see Figure 11.1). As using Euclidean LSH does not lead to better asymptotic complexities in high dimensions, nor to a more practical algorithm in low dimensions, the derivation of this result is omitted from this thesis.

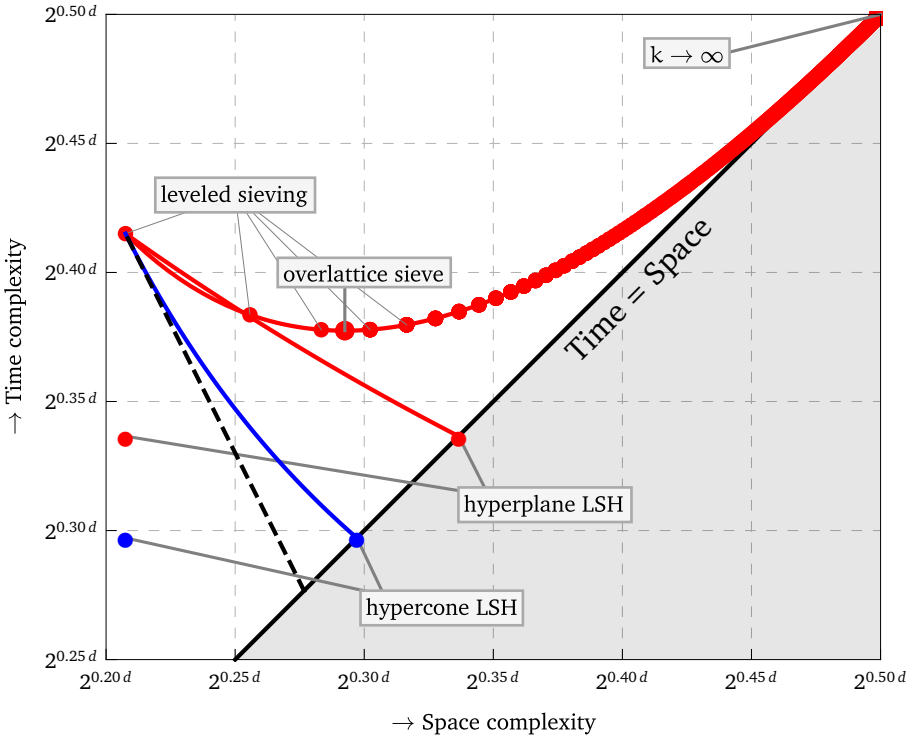


Figure 11.1: The space/time trade-offs from the previous two chapters (red), the estimated trade-off for hypercone LSH (dashed, cf. Proposition 11.3), the actual trade-off for the hypercone LSH sieve (blue, cf. Theorem 11.4), and the optimized complexities obtained by processing the hash tables sequentially in the Nguyễn–Vidick sieve (leftmost blue point, cf. Theorem 11.8).

are significantly larger and so it is unlikely that the method proposed in this chapter in its purest form can compete with the HashSieve from Chapter 10.

Outline. In Section 11.2 we first provide some background on hypercone or spherical LSH. Section 11.3 describes the result of applying this method to the Nguyễn–Vidick sieve [NV08], and states the main result regarding the heuristic complexities for solving SVP with this technique. In Section 11.4 we finally discuss practical implications of these results, and practical limitations of hypercone LSH.

11.2 — Hypercone locality-sensitive hashing

11.2.1 – Spherical LSH. For the spherical hash family described in [AINR14, AR15a], we assume that all points in the data set L lie on the unit sphere $S^{d-1} = \{\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\| = 1\}$. First, we sample $u = 2^{\Theta(\sqrt{d})}$ vectors $\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_u \in \mathbb{R}^d$ from a d-dimensional Gaussian distribution with average norm $\mathbb{E}\|\mathbf{s}_i\| = 1$.[2] This equivalently corresponds to drawing each vector entry from a univariate Gaussian distribution $\mathcal{N}(0, \frac{1}{d})$. To each $\mathbf{s}_i$

[2]Note that Andoni–Indyk–Nguyễn–Razenshteyn sample vectors with average norm $\sqrt{d}$ instead, which means that everything in our description is scaled by a factor $\sqrt{d}$.

we then associate a hash region H_i:

$$H_i = \left\{ \boldsymbol{x} \in \mathcal{S}^{d-1} : \langle \boldsymbol{x}, \boldsymbol{s}_i \rangle \geqslant 1/\sqrt[4]{d} \right\} \setminus \bigcup_{j=1}^{i-1} H_j. \qquad (i = 1, \dots, u) \tag{11.1}$$

Since we assume that $\boldsymbol{x} \in \mathcal{S}^{d-1}$ and with high probability we also have $\|\boldsymbol{s}_i\| \approx 1$ in high dimensions, the condition $\langle \boldsymbol{x}, \boldsymbol{s}_i \rangle \geqslant 1/\sqrt[4]{d}$ can also be read as $\|\boldsymbol{x} - \boldsymbol{s}_i\| \leqslant \sqrt{2} - \Theta(1/\sqrt[4]{d})$, i.e., $\boldsymbol{x}$ lies in the *almost-hemisphere* defined by $\boldsymbol{s}_i$. Finally, the hash of a vector $\boldsymbol{v}$ is given by (the index of) the region it lies in, i.e. if $\boldsymbol{v} \in H_i$, or equivalently $\langle \boldsymbol{v}, \boldsymbol{s}_i \rangle \geqslant 1/\sqrt[4]{d}$ and $\langle \boldsymbol{v}, \boldsymbol{s}_j \rangle \leqslant 1/\sqrt[4]{d}$ for all $j = 1, \dots, i-1$, then $h(\boldsymbol{v}) = i$.

Note that the parts of $\mathcal{S}^{d-1}$ that are covered by multiple hash regions are assigned to the *first* region H_i that covers this part of the sphere. As a result, the expected size of the hash regions decreases with i. Also note that the choice of $u = 2^{\Theta(\sqrt{d})}$ guarantees that with high probability, at the end the entire unit sphere is covered by these hash regions $H_1, H_2, \dots, H_u$; informally, each hash region covers a fraction $2^{-\Theta(\sqrt{d})}$ of the sphere, so we need $2^{\Theta(\sqrt{d})}$ regions to cover the entire sphere. Finally, taking $u = 2^{\Theta(\sqrt{d})}$ also guarantees that computing a hash can trivially be done in subexponential time $2^{\Theta(\sqrt{d})} = 2^{o(d)}$ by going through each of the hash regions $H_1, H_2, \dots, H_u$ one by one and checking whether it contains a given point $\boldsymbol{v}$.

The following result regarding the collision probabilities for spherical LSH is stated in [AINR14, Lemma 3.3] and [AR15a, Appendix B.1].

Lemma 11.1. *Let $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{S}^{d-1}$, and let θ denote the angle between $\boldsymbol{v}$ and $\boldsymbol{w}$. Then the spherical hash family $\mathcal{H}'$ satisfies:*

$$\mathbb{P}_{h' \in \mathcal{H}'}[h'(\boldsymbol{v}) = h'(\boldsymbol{w})] = \exp\left[-\frac{\sqrt{d}}{2} \tan^2\left(\frac{\theta}{2}\right) (1 + o(1)) \right]. \tag{11.2}$$

For $\theta_1 = \frac{\pi}{3}$ and $\theta_2 = \frac{\pi}{2}$ this leads to $\rho = \frac{\ln(p_1)}{\ln(p_2)} = \frac{\tan^2(\pi/6)}{\tan^2(\pi/4)}(1 + o(1)) = \frac{1}{3} + o(1)$, which asymptotically matches the lower bound of Dubiner [Dub10] of $\rho \geqslant 1/(2c^2 - 1)$ for $c = \sqrt{2}$: on the unit sphere, vectors at angle $\frac{\pi}{2}$ lie at distance $\sqrt{2}$ and vectors at angle $\frac{\pi}{3}$ lie at distance 1. Note that this value ρ is significantly smaller than the exponent ρ for hyperplane hashing: with $\theta_1 = \frac{\pi}{3}$ and $\theta_2 = \frac{\pi}{2}$, hyperplane hashing achieves an exponent $\rho = \log_2(\frac{3}{2}) \approx 0.585$ (see the proof of Proposition 10.3).

11.2.2 – Hypercone LSH. Whereas spherical LSH as described above only indicates how to partition the unit sphere in regions, this method can trivially be extended to all of $\mathbb{R}^d$ as follows: given a vector $\boldsymbol{v} \in \mathbb{R}^d$, the *hypercone* hash of a vector $\boldsymbol{v} \in \mathbb{R}^d$ is defined as the spherical hash of the normalized vector $\boldsymbol{v}/\|\boldsymbol{v}\| \in \mathcal{S}^{d-1}$. In other words, denoting the spherical hash family by $\mathcal{H}'$ and the hypercone hash family by $\mathcal{H}$, we have $h(\boldsymbol{v}) = h'(\boldsymbol{v}/\|\boldsymbol{v}\|)$ for $h \in \mathcal{H}$, $h' \in \mathcal{H}'$, and $\boldsymbol{v} \in \mathbb{R}^d$. As spherical hash regions correspond to (set-wise differences of) spherical caps, the normalization extends these hash regions to (set-wise differences of) hypercones.

Since Lemma 11.1 is stated only in terms of angles of vectors, and normalizing vectors does not influence pairwise angles between vectors, the same result of Lemma 11.1 applies to hypercone LSH as well, immediately leading to the following result:

Lemma 11.2. *Let $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^d$, and let θ denote the angle between $\boldsymbol{v}$ and $\boldsymbol{w}$. Then for large d, the hypercone hash family $\mathcal{H}$ satisfies:*

$$\mathbb{P}_{h \in \mathcal{H}}[h(\boldsymbol{v}) = h(\boldsymbol{w})] = \exp\left[-\frac{\sqrt{d}}{2}\tan^2\left(\frac{\theta}{2}\right)(1 + o(1))\right]. \tag{11.3}$$

11.3 — The Nguyễn–Vidick sieve with hypercone LSH

We will now describe the results of applying hypercone LSH to the heuristic sieve algorithm of Nguyễn and Vidick [NV08]. Note that in the Nguyễn–Vidick sieve, vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ are almost assumed to lie on the surface of a sphere, as they lie inside a thin spherical shell with inner radius γR and outer radius R with $\gamma = 1 - o(1)$. So for the Nguyễn–Vidick sieve, one could also use spherical instead of hypercone LSH.

11.3.1 – The Nguyễn–Vidick sieve. Recall that initially the Nguyễn–Vidick sieve starts with a long list L of long lattice vectors (generated using e.g. Klein's sampler [Kle00]), and it iteratively builds shorter lists of shorter lattice vectors L' by applying a sieve to L, as described in e.g. Algorithm 10.1. After $\mathrm{poly}(d)$ applications of the sieve, one hopes to be left with a list containing a shortest non-zero lattice vector, and the bottleneck of the algorithm is applying this sieving procedure $\mathrm{poly}(d)$ times. For further details on this algorithm, see Chapters 8, 9, and 10.

11.3.2 – The (NV-)HyperconeSieve. Algorithm 10.2 in Chapter 10 already describes quite generally how to apply locality-sensitive hashing to the Nguyễn–Vidick sieve to obtain an asymptotic space/time trade-off, and the only things that change compared to Chapter 10 by using hypercone LSH are the hash functions that we use, and the parameters k and t that follow from a careful optimization of the costs.

11.3.3 – A practical (NV-)HyperconeSieve variant. Before analyzing how we should choose the parameters, let us briefly consider how the Nguyễn–Vidick sieve with hypercone LSH could be made slightly more practical. In particular, note that each hash function requires the use of $u = 2^{\Theta(\sqrt{d})}$ random vectors (directions) $\mathbf{s}_1, \ldots, \mathbf{s}_u$. In total, this means that the algorithm uses $t \cdot k \cdot u = \tilde{O}(t)$ random unit vectors to define hash regions on the sphere, and all these vectors need to be generated and stored in memory. Generating so many random vectors from the surface of the unit hypersphere seems unnecessary, especially considering that we already have a list L containing many vectors defining random directions in space, and these are already stored in memory.

The above suggests to make the following modification to the algorithm: for building a single hash function $h_{i,j}$, instead of sampling $\mathbf{s}_1, \ldots, \mathbf{s}_u$ randomly from the surface of the sphere, we randomly sample these vectors from our list of lattice vectors L. In other words, we use the vectors in L to shape the hash regions, rather than sampling and storing new vectors in memory solely for this purpose. According to Assumption 8.1 the directions of these vectors are just as random as if we sampled the vectors from a Gaussian, and so using the same heuristic assumption we can justify that this modification does not drastically alter the behavior of the algorithm. Note that since we need $\tilde{O}(t) \ll \tilde{O}(n)$ random vectors in total (Theorem 11.4 will state exactly how much smaller t is compared to n), the hash functions $h_{i,j}$ can practically be considered independent for sufficiently large d.

11.3.4 – Relation with 2-level sieving. Now, note that for a single hash function, we first use a small set of hash region-defining vectors $\mathbf{s}_1, \ldots, \mathbf{s}_u$ where the radius of each hash region is approximately $(\sqrt{2} - o(1))R$, and we then apply the NV-sieve in each of these regions separately, where a vector is considered nearby if it is within a radius of approximately $(1-o(1))R$ of a center vector. This very closely resembles the ideas behind the 2-level sieve algorithm discussed in Chapter 9, where a list C_1 of outer centers is built (defining balls of radius $\gamma_1 R = (\sqrt{2} - o(1))R$), and each of the centers of this outer list contains an inner list C_2^w of center vectors (defining balls of radius $\gamma_2 R = (1 - o(1))R$). In fact, for $t = k = 1$, this modified HyperconeSieve is almost identical to the 2-level sieve with $\gamma_1 \approx \sqrt{2}$ and $\gamma_2 \approx 1$.

To understand why hypercone hashing performs differently than 2-level sieving, we highlight key differences between the two methods:

- The number of hash regions in the HyperconeSieve is subexponential in the dimension ($u = 2^{\Theta(\sqrt{d})}$), compared to a single exponential number of outer centers in the 2-level sieve ($|C_1| = 2^{\Theta(d)}$). This means that using hypercone LSH, there is no exponential overhead from using this space partitioning.
- Whereas 2-level sieving only uses one partitioning of the space ($t = 1$), leading to an exponential increase in the required number of vectors, with hypercone LSH we use exponentially many rerandomized partitions of the space ($t = 2^{\Theta(d)}$), so that the list size does not increase by an exponential factor.
- The analysis of spherical LSH [AINR14, AR15a] (and the closely related analysis of the celebrated Euclidean LSH family [AI06]) makes crucial use of the fact that the outer list C_1 is *ordered* to compute collision probabilities, and this same order is used each time a vector is assigned to a hash region.

For the last point, note that without this order imposed on the hash regions, the proof of Lemma 11.2 of [AINR14] would not hold, and the performance of spherical/hypercone LSH might be significantly worse.

11.3.5 – High-dimensional intuition. Now, to obtain a first basic estimate of the potential improvements to the time and space complexities using hypercone LSH, and the parameters that we will need to use to obtain the optimized complexities, as in the previous chapter we first note that in high dimensions "almost everything is orthogonal." In other words, angles close to 90° are more likely to occur between two random vectors under Assumption 8.1 than smaller angles. Under the extreme (and imprecise) assumption that all angles between pairwise reduced vectors are *exactly* 90°, we obtain the following estimate for the optimized time and space complexities using hypercone LSH.

Proposition 11.3. *Assuming that non-reducing vectors are always pairwise orthogonal, the NV-sieve with hypercone LSH solves SVP in time and space at most* $(4/3)^{2d/3+o(d)} \approx 2^{0.2767d+o(d)}$*, using the following parameters:*

$$k = \Theta(\sqrt{d}), \qquad t = (4/3)^{d/6+o(d)} \approx 2^{0.0692d+o(d)}. \tag{11.4}$$

Under this assumption, we further get the trade-off between the time and space complexities indicated by the dashed line in Figure 11.1.

Proof. Assuming that all reduced pairs of vectors are orthogonal, we obtain $\rho = \frac{1}{3} + o(1)$ as described in Section 11.2. Since the time complexity is dominated by performing $\tilde{O}(n)$

nearest-neighbor searches on a list of size $n = (4/3)^{d/2+o(d)} \approx 2^{0.2075d+o(d)}$, the result follows from Lemma 10.2. □

11.3.6 – Solving SVP in time and space $2^{0.2972d+o(d)}$. Not all reduced angles are actually 90°, and again we should carefully analyze what is the real probability that a vector $\boldsymbol{w}$ whose angle with $\boldsymbol{v}$ is more than 60°, is found as a candidate due to a collision in at least one of the hash tables. Proposition 11.3 should only be considered a rough estimate, and it gives a lower bound on the best time complexity that we may hope to achieve with this method. Note however that the estimated time complexity is significantly better than the similar estimate obtained for the hyperplane LSH-based HashSieve of Chapter 10, for which the estimated time complexity was $2^{0.3289d+o(d)}$, and that the estimate in Chapter 10 was also not far off from the final result (compare Proposition 10.3 with Theorem 10.4). Therefore, at this point we may already guess that also the exact asymptotic time complexity for hypercone LSH will be better than that of the HashSieve.

The following theorem, which follows from this detailed analysis, shows that this is indeed the case, and it describes exactly what the asymptotic time and space complexities are when the parameters are fully optimized to minimize the asymptotic time complexity.

Theorem 11.4. *The Nguyễn–Vidick sieve with hypercone LSH heuristically solves SVP in time and space $2^{0.2972d+o(d)}$ using the following parameters:*

$$k = 0.3727\sqrt{d} + o\left(\sqrt{d}\right), \qquad t = 2^{0.0896d+o(d)}. \tag{11.5}$$

By varying k and t, we further obtain the trade-off between the time and space complexities indicated by the solid blue curve in Figure 11.1.

Note again that the estimated parameters and complexities from Proposition 11.3 are not far off from the main result of Theorem 11.4, although the difference between the dashed and blue curves in Figure 11.1 is bigger than the difference between these two curves in Figure 10.1. Assuming that reduced vectors are always orthogonal is not realistic, but it provides a reasonable first estimate of the parameters that we have to use.

To prove Theorem 11.4, we will show how to choose a sequence of parameters $\{(k_d, t_d)\}_{d\in\mathbb{N}}$ such that for large d, the following holds:

1. The average probability that a reducing vector $\boldsymbol{w}$ collides with $\boldsymbol{v}$ in at least one of the t hash tables is at least constant in d:
$$p_1^* = \mathbb{P}_{h_{i,j}\in\mathcal{H}}[\boldsymbol{v},\boldsymbol{w} \text{ collide} \mid \theta(\boldsymbol{v},\boldsymbol{w}) \leqslant \tfrac{\pi}{3}] \geqslant 1-\varepsilon. \qquad (0<\varepsilon\neq\varepsilon(d)) \tag{11.6}$$
2. The average probability that a non-reducing vector $\boldsymbol{w}$ collides with $\boldsymbol{v}$ in at least one of the t hash tables is exponentially small:
$$p_2^* = \mathbb{P}_{h_{i,j}\in\mathcal{H}}[\boldsymbol{v},\boldsymbol{w} \text{ collide} \mid \theta(\boldsymbol{v},\boldsymbol{w}) > \tfrac{\pi}{3}] \leqslant n^{-0.5681+o(1)}. \tag{11.7}$$
3. The number of hash tables grows as $t = n^{0.4319+o(1)}$.

This would imply that for each search, the number of candidate vectors for comparison is of the order $n \cdot n^{-0.5681} = n^{0.4319}$. Overall we search the list $\tilde{O}(n)$ times, so after substituting $n = (4/3)^{d/2+o(d)}$ this leads to the following time and space complexities:

- Time (hashing): $\tilde{O}(n \cdot t) = 2^{0.2972d+o(d)}$.
- Time (searching): $\tilde{O}(n^2 \cdot p_2^*) = 2^{0.2972d+o(d)}$.
- Space: $\tilde{O}(n \cdot t) = 2^{0.2972d+o(d)}$.

The next two subsections are dedicated to proving Equations (11.6) and (11.7).

11.3.7 – Nearby vectors collide with constant probability. The following lemma shows how to choose k (in terms of t) to guarantee that Equation (11.6) holds.

Lemma 11.5. *Let $\varepsilon > 0$ and let $k = 6(\ln t - \ln\ln(1/\varepsilon))/\sqrt{d} \approx (6\ln t)/\sqrt{d}$. Then the probability that nearby vectors collide in at least one of the hash tables is at least $1 - \varepsilon$.*

Proof. The probability that a reducing vector $\boldsymbol{w}$ is a candidate vector, given the angle $\Theta = \Theta(\boldsymbol{v}, \boldsymbol{w}) \in (0, \frac{\pi}{3})$, is $p_1^* = \mathbb{E}_{\Theta \in (0, \frac{\pi}{3})}[p^*(\Theta)]$, where we recall that $p^*(\theta) = 1 - (1 - p(\theta)^k)^t$ and $p(\theta) = \mathbb{P}_{h \in \mathcal{H}}[h(\boldsymbol{v}) = h(\boldsymbol{w})]$ is given in Lemma 11.2. Since $p^*(\Theta)$ is strictly decreasing in Θ, we can obtain a lower bound by substituting $\Theta = \frac{\pi}{3}$ above. Using the bound $1 - x \leqslant e^{-x}$ which holds for all x, and inserting the given expression for k, we obtain:

$$p_1^* \geqslant p^*\left(\tfrac{\pi}{3}\right) = 1 - (1 - \exp(\ln\ln(\tfrac{1}{\varepsilon}) - \ln t))^t = 1 - \left(1 - \tfrac{\ln(1/\varepsilon)}{t}\right)^t \geqslant 1 - \varepsilon. \quad (11.8)$$

This completes the proof. □

11.3.8 – Distant vectors collide with low probability. We will again make use of Lemma 10.7, which says that the density at an angle θ is asymptotically proportional to $(\sin\theta)^d$. The following lemma relates the collision probability p_2^* of (11.7) to the parameters k and t. Since Lemma 11.5 relates k to t, this means that only the constant in the exponent of t ultimately remains to be chosen. Here we again write $n = 2^{c_n \cdot d}$, $t = 2^{c_t \cdot d}$, and $\gamma_1 = \frac{1}{2}\log_2(\frac{4}{3}) \approx 0.2075$.

Lemma 11.6. *Let $c_n \geqslant \gamma_1$. Then, if Assumption 8.1 holds, for large d the probability of bad collisions is bounded by*

$$p_2^* = \mathbb{P}_{\{h_{i,j}\} \subset \mathcal{H}}[\boldsymbol{v}, \boldsymbol{w} \text{ collide} \mid \theta(\boldsymbol{v}, \boldsymbol{w}) > \tfrac{\pi}{3}] \leqslant n^{-\alpha + o(1)}, \quad (11.9)$$

where $\alpha \in (0, 1)$ is defined as

$$\alpha = \frac{-1}{c_n}\left[\max_{\theta \in (\frac{\pi}{3}, \frac{\pi}{2})}\left\{\log_2 \sin\theta - \left(3\tan^2\left(\tfrac{\theta}{2}\right) - 1\right)c_t\right\}\right]. \quad (11.10)$$

Proof. First, if we know the angle $\theta \in (\frac{\pi}{3}, \frac{\pi}{2})$ between two bad vectors, then according to Lemma 11.2 the probability of a collision in at least one of the hash tables is equal to

$$p^*(\theta) = 1 - \left(1 - \exp\left[-\frac{k\sqrt{d}}{2}\tan^2\left(\frac{\theta}{2}\right)(1 + o(1))\right]\right)^t. \quad (11.11)$$

Letting $f(\theta)$ denote the density of angles θ on $(\frac{\pi}{3}, \frac{\pi}{2})$, we have

$$p_2^* = \mathbb{E}_{\Theta \in (\frac{\pi}{3}, \frac{\pi}{2})}[p^*(\Theta)] = \int_{\pi/3}^{\pi/2} f(\theta) p^*(\theta) d\theta. \quad (11.12)$$

Substituting $p^*(\theta)$ and the expression of Lemma 10.7 for $f(\theta)$, noting that $\int_{\pi/3}^{\pi/2} f(\theta)d\theta \approx \int_0^{\pi/2} f(\theta)d\theta = 1$ (i.e., the normalizing constant which we omit is negligible), we get

$$p_2^* = \int_{\pi/3}^{\pi/2} (\sin\theta)^d \left[1 - \left(1 - \exp\left[-3\ln t \tan^2\left(\frac{\theta}{2}\right)(1 + o(1))\right]\right)^t\right] d\theta. \quad (11.13)$$

For convenience, let us write $w(\theta) = [-3 \ln t \tan^2\left(\frac{\theta}{2}\right)(1+o(1))$. Note that for θ bounded away from $\frac{\pi}{3}$ we have $w(\theta) \ll -\ln t$ so that $(1-\exp w(\theta))^t \approx 1 - t\exp w(\theta)$, in which case we can simplify the expression between square brackets. However, the integration range includes $\frac{\pi}{3}$ as well, so to be careful we will split the integration interval at $\frac{\pi}{3}+\delta$, where $\delta = \Theta(d^{-1/2})$. (Note that any value δ with $\frac{1}{d} \ll \delta \ll 1$ suffices.)

$$p_2^* = \underbrace{\int_{\pi/3}^{\pi/3+\delta} f(\theta)p^*(\theta)d\theta}_{I_1} + \underbrace{\int_{\pi/3+\delta}^{\pi/2} f(\theta)p^*(\theta)d\theta}_{I_2}. \tag{11.14}$$

Bounding I_1. Using $f(\theta) \leqslant f(\frac{\pi}{3}+\delta)$, $p^*(\theta) \leqslant 1$, and $\sin(\frac{\pi}{3}+\delta) = \frac{1}{2}\sqrt{3}\,[1+O(\delta)]$ (which follows from a Taylor expansion of $\sin x$ around $x = \frac{\pi}{3}$), we obtain

$$I_1 \leqslant \operatorname{poly}(d)\sin^d\left(\frac{\pi}{3}+\delta\right) = \operatorname{poly}(d)\left(\frac{1}{2}\sqrt{3}\right)^d (1+O(\delta))^d = 2^{-\gamma_1 d + o(d)}. \tag{11.15}$$

Bounding I_2. For I_2, our choice of δ is sufficient to make the aforementioned approximation work[3]. Thus, for I_2 we obtain the simplified expression

$$I_2 \leqslant \operatorname{poly}(d)\int_{\pi/3+\delta}^{\pi/2} (\sin\theta)^d t\exp\left[-3\ln t\tan^2\left(\frac{\theta}{2}\right)(1+o(1))\right]d\theta \tag{11.16}$$

$$\leqslant \int_{\pi/3}^{\pi/2} 2^{d\log_2\sin\theta - (3\tan^2(\frac{\theta}{2})-1)\log_2 t + o(d)}d\theta. \tag{11.17}$$

Note that the integrand is exponential in d and that the exponent is a continuous, differentiable function of θ. So the asymptotic behavior of the entire integral I_2 is the same as the asymptotic behavior of the integrand's maximum value:

$$\log_2 I_2 \leqslant \max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})}\left\{d\log_2\sin\theta - \left(3\tan^2(\tfrac{\theta}{2})-1\right)\log_2 t\right\} + o(d). \tag{11.18}$$

Bounding $p_2^* = I_1 + I_2$. Combining (11.15) and (11.18), and substituting the expression for k from Lemma 11.5 (assuming $\varepsilon > 0$ is fixed), we have

$$\frac{\log_2 p_2^*}{d} \leqslant \max\left\{-\gamma_1, \max_{\theta\in(\frac{\pi}{3},\frac{\pi}{2})}\left\{\log_2\sin\theta - (3\tan^2(\tfrac{\theta}{2})-1)c_t\right\}\right\} + o(1). \tag{11.19}$$

The assumption $c_n \geqslant \gamma_1$ and the definition of $\alpha < 1$ now give

$$\frac{\log_2 p_2^*}{d} \leqslant -\alpha c_n + o(1), \tag{11.20}$$

which completes the proof. □

[3] By choosing the order terms in k appropriately, the $o(1)$-term inside $w(\theta)$ may be canceled out, in which case the δ-term dominates. Note that the $o(1)$-term in $w(\theta)$ can be further controlled by the choice of $\gamma = 1 - o(1)$.

11.3.9 – Balancing the parameters. Similar to Chapter 10, we can write the overall time and space complexities as $2^{c_{\text{time}}d+o(d)}$ and $2^{c_{\text{space}}d+o(d)}$ respectively, with

$$c_{\text{time}} = c_n + \max\{c_t, (1-\alpha)c_n\}, \qquad c_{\text{space}} = c_n + c_t. \tag{11.21}$$

To balance the time complexities of hashing and searching, so that the overall time complexity is minimized, we solve $(1-\alpha)\gamma_1 = c_t$ numerically for c_t to obtain the following corollary. Here θ^* denotes the angle θ maximizing the expression in (11.10).

Corollary 11.7. *Taking* $c_t \approx 0.089624$ *leads to:*

$$\theta^* \approx 0.425395\pi, \quad \alpha \approx 0.568115, \quad c_{\text{time}} \approx 0.297143, \quad c_{\text{space}} \approx 0.297143. \tag{11.22}$$

Thus, using $t \approx 2^{0.089624d}$ *hash tables and a hash length of* $k = \Theta(\sqrt{d})$*, the heuristic time and space complexities of the algorithm are balanced at* $2^{0.297143d+o(d)}$.

Note that the dominant angle $\theta^* \approx 0.425395\pi$ is reasonably close to $\frac{1}{2}\pi$, which explains why the final result of Theorem 11.4 is again not far off from the result in Estimate 11.3 based on the assumption that $\theta^* = \frac{1}{2}\pi$.

11.3.10 – Trade-off between the space and time complexities. Finally, $c_t = 0$ leads to the original Nguyễn–Vidick sieve algorithm with a linear space complexity and a quadratic time complexity, while $c_t \approx 0.089624$ minimizes the heuristic time complexity at the cost of more space. One can obtain a continuous trade-off between these two extremes by considering values $c_t \in (0, 0.089624)$. Numerically evaluating the resulting complexities for this range of values of c_t leads to the blue curve in Figure 11.1.

11.3.11 – Solving SVP in time $2^{0.2972d+o(d)}$ and space $2^{0.2075d+o(d)}$. Finally, note that the space complexity increases by a factor t and thus increases exponentially compared to the Nguyễn–Vidick sieve. To get rid of this exponential increase in the memory, instead of storing all hash tables in memory at the same time we could analogously process the hash tables one by one, as previously described in Algorithm 10.3; we first build one hash table by adding all preselected center vectors to their corresponding hash buckets, and for a given target vector we then look for center vectors in the target vector's bucket at distance at most γR. This may lead to some short difference vectors being found, and all the found vectors are added to our list L'. We then repeat this $t = 2^{0.0896n+o(n)}$ times (each time removing the previous hash table from memory) to finally achieve the following result.

Theorem 11.8. *The space-efficient Nguyễn–Vidick sieve with hypercone LSH with*

$$k = 0.3727\sqrt{d} + o\left(\sqrt{d}\right), \qquad t = 2^{0.0896d+o(d)}, \tag{11.23}$$

and $\gamma \to 1$ *heuristically solves SVP in time* $2^{0.2972d+o(d)}$ *and space* $2^{0.2075d+o(d)}$*. These complexities are indicated by the leftmost blue point in Figure 11.1.*

To put the exponent into context, taking into account the list size $n \approx 2^{0.2075d+o(d)}$: a naive search would take time $\tilde{O}(n^2)$, leveled sieving and the overlattices approach reduce the cost to $\tilde{O}(n^{1.82})$, hyperplane LSH reduces the cost to $\tilde{O}(n^{1.62})$, and hypercone LSH reduces it to $\tilde{O}(n^{1.43})$.

11.4 — The GaussSieve with hypercone LSH

11.4.1 – The GaussSieve algorithm. As described in Chapters 8 and 10, a more practical algorithm for solving the shortest vector problem in high dimensions is the GaussSieve algorithm of Micciancio and Voulgaris [MV10b]. Its asymptotic complexities are conjectured to be the same as those of the Nguyễn–Vidick sieve, but the constants are much smaller, and the algorithm is significantly more memory-efficient; the algorithm only adds new vectors to the system when it sees that the current set of vectors does not suffice to find a shortest vector. This is quite different from the Nguyễn–Vidick sieve, where many center vectors are discarded in each iteration of the sieve, and intuitively more memory is used than needed.

11.4.2 – The HyperconeSieve algorithm. For the GaussSieve, it is crucial that we can extend the hash functions to all of $\mathbb{R}^d$, as sampled vectors and list vectors may be significantly different in length. For the GaussSieve it is therefore impossible to use spherical LSH directly, but applying hypercone LSH is straightforward with the previous chapter on hyperplane hashing in mind. As Algorithm 10.5 describes how to apply any LSH method to the GaussSieve, we do not repeat the algorithm description here, and just mention that the HyperconeSieve can be constructed from the GaussSieve by replacing hyperplane LSH by hypercone LSH in Algorithm 10.5, and using the parameters described in Theorem 11.4.

11.4.3 – High-dimensional estimates. Theoretically, Theorem 11.4 and Figure 11.1 show that in high dimensions, hypercone LSH leads to even bigger speed-ups and better space complexities for sieving than the hyperplane LSH method considered in Chapter 10. With heuristic time and space complexities of less than $2^{0.2972d+o(d)} < 2^{3d/10+o(d)}$, one might conclude that in high dimensions, to achieve e.g. $3x$ bits of security for a lattice-based cryptographic primitive relying on the hardness of exact SVP, one should use a lattice of dimension at least $10x$. As most cryptographic schemes are broken even if a "somewhat short" lattice vector is found (which by using BKZ [Sch87, SE94] means that we can significantly reduce the dimension in which we need to run our SVP algorithm), and the time complexity of sieving with hypercone LSH is slightly lower than $2^{3d/10+o(d)}$, one should probably use lattices of dimension much higher than $10k$ to guarantee $3k$ bits of security. So if we simply look at the leading term in the exponent, various parameter choices relying on the estimates of e.g. Chen and Nguyễn [CN11] (solving SVP in dimension 200 takes time approximately 2^{111}) would be too aggressive.

Although the leading term $0.2972d$ in the exponent dominates the complexity in high dimensions, this does not tell the whole story as $o(d)$-terms in the exponent are not negligible for moderate d. While for the hyperplane LSH method of Charikar [Cha02] considered in Chapter 10, hash values can be computed in linear time, with hypercone LSH even the cost of computing a single hash value is already sub-exponential (but super-polynomial) in d. So although the (NV-)HyperconeSieve is asymptotically superior to all previous SVP algorithms, it is not clear whether it will outperform the hyperplane LSH-based HashSieve algorithm of Chapter 10 for any feasible dimension d, let alone whether it will outperform enumeration-based SVP solvers. To really improve upon other SVP algorithms, one would have to find an LSH method which is (i) efficient to compute (i.e. polynomial time for computing a hash value) and (ii) asymptotically (almost) as good as the optimal spherical and hypercone LSH families.

CHAPTER 12

Cross-polytope locality-sensitive hashing

12.1 — Overview

Context. Chapters 10 and 11 described how existing LSH techniques can be used to obtain considerable speed-ups for solving SVP on arbitrary lattices with sieving. The hyperplane LSH method of Charikar [Cha02] discussed in Chapter 10 is simple to analyze and implement but asymptotically suboptimal, while the spherical or hypercone LSH method of Andoni–Indyk–Nguyễn–Razenshteyn [AINR14, AR15a] of Chapter 11 is asymptotically optimal within the LSH framework, but may be less practical due to the large cost of computing hash values. Within the framework of combining LSH with sieving, the best we may hope for is finding a method achieving the best of both worlds, i.e., hash functions with a computational cost comparable to Charikar's method, and an asymptotic performance comparable to hypercone LSH. If such a hash family would further be able to exploit the additional structure in certain lattices used in cryptography (such as cyclic or negacyclic lattices [GGH13, HPS98, HHGPW10, LPR10, SS11]) to obtain additional speed-ups, that would be *ideal*. As such a method does not seem to exist in the literature, the first question to ask is: is it possible to improve upon current LSH techniques, to get these desired properties? And can such techniques be applied to lattice sieving?

Cross-polytope locality-sensitive hashing. In this chapter we show that it is indeed possible to improve upon hypercone hashing in practice, by analyzing a hash family based on cross-polytopes. This family of hash functions was previously proposed by Terasawa and Tanaka [TT07], who already showed that this hash family performs well in practice (and better than e.g. hyperplane hashing), but they did not give any provable bounds on the large-d asymptotics of the collision probabilities or on ρ. We study the asymptotics of this hash family, and show that this family (i) achieves the exact same query exponents ρ for the angular distance as hypercone hashing (see Figure 12.1), and (ii) has a polynomial (quadratic or even sub-quadratic in d) complexity for evaluating a single hash function. This means that we obtain the same asymptotic exponents for solving SVP as with hypercone LSH, with a practical efficiency comparable to hyperplane LSH.

Cyclic (ideal) lattices. Besides the subexponential speed-up for solving SVP on general lattices compared to hypercone LSH (and exponential speed-up over hyperplane LSH), we show that this hash family can also be used to solve SVP on cyclic (ideal) lattices even faster. The cyclic structure of the base hash function allows us to potentially

*This chapter is based on results from [AIL+15, BL15a].

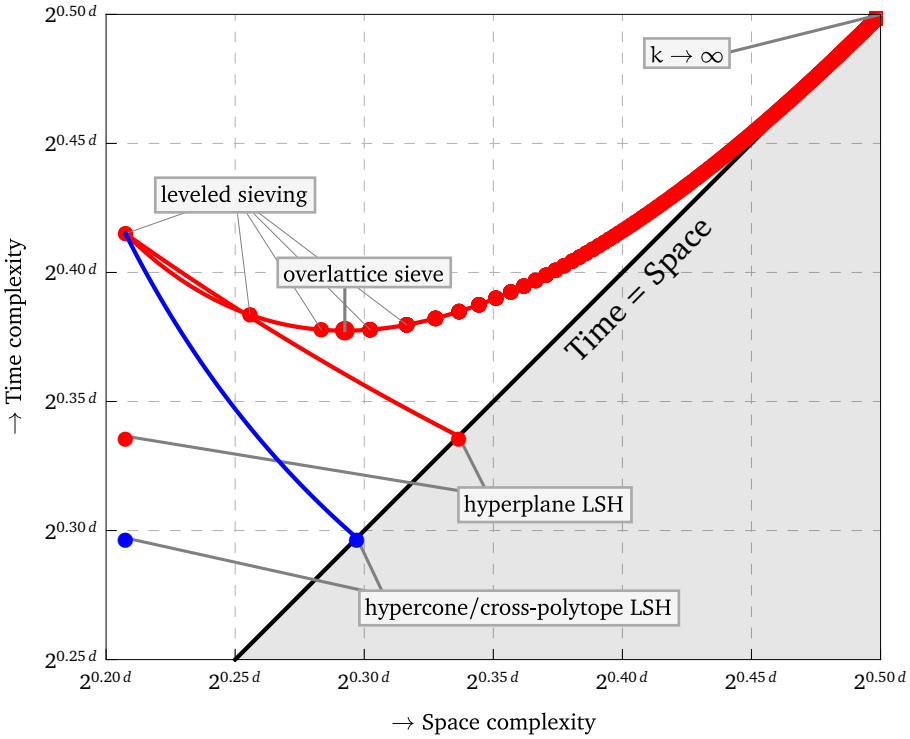


Figure 12.1: The space/time trade-offs described in Chapters 9 and 10 (red), the estimated trade-off for the GaussSieve-based hypercone and cross-polytope LSH sieves (blue, cf. Theorems 11.4 and 12.2), and the optimized complexities obtained by processing the hash tables sequentially in the Nguyễn–Vidick sieve for hypercone and cross-polytope LSH (leftmost blue point, see Theorems 11.8 and 12.2).

obtain a factor d speed-up and decrease in the memory complexity for cyclic lattices. For this however we also need the rerandomized hash functions to satisfy the same cyclic property, which we will show implies that the random (pseudo-)rotation matrices need to be circulant. We show that this extra condition on the rotation matrices does not seem to affect the quality of the rotations significantly, and using such circulant matrices we thus obtain the linear improvement in the time and memory complexity for certain cyclic lattices. Experiments validate these results, and show that indeed cross-polytope LSH may be a significant improvement over both hyperplane LSH and hypercone LSH.

Outline. In Section 12.2 we first describe cross-polytope LSH, and we analyze the asymptotics of this hash family, showing that asymptotically it performs equally well as hypercone LSH, but with a computational efficiency comparable to hyperplane LSH. Section 12.3 describes the theoretical results of applying this method to the Nguyễn–Vidick sieve [NV08], while Section 12.4 describes the results of applying the same techniques to the GaussSieve of Micciancio and Voulgaris [MV10b]. Section 12.5 finally describes how to reduce both the time and space complexity for running SVP on cyclic lattices by a factor $O(d)$, and we illustrate these ideas by applying this algorithm to cyclic lattices appearing in the cryptanalysis of lattice-based cryptography.

12.2 — Cross-polytope locality-sensitive hashing

Let us first describe the cross-polytope LSH method first outlined by Terasawa and Tanaka [TT07], and how it performs in theory and in practice. To construct hash functions, we make use of the vertices of *cross-polytopes*. The d-dimensional cross-polytope is defined as the ℓ_1-unit sphere in $\mathbb{R}^d$, with vertices (corners) given by $\pm e_i$ for $i = 1, \dots, d$ where e_i is the ith unit vector in $\mathbb{R}^d$. The base hash function associated to the cross-polytope then maps a vector x to the nearest vertex to x, i.e., if $x = 2e_1 - 3e_2$, then $h(x) = -e_2$. Note that equivalently

$$h(x) = \operatorname{sgn}(x_{i^*}) \cdot e_{i^*} \qquad (i^* = \operatorname{argmax}_{i \in \{1,\dots,d\}} |x_i|) \tag{12.1}$$

where $\operatorname{sgn}(\alpha) = 1$ if $\alpha \geqslant 0$, and $\operatorname{sgn}(\alpha) = -1$ otherwise, represents the sign of α.

The above only defines one hash function h. To obtain a family of hash functions $\mathcal{H}$, we randomly rotate the cross-polytope in space, and then apply the base hash function h. Equivalently, we first apply the (inverse) rotation to the data set, and then apply the base hash function to this rotated data set. Formally, for $A \in \mathbb{R}^{d \times d}$ a random matrix with i.i.d. Gaussian entries with variance $\frac{1}{d}$, we define the hash function h_A associated to this choice of A by $h_A(x) = h(Ax)$, where h is the base hash function defined above:

$$\mathcal{H} = \left\{ h_A : h_A(x) = h(Ax),\ A = (a_{i,j}),\ a_{i,j} \sim \mathcal{N}(0, \tfrac{1}{d}) \right\}. \tag{12.2}$$

Although A can intuitively be understood to be a rotation matrix, A does not necessarily define a perfect rotation, as $A^T A$ may not be exactly equal to the $d \times d$ identity matrix. We use Gaussians rather than perfect rotation matrices both for the analysis and for implementing the generation of these matrices in practice, although using perfect rotation matrices A should lead to similar asymptotic results.

12.2.1 – Collision probabilities. The main result regarding this hash function family $\mathcal{H}$ is the following theorem.

Theorem 12.1. *Let $\theta = \theta(v, w)$ denote the angle between v and w. Then, for large d,*

$$\mathbb{P}_{h_A \in \mathcal{H}} [h_A(v) = h_A(w)] = \exp\left[-(\ln d) \tan^2\left(\frac{\theta}{2}\right) (1 + o(1)) \right]. \tag{12.3}$$

Sketch of the proof. We will sketch the proof below, where instead of the cross-polytope hash function $h(x) = \pm \operatorname{argmax}_i |x_i|$ we consider the simplex hash function $h(x) = \operatorname{argmax}_i x_i$ also described in [TT07]. For a full, complete proof for the cross-polytope hash functions, we refer the reader to [AIL+15].

Without loss of generality, let $v, w \in \mathbb{R}^d$ be normalized vectors of norm 1; as the hash functions are scale-invariant, we can always normalize the vectors before applying a hash function h_A to it. Let v and w have mutual angle θ, and again w.l.o.g. suppose that $v = (1, 0, \dots, 0)$, and $w = (\cos\theta, \sin\theta, 0, \dots, 0)$, so that $\theta(v, w) = \theta$.

Let $A = (a_1 | a_2 | \dots | a_d)$ be a random Gaussian matrix. Then $Av = a_1$ and $Aw = a_1 \cos\theta + a_2 \sin\theta$. Let us further suppose that $h(v) = e_{i^*}$ for some index i^*. We will inspect the probability that under these circumstances, we also have $h(w) = e_{i^*}$.

First, note that $i^* = \operatorname{argmax}_i (Av)_i = \operatorname{argmax}_i a_{1,i}$ implies that a_{1,i^*} is the maximum over the d normally distributed random variables $a_{1,1}, \dots, a_{1,d} \sim \mathcal{N}(0, \frac{1}{d})$. For large d,

the distribution of $\sqrt{d} \cdot a_{1,i^*}$ converges to the extreme value distribution with parameters $\alpha_d = \sqrt{2 \ln d}\,(1 + o(1))$ and $\beta_d = (1 + o(1))/\sqrt{2 \ln d}$; see e.g. [ABN08, Theorem 8.3.4 and Example 8.3.4]. This distribution has mean $\alpha_d + \gamma_E \beta_d \approx \alpha_d = \Theta(\sqrt{\log d})$, where $\gamma_E \approx 0.577$ is the Euler-Mascheroni constant, and variance $6\beta_d^2/\pi^2 = \Theta(1/\log d)$. Due to the small variance, with high probability a_{1,i^*} will be very close to its mean[1]. For large d, this implies that

$$a_{1,i^*} = \sqrt{\frac{2 \ln d}{d}}\,(1 + o(1)). \tag{12.4}$$

Furthermore, by similar reasoning for $\boldsymbol{w}$ we have $h(\boldsymbol{w}) = \boldsymbol{e}_{i^*}$ if and only if

$$(A\boldsymbol{w})_{i^*} = a_{1,i^*} \cos\theta + a_{2,i^*} \sin\theta = \sqrt{\frac{2 \ln d}{d}}\,(1 + o(1)). \tag{12.5}$$

Intuitively, if $(A\boldsymbol{w})_{i^*}$ is significantly smaller than the right hand side, then there will be another index i' with $(A\boldsymbol{w})_{i'} > (A\boldsymbol{w})_{i^*}$. On the other hand, with high probability there are no indices i with significantly larger values than the right hand side.

Using (12.4), the condition on $(A\boldsymbol{w})_{i^*}$ can be rewritten to a condition on a_{2,i^*} as:

$$a_{2,i^*} = \sqrt{\frac{2 \ln d}{d}} \cdot \frac{1 - \cos\theta}{\sin\theta}\,(1 + o(1)) = \sqrt{\frac{2 \ln d}{d}} \cdot \tan\left(\frac{\theta}{2}\right)(1 + o(1)). \tag{12.6}$$

Recall that i^* is a fixed number between 1 and d indicating the position of the maximum of $\boldsymbol{a}_i$, and this number i^* is independent of the vector $\boldsymbol{a}_2$. As a_{2,i^*} is sampled from a normal distribution $\mathcal{N}(0, \frac{1}{d})$, we can estimate the probability of the above event using standard tail bounds on the normal distribution (see e.g. [AR15a, Lemma B.1]) to obtain

$$\mathbb{P}\left[a_{2,i^*} = \sqrt{\frac{2 \ln d}{d}} \cdot \tan\left(\frac{\theta}{2}\right)(1 + o(1))\right] = \frac{\exp\left[-\ln d \tan^2(\frac{\theta}{2})(1 + o(1))\right]}{\sqrt{4\pi \ln d} \cdot \tan(\frac{\theta}{2})(1 + o(1))}. \tag{12.7}$$

Eliminating terms that disappear in the $o(1)$ in the exponent in the numerator, we get the claim from the theorem. □

12.2.2 – Computational complexity. For the base cross-polytope hash function h, there is an obvious algorithm for computing $h(\boldsymbol{x})$: find the maximum entry of $\boldsymbol{x}$, and include its sign in the hash value. This base hash function can therefore be computed in time $O(d)$. For the rerandomized hash functions we first have to multiply $\boldsymbol{x}$ by a random rotation matrix, the cost of which is (at most) quadratic in d. Therefore the total cost of computing one hash value is $O(d^2)$, i.e. significantly less than for the hypercone hash family considered in the previous chapter with hash cost $2^{\Theta(\sqrt{d})}$. Using structured matrices A with a sufficient amount of randomness (using e.g. Hadamard transformations as described in [AIL+15]), the computational complexity of computing a hash value may be further reduced to $O(d \log d)$.

[1] This argument is not quite formal, as one has to take into account the actual distribution function as well, for in case the distribution has large tails. As mentioned before, the corresponding paper [AIL+15] contains a more detailed and rigorous proof.

12.3 — The Nguyễn–Vidick sieve with cross-polytope LSH

12.3.1 – Solving SVP in time and space $2^{0.2972d+o(d)}$. As in the previous two chapters, applying the technique of cross-polytope locality-sensitive hashing can be done similarly as in Algorithm 10.2. Note that the asymptotic performance of a hash family mainly depends on the exponent ρ, which in this case for given angles θ_1, θ_2 is given by

$$\rho = \frac{\log p(\theta_1)}{\log p(\theta_2)} = \frac{-(\ln d)\tan^2\left(\frac{\theta_1}{2}\right)(1+o(1))}{-(\ln d)\tan^2\left(\frac{\theta_2}{2}\right)(1+o(1))} = \frac{\tan^2\left(\frac{\theta_1}{2}\right)}{\tan^2\left(\frac{\theta_2}{2}\right)}(1+o(1)) \tag{12.8}$$

This is exactly the same exponent as for the hypercone hash family considered in the previous chapter, since the dependence of the asymptotic collision probabilities on the parameters θ_1, θ_2 is exactly the same. In other words, the parameter $\rho(\theta_1, \theta_2)$ for given θ_1, θ_2 is asymptotically equivalent to the exponent $\rho(\theta_1, \theta_2)$ for hypercone LSH, regardless of the values of θ_1, θ_2. As a result, the asymptotic performance of sieving with either LSH technique is exactly the same and we obtain the following result.

Theorem 12.2. *The Nguyễn–Vidick sieve with cross-polytope LSH heuristically solves SVP in time and space $2^{0.2972d+o(d)}$ using the following parameters:*

$$k = \frac{0.2689\,d}{\log_2 d} + o\left(\frac{d}{\log d}\right), \qquad t = 2^{0.0896d+o(d)}. \tag{12.9}$$

By varying k and t, we further obtain the trade-off between the time and space complexities indicated by the solid blue curve in Figure 12.1.

Note that the parameter k has changed compared to hypercone LSH. To find the new value of k, observe that to guarantee that nearby vectors are found with constant probability we need to choose k such that $p(\frac{\pi}{3})^k \sim \frac{1}{t}$. After rewriting, this becomes $k \sim 3c_t \cdot (\frac{d}{\log_2 d}) + o(\frac{d}{\log d})$, which for $c_t \approx 0.0896$ leads to the given leading constant. Although almost linear in d, the value of k is very small in practice; looking at the leading term $k_0 = \frac{0.27d}{\log_2 d}$, for $d = 64$ we have $k_0 < 3$ and for $d = 128$ we have $k_0 < 5$.

12.3.2 – Solving SVP in time $2^{0.2972d+o(d)}$ and space $2^{0.2075d+o(d)}$. As described in the previous two chapters, for the Nguyễn–Vidick sieve we can get rid of the exponential increase in the memory by processing the hash tables sequentially: see Algorithm 10.3. This leads to the following result.

Theorem 12.3. *The space-efficient Nguyễn–Vidick sieve with cross-polytope LSH with*

$$k = \frac{0.2689\,d}{\log_2 d} + o\left(\frac{d}{\log d}\right), \qquad t = 2^{0.0896d+o(d)}, \tag{12.10}$$

and $\gamma \to 1$ heuristically solves SVP in time $2^{0.2972d+o(d)}$ and space $2^{0.2075d+o(d)}$. These complexities are indicated by the leftmost blue point in Figure 12.1.

We stress that the trade-off curve illustrated in Figure 12.1 is in that sense not relevant for the Nguyễn–Vidick sieve, but it is relevant for the GaussSieve, for which we cannot get the strict speed-up described in Theorem 12.3.

12.4 — The GaussSieve with cross-polytope LSH

12.4.1 – The CrossPolytopeSieve. In practice, the GaussSieve achieves a much better time complexity than the Nguyễn–Vidick sieve, and similar to the previous chapters we can apply cross-polytope LSH to the GaussSieve as well, as illustrated in Algorithm 10.5. For this algorithm we expect to achieve a similar asymptotic performance as the Nguyễn–Vidick sieve, as described in Theorem 12.2. Although this already offers a substantial (albeit subexponential) improvement over hypercone LSH, and an exponential improvement over sieving with hyperplane LSH and other sieve algorithms, to make the resulting algorithm truly practical we would like to further reduce the worst-case quadratic cost of computing hashes. Note that in hyperplane LSH the cost of computing one hash function is linear, so that the total cost a k-combined hash function is $k \cdot \Theta(d) = \Theta(d^2)$, while for cross-polytope LSH the cost per function is quadratic, for a total cost of $k \cdot \Theta(d^2) = \Theta(d^3/\log n)$ for each composed hash function.

Theoretically, to compute hashes we first multiply a target vector $\boldsymbol{v}$ by a fully random Gaussian matrix A where each entry $a_{i,j}$ is drawn from the same Gaussian distribution, and then look for the largest coordinate of $\mathbf{r} = A\boldsymbol{v}$; the index of the largest coordinate of $\mathbf{r}$ (together with the sign of this coordinate) will be the hash value. As also described in [Ach01,Laa15d,LHC06], in practice it may be possible to reduce the amount of entropy in the hash functions (the "randomness") without significantly affecting the performance of the scheme. As long as the amount of entropy is high enough that we can build sufficiently many random, independent hash functions, the algorithm will still work fine. Some possibilities to reduce the complexity of computing hashes in practice are:

- Use low-precision floating-point matrices A.
- Use sparse random projection matrices.
- Use structured matrices that allow for fast matrix-vector multiplication.

Using structured matrices that allow for e.g. the use of Fast Fourier Transforms or Fast Hadamard Transforms (see [AIL+15] for details), the cost of computing matrix-vector multiplications may be reduced from $\Theta(d^2)$ to $\Theta(d \log d)$. In that case, the cost per combined hash function $k \cdot \Theta(d \log d) = \Theta(d^2)$ is asymptotically equivalent to hyperplane LSH. In other words, we get the best of both worlds: the same cost for computing hashes as hyperplane LSH, and the same asymptotic performance as hypercone LSH.

12.4.2 – Relation with hyperplane hashing. To put the hash family $\mathcal{H}$ into context, recall that the hyperplane hash family of Charikar [Cha02] used in the HashSieve (Chapter 10) is defined as follows: one samples a random vector $\mathbf{a}$ from a Gaussian distribution, and assigns a hash value to a vector $\boldsymbol{v}$ based on whether the inner product $\langle \boldsymbol{v}, \mathbf{a} \rangle$ is positive or not. Equivalently, we multiply $\boldsymbol{v}$ by a random Gaussian matrix A to get an output vector $\mathbf{r} = A\boldsymbol{v}$, and we check whether $\mathbf{r}$ lies closer to $+\boldsymbol{e}_1$ or closer to $-\boldsymbol{e}_1$, where $\boldsymbol{e}_1 = (1, 0, \ldots, 0)$. Note that only the first row of A influences the hash values.

Formulating the hyperplane hash family this way, we can clearly see similarities with cross-polytope hashing, where all unit vectors $\pm\boldsymbol{e}_i$ are compared to the output vector $\mathbf{r} = A\boldsymbol{v}$ to see which is closest. This suggests a natural generalization of both hyperplane and cross-polytope hashing as follows, called *partial* cross-polytope LSH [AIL+15]:

$$h^{(m)}(\boldsymbol{x}) = \operatorname{sgn}(x_{i^*}) \cdot \boldsymbol{e}_{i^*}. \qquad (i^* = \operatorname{argmax}_{i \in \{1,\ldots,m\}} |x_i|) \tag{12.11}$$

Then a hash family can again be constructed by multiplying a vector by a random Gaussian

matrix A before taking the hash value. Setting $m = 1$ then exactly corresponds to the hyperplane hashing technique of Charikar, while for $m = d$ we obtain the cross-polytope LSH family of (12.1). Note that this generalization with arbitrary m and multiplying with a random Gaussian matrix $A \in \mathbb{R}^{d\times d}$ can also be viewed as first applying a dimension-reducing random projection $A' \in \mathbb{R}^{m\times d}$ (consisting of the first m rows of A) onto a low-dimensional subspace of dimension m, and then using the standard full-dimensional cross-polytope hash function in this m-dimensional space.

12.4.3 – Reducing the memory with probing. The idea of probing, where various hash buckets in each hash table are traversed and checked for nearby vectors to our target vector $\boldsymbol{v}$ (rather than only the bucket labeled $h(\boldsymbol{v})$), can also be applied to the cross-polytope hashing and may lead to a significant reduction in the number of hash tables [LJW+07, Pan06, SLH12]. For a given vector $\boldsymbol{v}$, the *highest-quality* bucket (the bucket most likely to contain vectors for reductions) is the one labeled $h(\boldsymbol{v})$, containing other vectors which also have the same index of the largest coordinate. It is not hard to see that the second-best bucket for reductions with $\boldsymbol{v}$ is exactly that bucket corresponding to the second-largest absolute coordinate of $\boldsymbol{v}$. For instance, if $\boldsymbol{v} = (3, -1, -8, -5, 11)$ then the vectors whose largest absolute coordinate is the fifth coordinate are most likely to be nearby $\boldsymbol{v}$, and the next best option to check is those vectors whose largest coordinate in absolute value is the third coordinate, and whose third coordinate is negative. Therefore, we may consider the indices $i \in \{1, \dots, d\}$ sorted by the absolute values of $(A\boldsymbol{v})_i$ as our probing sequence or "ranking" for $\boldsymbol{v}$ for a single hash function h_A. The further we are in the sequence, the less likely it is that nearby vectors have these hash values.[2]

The remaining question is how to combine multiple cross-polytope rankings when we have more than one hash function in one hash table. Consider two points $\boldsymbol{v} = \boldsymbol{e}_1$ and $\boldsymbol{w} = \boldsymbol{e}_1 \cos\theta + \boldsymbol{e}_2 \sin\theta$ at angle θ. Let $A^{(i)}$ denote the Gaussian matrix of the i-th hash function $h_i = h_{A^{(i)}}$, and let $\mathbf{r}^{(i)} = A^{(i)}\boldsymbol{v}$ be the randomly rotated versions of point $\boldsymbol{v}$. Given $\mathbf{r}^{(i)}$, we are interested in the probability of $\boldsymbol{w}$ hashing to a certain combination of the individual cross-polytope rankings. More formally, let $R^{(i)}_{\alpha_i}$ be the index of the α_i-th largest element of $\mathbf{r}^{(i)}$, where $\boldsymbol{\alpha} \in \{1, \dots, d\}^k$ specifies the alternative probing location. Then we would like to compute

$$\mathop{\mathbb{P}}_{A^{(1)},\dots,A^{(k)}}\left[h_{A^{(i)}}(\boldsymbol{w}) = R^{(i)}_{\alpha_i} \text{ for all } i \in \{1, \dots, k\} \;\middle|\; A^{(i)}\boldsymbol{v} = \mathbf{r}^{(i)}\right] \tag{12.12}$$

$$= \prod_{i=1}^{k} \mathop{\mathbb{P}}_{A^{(i)}}\left[\operatorname{argmax}_j \left\{(A^{(i)}\boldsymbol{e}_1 \cos\theta + A^{(i)}\boldsymbol{e}_2 \sin\theta)_j\right\} = R^{(i)}_{\alpha_i} \middle| A^{(i)}\boldsymbol{e}_1 = \mathbf{r}^{(i)}\right]. \tag{12.13}$$

If we knew this probability for all $\boldsymbol{\alpha} \in \{1, \dots, d\}^k$, we could sort the k-dimensional probing locations by their probabilities. To approximate the individual terms in the latter product efficiently for a single value of i (omitting superscripts), w.l.o.g. we permute the

[2] In order to simplify notation, we again consider the simplex hash family rather than the cross-polytope family, where both the standard basis vector $+\boldsymbol{e}_i$ and its opposite $-\boldsymbol{e}_i$ are mapped to the same hash value. It is easy to extend the multiprobe scheme defined here to cross-polytope LSH; see e.g. [AIL+15].

rows of A so that $R_i = i$ and get

$$\mathop{\mathbb{P}}_{A\sim\mathcal{N}(0,1)^{d\times d}}\left[\operatorname{argmax}_j\left\{(\mathbf{r}\cos\theta + A\boldsymbol{e}_2\sin\theta)_j\right\} = \alpha_i \;\middle|\; A\boldsymbol{e}_1 = \mathbf{r}\right] \tag{12.14}$$

$$= \mathop{\mathbb{P}}_{\mathbf{y}\sim\mathcal{N}(0,1)^{d}}\left[\operatorname{argmax}_j\left\{(\mathbf{r} + \mathbf{y}\tan\theta)_j\right\} = \alpha_i\right]. \tag{12.15}$$

The latter expression is the Gaussian measure of the set $S = \{\mathbf{y} \in \mathbb{R}^d : \operatorname{argmax}_j |(\mathbf{r} + \mathbf{y}\tan\theta)_j| = \alpha_i\}$. We can estimate the measure of S by its distance to the origin (see [AIL+15, Appendix A]). Then the probability of probing location $R^{(i)}_{\alpha_i}$ is proportional to $\exp(-\|\mathbf{y}_{\mathbf{r},\alpha_i}\|^2)$, where $\mathbf{y}_{\mathbf{r},\alpha_i}$ is the shortest vector $\mathbf{y}'$ such that $\operatorname{argmax}_j\{(\mathbf{r}+\mathbf{y})_j\} = \alpha_i$.[3] Note that for computational performance and simplicity, we can make a further approximation and use $\mathbf{y}_{\mathbf{r},\alpha_i} = (\max_j |r_j| - |r_{\alpha_i}|)\cdot \boldsymbol{e}_{\alpha_i}$, i.e., we only consider modifying a single coordinate to reach the set S.

Once we have estimated the probabilities for each $\alpha_i \in \{1,\dots,d\}$ in one layer i, we incrementally construct the probing sequence using a binary heap, similar to the approach in [LJW+07]. For a probing sequence of length m, the resulting algorithm has running time $\tilde{O}(t + m\log m)$.

12.4.4 – Experimental results. To illustrate the practicability of sieving with cross-polytope LSH, as well as to verify the asymptotic behavior, we performed experiments with the GaussSieve with cross-polytope LSH in dimensions 40–80, with parameters as shown in Figure 12.2a. Already in mid-size dimensions ($d > 50$), we observe that the costs are similar to the asymptotic estimates for small choices of k. In low dimensions, we observe that values of k slightly smaller than the theoretical leading term may work better than choosing k larger (rounding up). In the worst case, choosing k too small means following the time/space trade-off curve of Figure 12.1 to the left and up, i.e., using more time at the cost of less space. Choosing k too large may force us to travel further to the right and upwards in Figure 12.1, i.e., using more time and more space.

Overall, in these experiments we already observe that the GaussSieve with cross-polytope LSH has a distinguished lower increase in the time complexity in practice compared to the traditional GaussSieve and the hyperplane LSH-based, GaussSieve-based HashSieve of Chapter 10, and the crossover points between both methods seems to lie in moderate dimensions. As the gap between the CrossPolytopeSieve and other algorithms will only increase as d increases, this clearly highlights the potential of the CrossPolytopeSieve on arbitrary lattices.

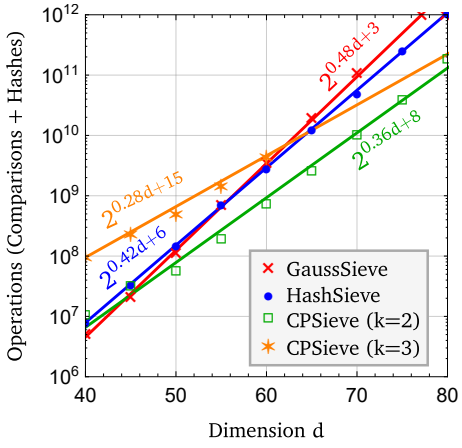
12.5 — The ideal GaussSieve with cross-polytope LSH

While the CrossPolytopeSieve is very capable of solving the shortest vector problem on arbitrary lattices, it was already shown in various papers [BNvdP14, IKMT14, Sch13] that for certain ideal lattices it is possible to obtain substantial polynomial speed-ups for sieving in practice, which may make sieving even more competitive with e.g. enumeration-based SVP solvers. As ideal lattices are commonly used in lattice cryptography, and our

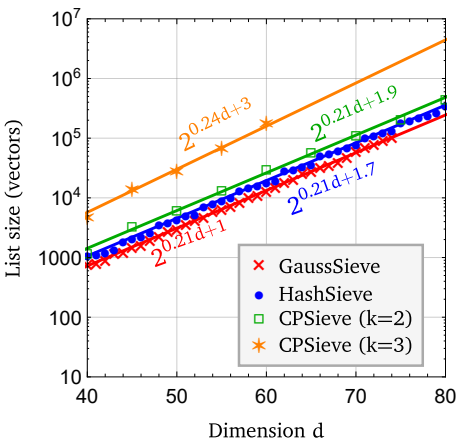
[3] Note that the term $\tan\theta$ has disappeared from the expression. Intuitively it should be clear that the optimal probing sequence does not depend on the angle between two vectors: if we want to find vectors at angle at most 30°, that will not lead to a difference sequence of buckets to check than if we want to find vectors at angle at most 80°.

Dimension (d)	40	45	50	55	60	65	70	75	80	85	90	95	100
Hash length (k)	2.0	2.2	2.4	2.6	2.7	2.9	3.1	3.2	3.4	3.6	3.7	3.9	4.1
Hash tables (t)	12	16	22	30	42	57	77	105	144	196	268	365	498

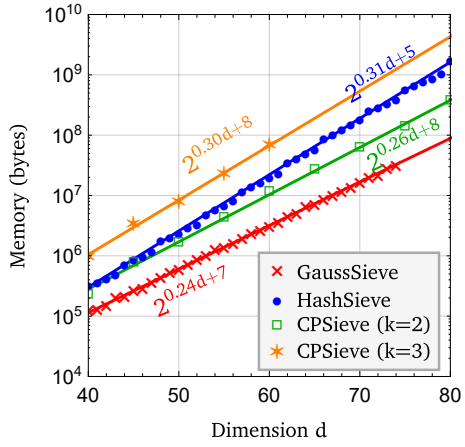
(a) Parameters k, t in the CrossPolytopeSieve (the values of Theorem 12.2, with t rounded to the nearest integer) without taking into account the possible effects of probing. Note that although k is quasi-linear in d, it only increases by 2 as d increases from 40 to 100.



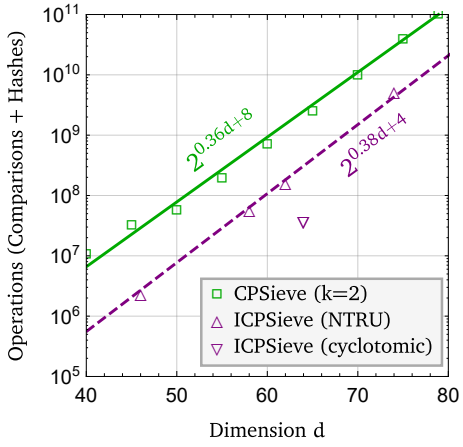
(b) CrossPolytopeSieve computations



(c) List sizes



(d) Space complexities



(e) Ideal vs. non-ideal lattices

Figure 12.2: Experimental data obtained from applying the GaussSieve, HashSieve, and CrossPolytopeSieve (CPSieve) without probing to LLL-reduced random lattice bases from the SVP challenge [SG15]. Markers indicate experimental data, lines and line labels represent least-squares fits of the data. Note that using $k = 3$ in low dimensions leads to a big increase in the list size, as one would have to use a very large number of hash tables to compensate for the fine-grained selection of vectors for comparison, which in turn would lead to a high cost of computing hashes. For smaller t and $k = 3$, this leads to missing nearby vectors quite often, increasing the list size, the memory, and the time till termination. The last figure shows the speed-ups obtained for cyclic lattices such as NTRU lattices and power-of-two cyclotomic lattices, where k is fixed at $k = 2$. As there is only one power of two between 40 and 80, we only have one data point for cyclotomic lattices.

main goal is to estimate the complexity of SVP on lattices that are actually used in lattice cryptography, it is important to know if the CrossPolytopeSieve can be sped up on ideal lattices as well. We will show that this is indeed the case, using similar techniques as in [BNvdP14, IKMT14, Sch13] but where we need to do some extra work to make sure these speed-ups apply here as well.

Ideal lattices are defined in terms of ideals of polynomial rings. Given a ring $R = \mathbb{Z}[X]/(g)$ where $g \in \mathbb{Z}[X]$ is a degree-d monic polynomial, we can represent a polynomial $v(X) = \sum_{i=1}^{d} v_i X^{i-1}$ in this ring by a vector $\boldsymbol{v} = (v_1, \ldots, v_d)$. Then, given a set of generators $b_1, \ldots, b_k \in R$, we define the ideal $I = \langle b_1, \ldots, b_k \rangle$ by the properties (i) $b_1, \ldots, b_k \in I$; (ii) if $a, b \in I$ then also $\lambda a + \mu b \in I$ for scalars $\lambda, \mu \in \mathbb{Z}$; and (iii) if $a \in R$ and $b \in I$ then $a \cdot b \in I$. Note that when these polynomials are translated to vectors, the first property corresponds exactly to the property of a lattice, while the second property makes this an ideal lattice. In terms of lattices, the second property can equivalently be written as:

$$(v_1, \ldots, v_d) \in \mathcal{L} \;\Leftrightarrow\; (w_1, \ldots, w_d) \in \mathcal{L}, \text{ where } w \equiv X \cdot v \bmod g. \tag{12.16}$$

Below we will restrict our attention to two specific choices of g as follows:

Cyclic lattices. If $g(X) = X^d - 1$ and $\boldsymbol{v} = (v_1, \ldots, v_d)$, then $w \equiv X \cdot v$ implies that $\boldsymbol{w} = (v_d, v_1, \ldots, v_{d-1})$, i.e. multiplying a polynomial in the ring by X corresponds to a right-shift (with carry) of the corresponding vector. So in a cyclic lattice, a cyclic shift of a lattice vector is also always in the lattice.

Negacyclic lattices. For the case $g(X) = X^d + 1$, multiplying a polynomial by X in the ring corresponds to a right-shift with carry, but in this case an extra minus sign appears with the carry: $w \equiv X \cdot v$ implies that $\boldsymbol{w} = (-v_d, v_1, \ldots, v_{d-1})$.

Whereas the above descriptions of cyclic and negacyclic lattices are quite general, below we list two instances of these lattices that appear in lattice-based cryptography.

NTRU lattices. Cyclic lattices most notably appear in the cryptanalysis of the NTRU cryptosystem [HPS98, HHGPW10], where the polynomial ring is $R = \mathbb{Z}_q[X]/(X^{d'} - 1)$ for d', q prime. Due to the modular ring, the corresponding lattice is not quite cyclic but rather "block-cyclic". The NTRU lattice is formed by the $d = 2d'$ basis vectors $\mathbf{b}_i = (q \cdot \boldsymbol{e}_i \| \mathbf{0})$ for $i = 1, \ldots, d'$ and $\mathbf{b}_{d'+i} = (\mathbf{h}_i \| \boldsymbol{e}_i)$ for $i = 1, \ldots, d'$, where $\boldsymbol{e}_i$ corresponds to the ith unit vector, and $\mathbf{h}_i$ corresponds to the ith cyclic shift of the public key $\mathbf{h}$ generated from the private key $\mathbf{f}, \mathbf{g}$ (see [HPS98] for details). In this case, if $\boldsymbol{v} = (\boldsymbol{v}_1 \| \boldsymbol{v}_2) \in \mathcal{L}$, then also cyclically shifting both $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ to the right or left leads to a lattice vector. Finding a shortest non-zero vector in this lattice corresponds to finding the secret key $(\mathbf{g} \| \mathbf{f})$ and breaking the cryptosystem, so clearly the complexity of SVP in these lattices is of practical interest.

Power-of-two cyclotomic lattices. Negacyclic lattices commonly appear in lattice cryptography, where $d = 2^k$ is a power of 2 so that, among others, g is irreducible. The 128-dimensional ideal lattice attacked by Ishiguro–Kiyomoto–Miyake–Takagi [IKMT14] and Bos–Naehrig–van de Pol [BNvdP14] from the ideal lattice challenge [PS15] also belongs to this class of lattices. Lattices of this form previously appeared in the context of lattice-based cryptography in e.g. [GGH13, LPR10, SS11].

12.5.1 – The ideal GaussSieve. For the cyclic and negacyclic ideal lattices mentioned above, cyclic shifts of a vector are also in the lattice (modulo minus signs) and have the same Euclidean norm. As first described by Schneider [Sch13], this property can be used in the GaussSieve as follows. First, note that any vector $\boldsymbol{v}$ can be viewed as representing d vectors, namely its d shifted versions $\boldsymbol{v}, \boldsymbol{v}_{(1)}, \boldsymbol{v}_{(2)}, \ldots, \boldsymbol{v}_{(d-1)}$, where we write $\boldsymbol{x}_{(s)} = (x_{d-s+1}, \ldots, x_d, x_1, \ldots, x_{d-s})$ for the sth cyclic right-shift of $\boldsymbol{x} = (x_1, \ldots, x_d)$. Similarly, another vector $\boldsymbol{w}$ represents d different lattice vectors $\boldsymbol{w}, \boldsymbol{w}_{(1)}, \boldsymbol{w}_{(2)}, \ldots, \boldsymbol{w}_{(d-1)}$.

Non-ideal lattices. In the standard GaussSieve, we would treat these $2d$ shifts of $\boldsymbol{v}$ and $\boldsymbol{w}$ as unrelated different vectors, and we would potentially store all of them in the system as we encounter them, leading to a storage cost of $2d$ vectors. Furthermore, to make sure that the list remains pairwise reduced, all $\binom{2d}{2} \approx 2d^2$ pairs of vectors are compared for reductions, leading to a time cost of approximately $2d^2$ vector comparisons.

Ideal lattices. To make use of the cyclic structure of certain ideal lattices, the main idea of the ideal GaussSieve is that comparing $\boldsymbol{v}_{(s)}$ to $\boldsymbol{w}_{(s')}$ is the same as comparing $\boldsymbol{v}_{(s-s')}$ to $\boldsymbol{w}$ for any s, s': there exist shifts of $\boldsymbol{v}$ and $\boldsymbol{w}$ that are nearby iff there exists a shift of $\boldsymbol{v}$ that is nearby to $\boldsymbol{w}$. So we only store the two representative vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ in the system (storage cost of 2 vectors), and we only compare all d shifts of $\boldsymbol{v}$ to the single vector $\boldsymbol{w}$ (d comparisons). To make sure that also $\boldsymbol{v}$ ($\boldsymbol{w}$) and its own cyclic shifts are pairwise reduced, we further need $d/2$ ($d/2$) comparisons to compare $\boldsymbol{v}$ to $\boldsymbol{v}_{(s)}$ ($\boldsymbol{w}$ to $\boldsymbol{w}_{(s)}$) for $s = 1, \ldots, d/2$. In total, we therefore need $2d$ comparisons to reduce $\boldsymbol{v}, \boldsymbol{w}$ and all their cyclic shifts.

Overall, this shows that in cyclic and negacyclic lattices, the memory cost of the GaussSieve goes down by a factor d, and the number of inner products that we compute to make sure the list is pairwise reduced also goes down by a factor approximately d. Although only polynomial, a factor 100 speed-up and using 100 times less memory in dimension 100 can be very useful.

12.5.2 – Hashing shifted vectors is shifting hashes of vectors. To see how we can obtain similar improvements for the CPSieve, let us first look at the basic hash function $h(\boldsymbol{x}) = \operatorname{argmax}_i |x_i|$, where for simplicity we again omit the sign of the hash function, leading to d different hash values, and we let the output of a hash be a number in $\{1, \ldots, d\}$ rather than a vector $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d\}$. Suppose we have a cyclic lattice, and for some lattice vector $\boldsymbol{v}$ we have $h(\boldsymbol{v}) = i$ for some $i \in \{1, \ldots, d\}$. Due to the choice of the hash function, we know that if we shift the entries of $\boldsymbol{v}$ to the right by s positions to get $\boldsymbol{v}_{(s)}$, then the hash of this vector will increase by s as well, modulo d (where the modular computations are performed with offset 1):

$$h(\boldsymbol{v}_{(s)}) = [h(\boldsymbol{v}) + s] \bmod d. \tag{12.17}$$

As a result, given a vector $\boldsymbol{v}$ and its base hash value, we know exactly what the hash value of its rotations will be. We also know that $h(\boldsymbol{v}) = h(\boldsymbol{w})$ if and only if $h(\boldsymbol{v}_{(s)}) = h(\boldsymbol{w}_{(s)})$ for any s. For the basic hash function h, this property allows us to use a similar trick as in the ideal GaussSieve: we only store one representative of $\boldsymbol{w}$ in the hash tables, and for finding shifts of $\boldsymbol{v}$ and $\boldsymbol{w}$ which are nearby, we only compare all d shifts $\boldsymbol{v}_{(s)}$ to the vectors in the buckets $h(\boldsymbol{v}_{(s)})$. We are then guaranteed that if any pair of vectors $\boldsymbol{v}_{(s)}$ and $\boldsymbol{w}_{(s')}$

can be reduced and have the same hash value, we will encounter this reduction when we compare $\boldsymbol{v}_{(s-s')}$ and $\boldsymbol{w}$ as they will also have the same hash values.

12.5.3 – Ideal rerandomizations through circulant matrices. While this shows that the basic hash function h has this nice property that allows us to obtain the linear reductions in the time and space complexity similar to the ideal GaussSieve, to make this algorithm work we will need all hash functions from $\mathcal{H}$ to satisfy these properties. And once we apply a random (pseudo-)rotation A to a vector $\boldsymbol{v}$, we may lose the property described in (12.17):

$$h_A(\boldsymbol{v}_{(s)}) = h(A\boldsymbol{v}_{(s)}) \stackrel{?}{=} [h(A\boldsymbol{v}) + s] \bmod d = [h_A(\boldsymbol{v}) + s] \bmod d, \tag{12.18}$$

The second equality is crucial here, as without preserving the property that the hash of a shift of a vector equals the modular shift of the hash of a vector, it might be the case that there exists a pair of vectors $\boldsymbol{v}_{(s)}$ and $\boldsymbol{w}_{(s')}$ that is nearby *and has the same hash value*, while we will not compare $\boldsymbol{v}_{(s-s')}$ and $\boldsymbol{w}$ because they have different hash values. If that happens, then not all $2d$ shifts of both vectors are pairwise reduced, which implies that we will miss some of the potential reductions within our list. Then the list size inevitably goes up, and we may lose the factor d speed-up again.

To guarantee that the second equality in (12.18) is always an equality, we would like to make sure that $A\boldsymbol{v}_{(s)} = (A\boldsymbol{v})_{(s)}$, i.e., multiplying a shifted vector by a random Gaussian matrix A is the same as shifting the vector which has already been multiplied by A by the same number of positions. After all, in that case we would have

$$h_A(\boldsymbol{v}_{(s)}) = h(A\boldsymbol{v}_{(s)}) = h((A\boldsymbol{v})_{(s)}) = [h(A\boldsymbol{v}) + s] \bmod d = [h_A(\boldsymbol{v}) + s] \bmod d, \tag{12.19}$$

where the second equality follows from the condition $A\boldsymbol{v}_{(s)} = (A\boldsymbol{v})_{(s)}$ and the third equality follows from the property (12.17) of the base hash function h. So if we can guarantee that $A\boldsymbol{v}_{(s)} = (A\boldsymbol{v})_{(s)}$ for all $\boldsymbol{v}$ and s, then the entire hash family $\mathcal{H}$ satisfies the property we need to obtain a linear speed-up.

Now, it is not hard to see that the condition $A\boldsymbol{v}_{(s)} = (A\boldsymbol{v})_{(s)}$ for all $\boldsymbol{v} \in \mathbb{R}^d$ and $s \in \{1, \dots, d\}$ is equivalent to the condition that A is *circulant*; substituting $\boldsymbol{v} = \boldsymbol{e}_1$ and varying $s = 1, \dots, d$ tells us that $a_{i,j} = a_{1,[j-i+1] \bmod d}$ for all i and j. Note that if A is circulant, then also this condition is satisfied. In other words, for matrices A to satisfy these properties, we are free to choose the first row of A, and the i-th row of the matrix is then defined as the $(i-1)$-th cyclic shift of A.

So finally, the question becomes: can we simply impose the condition that this pseudo-rotation matrix A is circulant? While proving that the answer is yes or no seems difficult, experimentally the answer seems to be yes: by only generating the first row of each rerandomization matrix A at random from a standard Gaussian distribution, and then deriving the remaining entries of A from the first row, we obtain circulant matrices which appear to be as suitable for (pseudo-)rotations as fully random Gaussian matrices. The resulting circulant matrices on average appear to be as orthogonal as non-circulant ones, thus preserving relative norms and distances between vectors, and do not seem to perform worse in our experiments than non-circulant matrices.

Algorithm 12.1 Reducing a vector $\boldsymbol{v}$ with L in the ideal CrossPolytopeSieve

1: **for each** hash table T_i **do**
2: Compute the k base hash values $(H_1, \ldots, H_k) = (h_{i,1}(\boldsymbol{v}), \ldots, h_{i,k}(\boldsymbol{v}))$
3: **for each** cyclic shift $s = 0, \ldots, d-1$ **do**
4: Compute $\boldsymbol{v}_{(s)}$'s partial hash values $H_j^{(s)} = [H_j + s] \bmod d$ for $j = 1, \ldots, k$
5: Compute $\boldsymbol{v}_{(s)}$'s hash value $h_i(\boldsymbol{v}_{(s)}) = \sum_{j=1}^{k} (H_j^{(s)} - 1) d^{j-1} \in \{0, \ldots, d^k - 1\}$
6: **for each** $\boldsymbol{w} \in T_i[h_i(\boldsymbol{v}_{(s)})]$ **do**
7: Reduce $\boldsymbol{v}_{(s)}$ with $\boldsymbol{w}$
8: Reduce $\boldsymbol{w}$ with $\boldsymbol{v}_{(s)}$
9: **if** $\boldsymbol{w}$ has changed **then**
10: Remove $\boldsymbol{w}$ from the list L
11: Remove $\boldsymbol{w}$ from all t hash tables T_i
12: Add $\boldsymbol{w}$ to the stack S (unless $\boldsymbol{w} = \boldsymbol{0}$)
13: **if** $\boldsymbol{v}$ or one of its shifts has been modified **then**
14: Add the reduced (shift of) $\boldsymbol{v}$ to the stack S (unless $\boldsymbol{v} = \boldsymbol{0}$)
15: **else**
16: Add $\boldsymbol{v}$ to the list L
17: Add $\boldsymbol{v}$ to all t hash tables T_i to the buckets $T_i[h_i(\boldsymbol{v})]$

Remark 12.1. The angular/hyperplane hash functions of the HashSieve (Chapter 10), as well as the hypercone hash functions in the GaussSieve with hypercone LSH (Chapter 11) do not have the properties mentioned above, and although it may be possible to obtain the trivial decrease in the space complexity of a factor d, it seems impossible to achieve the same factor d time speed-up for these other methods.

Remark 12.2. By using circulant matrices, computing hashes of shifted vectors can simply be done by shifting the hash of the original vector. Also, one can compute the product of a circulant matrix with an arbitrary vector in $O(d \log d)$ time using Fast Fourier Transforms [GL96] instead of $O(d^2)$ time. However, the even faster random rotations described in [AIL+15] using Hadamard transforms cannot be used here, as we need A to be circulant to obtain this additional factor d speed-up.

12.5.4 – Modifying the reductions. Using these circulant matrices on ideal (cyclic) lattices, the reduction in Lines 4–13 of the GaussSieve (Algorithm 8.2) can be replaced by the steps described in Algorithm 12.1. It is described for the case where both $+\boldsymbol{e}_i$ and $-\boldsymbol{e}_i$ are mapped to the same hash value i, but it can easily be adjusted for the full cross-polytope hash family with $2d$ hash values.

NTRU lattices. The lattice basis of an NTRU encryption scheme [HPS98] can be described by a prime power d', a small power of two q, the ring $R = \mathbb{Z}_q[X]/(X^{d'} - 1)$, and two polynomials $f, g \in R$ with small coefficients, e.g. with coefficients in $\{-1, 0, 1\}$. We require that f is invertible in R and set $h = g/f \bmod q$. The public basis is then given

by d', q and h as the $d \times d$ matrix B (where $d = 2d'$) as follows:

$$B = \left(\begin{array}{cccc|ccccc} q & & & & & & & & \\ & q & & & & & 0 & & \\ & & \ddots & & & & & & \\ & & & q & & & & & \\ \hline h_0 & h_1 & \cdots & h_{d'-1} & 1 & & & & \\ h_{d'-1} & h_0 & \cdots & h_{d'-2} & & 1 & & & \\ \vdots & \vdots & \ddots & \vdots & & & \ddots & & \\ h_1 & h_2 & \cdots & h_0 & & & & 1 & \end{array}\right). \tag{12.20}$$

The rows of this matrix form the basis vectors, and the top-right block as well as the off-diagonal entries of the top-left and bottom-right blocks are all 0. Note that not only $(g \| f)$ but also all block-wise rotations $(g \cdot X^k \| f \cdot X^k)$ are short vectors in the lattice. More generally, each block of $d' = d/2$ entries of a lattice vector can be shifted (without minus sign) to obtain another valid lattice vector.

For these lattices we can apply the techniques described above, but as we only have $d/2$ shifts of a vector in d dimensions, the expected speed-ups and memory gains are not equal to the dimension, but only to half the dimension of the lattice we are investigating. In Algorithm 12.1, this means that we only consider cyclic shifts $s = 0, \ldots, d' - 1$, and a cyclic shift now corresponds to shifting both blocks of $\boldsymbol{v}$ so the right by s positions.

Power-of-two cyclotomic ideal lattices. Recall that power-of-two cyclotomic deal lattices are defined over the ring $\mathbb{Z}[X]/(X^d + 1)$ where d is a power of 2. These are negacyclic lattices, and so for any lattice vector $\boldsymbol{v}$ all its $2d$ shifts (with negative carry) are in the lattice as well, and $\boldsymbol{v}_{(d)} = -\boldsymbol{v}$. If we map both $\pm\boldsymbol{v}$ to the same hash value (i.e. we ignore the sign of the maximum value), then Algorithm 12.1 can be used for speeding up the reductions by a factor approximately d.

12.5.5 – Experiments for ideal lattices. For testing the performance of SVP algorithms on ideal lattices, and comparing it with non-ideal lattices, we focused on NTRU lattices where $d = 2d'$ and d' is prime, and on cyclotomic lattices where $d = 2^s$ is a power of 2, which can be generated with the ideal lattice challenge generator [PS15]. For the NTRU lattices we considered values $d = 38, 46, 58, 62, 74$, while for the cyclotomic lattices we restricted our experiments to only $d = 64$; for $d = 32$ the data will be unreliable as the algorithm terminates very quickly and the basis reduction routine sometimes already finds a shortest vector, while $d = 128$ is out of reach for our single-core proof-of-concept implementation; investigating the costs of solving the 128-dimensional ideal lattice challenge with the ideal CrossPolytopeSieve, as done in [BNvdP14, IKMT14], is left for future work.

The results of these experiments on ideal lattices are shown in Figure 12.2e in comparison to the random, non-ideal complexities of the CrossPolytopeSieve with $k = 2$. The costs for NTRU lattices are an order of magnitude lower than in the non-ideal case, and for cyclotomic lattices (where we get an extra factor 2 improvement in both the time and the space) we obtain yet another order of improvement over NTRU lattices.

CHAPTER 13

Hypercone locality-sensitive filtering

13.1 — Overview

Context. In Chapters 10 and 11 we considered applying existing techniques from the nearest neighbor literature to sieving: the celebrated hyperplane LSH method of Charikar [Cha02] (Chapter 10) and the spherical (hypercone) LSH method of Andoni–Indyk–Nguyễn–Razenshteyn [AINR14, AR15a] (Chapter 11). As the latter method is asymptotically optimal but less efficient in practice due to the large overhead of computing hashes, we asked whether we could achieve the same optimality as hypercone LSH but with the efficiency of hyperplane LSH. In Chapter 12 we saw that this is indeed possible, by improving upon the existing nearest neighbor search literature and constructing a new, efficient LSH method (cross-polytope LSH). As the asymptotic performance of cross-polytope LSH matches lower bounds of e.g. Dubiner [Dub10] and Andoni and Razenshteyn [AR15b], and the overhead of computing hashes is minimal, is using cross-polytope LSH the best we can do for finding nearby vectors in lattice sieving?

Low-density NNS. Looking closely at the fine print of various lower bounds on locality-sensitive hashing [AR15b, Dub10, MNP07, OWZ11, OWZ14], we see that all these results rely on the same assumption[1]: the probability of collision for nearby and distant vectors is assumed to be at least $2^{-o(d)}$. After all, if e.g. the probability of collision in one hash function for nearby vectors is only $2^{-\Theta(d)}$, then we need at least $t = 2^{\Theta(d)}$ hash tables to guarantee that nearby vectors are found through collisions. As the data set in NNS is commonly assumed to be of *low density*, i.e. of size only $n = 2^{o(d)}$, a number of hash tables $t \gg \mathrm{poly}(n)$ is clearly not going to be useful here.

High-density NNS. In lattice sieving however, note that we commonly have a list of lattice vectors of size $n = (4/3)^{d/2+o(d)} \approx 2^{0.21d+o(d)}$ that we are searching in, and so in sieving we are in the *high-density regime* of $n = 2^{\Theta(d)}$. In that case, a collision probability $2^{-\Theta(d)}$ and a number of hash tables $t = 2^{\Theta(d)}$ is not unthinkable: whereas in the previous chapters the collision probabilities for a single hash function were $2^{-o(d)}$, the number of hash tables we used for sieving was always $t = 2^{\Theta(d)}$. Does the same bound $\rho \geqslant 1/(2c^2 - 1)$ on LSH for approximation factor c also hold in high-density regimes? Or can we design nearest neighbor methods specifically for the case of $n = 2^{\Theta(d)}$ that perform even better in this regime, which may be useful for lattice sieving?

*This chapter is based on results from [BDGL16].

[1] Strictly speaking Dubiner [Dub10] does not state this condition, but a complete proof of the statements in [Dub10] still has not been published, and the same condition may be necessary to complete the proof.

Hypercone locality-sensitive filtering. In this chapter we introduce the concept of *locality-sensitive filtering* (LSF), which in short corresponds to locality-sensitive hashing where only few vectors are actually assigned to a bucket for one hash function. We analyze its properties, its relation with LSH, and how this potentially leads to an improved performance over LSH and smaller exponents $\rho < 1/(2c^2-1)$, given access to a certain decoding oracle. To instantiate these filters, we propose to use filters defined by hypercones similar to hypercone LSH, but with a larger parameter α than in Chapter 11. We highlight similarities and differences with spherical/hypercone LSH and show how this potentially leads to an improved performance over what is possible with locality-sensitive hashing, given access to such a decoding oracle.

List-decodable random product codes. These results depend on the existence of an efficient algorithm for finding the relevant hypercones that a vector lies in. To instantiate this oracle, we propose to use filters where the random filter vectors are taken from certain structured, concatenated codes over the sphere (rather than sampled at random and independently) such that we can compute a vector's relevant filters with minimal overhead using a technique very similar in spirit to list decoding. A crucial issue is to prove that these filters behave equally well as when all the filters were chosen independently, and by carefully selecting the parameters, we show that this is indeed the case.

Application to lattice sieving. We finally apply this new nearest-neighbor method for high-density settings to lattice sieving, and show that we obtain an asymptotic complexity for solving SVP of only $2^{0.292d+o(d)}$, improving upon the asymptotic time complexity of $2^{0.298d+o(d)}$ using hypercone LSH (Chapter 11) or cross-polytope LSH (Chapter 12). Figure 13.1 illustrates the asymptotic time-memory trade-off for this algorithm, showing that especially the trade-off for smaller amounts of memory is significantly better than with LSH methods. Experimental results further show that the improvement is relevant in moderate dimensions as well, as the subexponential overhead is small.

Outline. In Section 13.2 we first describe the locality-sensitive filtering framework, under the assumption of an efficient decoding oracle, and without specifying the exact filters that we will use. Section 13.3 continues with a specific instantiation of these filters using hypercones of smaller size than in hypercone LSH, and shows the potential improvements with this method. Section 13.3 also shows how to design these filters so that decoding can be done efficiently, without diminishing the asymptotic performance of this scheme. Sections 13.4 and 13.5 finally show the (theoretical and practical) effects of applying this method to sieving algorithms.

13.2 — The locality-sensitive filtering (LSF) framework

13.2.1 – Locality-sensitive filters. Instead of locality-sensitive (hash) functions considered in previous chapters, here we will consider locality-sensitive filters, which map an input list L to an output list $L' \subseteq L$. Vectors $\mathbf{v} \in L'$ are said to *survive* this filter. Ideally we would like to use filters with the property that only nearby vectors will survive the same filter, so that if we apply a (sequence of) filter(s) to an input set L, then the output list L' only contains vectors which are nearby in space. One output list is best compared with one hash bucket in LSH, where in LSH we make the additional assumption that each vector is assigned to exactly one bucket in a hash table (not accounting for probing).

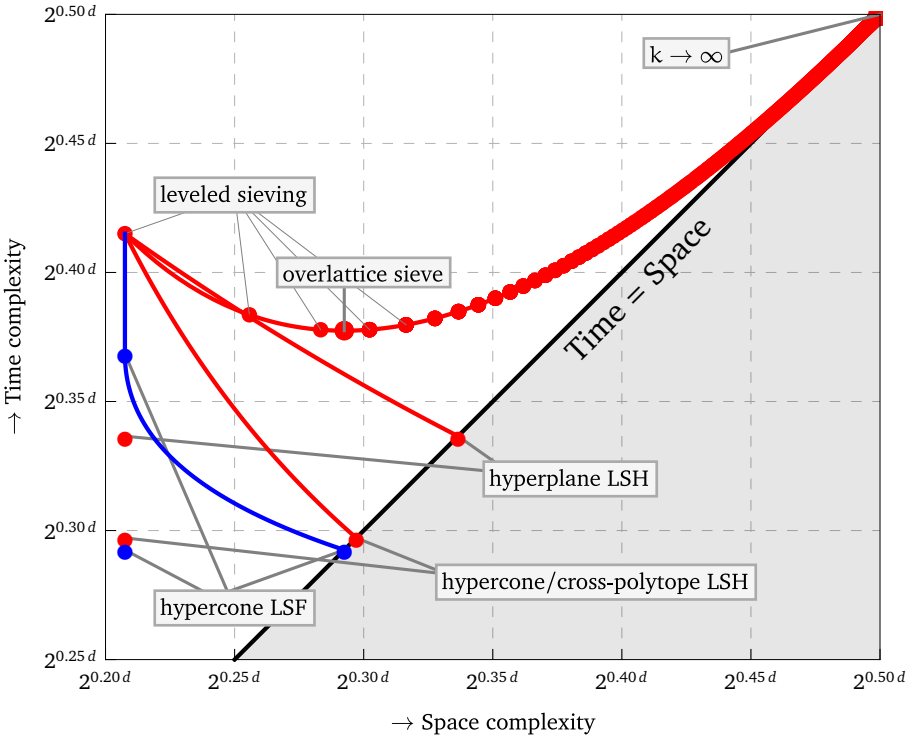


Figure 13.1: The asymptotic time-memory trade-off for hypercone filtering (blue) compared to the asymptotic complexities of other sieving algorithms considered in previous chapters (red). The blue curve and the points on this curve apply to both the GaussSieve and the Nguyễn–Vidick sieve, while the bottom-left blue point and the two red points above it only apply to the Nguyễn–Vidick sieve.

To solve the nearest neighbor problem with these filters, we propose the following method. Given a distribution $\mathcal{F}$ of filters, we draw $t \cdot k$ filters $f_{i,j} \sim \mathcal{F}$, and combine k at a time to build t filters f_i, where $\boldsymbol{w}$ passes through the filter f_i if it passes through all filters $f_{i,j}$ for $j = 1, \dots k$. Then, given the list L, we build t different filtered buckets $B_{f_1}, \dots, B_{f_t}$, where a vector $\boldsymbol{w} \in L$ is inserted into the bucket B_{f_i} iff $\boldsymbol{w}$ survives filter f_i. Finally, given a query vector $\boldsymbol{v}$, we check which of the filters it passes through, gather all the candidate vectors that pass through at least one of the same combined filters f_i, and search this set of candidates for a nearest neighbor. With a suitable filter family $\mathcal{F}$ and parameters k and t, this may allow us to solve (approximate) nearest-neighbor searching.

13.2.2 – Performance of LSF. For analyzing the performance of LSF, we assume that we have an efficient oracle $\mathcal{O}$ which identifies the filters that a vector $\boldsymbol{v}$ passes through (the *relevant* filters for $\boldsymbol{v}$) in time $\tilde{O}(F_{\boldsymbol{v}})$, where $F_{\boldsymbol{v}}$ counts $\boldsymbol{v}$'s relevant filters. This assumption is crucial, as without this oracle finding the relevant filters may take time $\tilde{O}(t)$, and we may not obtain an improved performance over LSH. Assuming the filter distribution $\mathcal{F}$ is spherically symmetric, similar to collision probabilities in LSH we write

$$p(\theta) = \mathbb{P}_{f\sim\mathcal{F}}[\boldsymbol{v}, \boldsymbol{w} \in B_f \mid \theta(\boldsymbol{v}, \boldsymbol{w}) = \theta], \qquad p_0 = \mathbb{P}_{f\sim\mathcal{F}}[\boldsymbol{v} \in B_f]. \tag{13.1}$$

Here p_0 may be informally thought of as $\lim_{\theta\to 0} p(\theta)$, i.e., the probability that $\boldsymbol{v}$ collides with itself in a filter. Now, $\boldsymbol{v}$ survives a sequence of k filters with probability p_0^k, so $F_{\boldsymbol{v}} = O(t \cdot p_0^k)$. On the other hand, a vector $\boldsymbol{w}$ at angle θ with $\boldsymbol{v}$ collides with $\boldsymbol{v}$ in a k-concatenated filter with probability $p(\theta)^k$. If for simplicity we assume that n vectors lie at an angle θ_2 with our target vector $\boldsymbol{v}$ and there is one near neighbor at angular distance θ_1, then the costs of processing a query with an efficient oracle are $Q = t \cdot p_0^k + t \cdot p(\theta_2)^k \cdot n$, with the first term counting the relevant filters and the second term counting the collisions of distant vectors.

Next, to guarantee that we will find a nearby vector at angle θ_1 with probability $1-\varepsilon$ if it exists, for some constant $\varepsilon \in (0,1)$, we need $1-(1-p(\theta_1)^k)^t = O(t\cdot p(\theta_1)^k) \geqslant 1-\varepsilon$, or $t = O(1/p(\theta_1)^k)$. We further want to minimize the total cost Q of processing a query, which corresponds to balancing the two contributions to Q; larger k and t lead to more selective filtering and fewer comparisons, but increase the cost of finding the relevant filters. Equating the two terms translates to $p_0^k = p(\theta_2)^k \cdot n$ up to subexponential terms. Solving for k, we obtain expressions for k and t, which in turn can be substituted into Q to find the best parameters for LSF.

Theorem 13.1. *Let $\mathcal{F}$ be an LSF family with collision probability function p, and let $\mathcal{O}$ be an oracle computing a vector $\boldsymbol{v}$'s relevant filters in time $\tilde{O}(F_{\boldsymbol{v}})$. Let ρ, ρ_t be defined as*

$$\rho = \frac{\log p_0 - \log p(\theta_1)}{\log p_0 - \log p(\theta_2)}, \qquad \rho_t = \frac{-\log p(\theta_1)}{\log p_0 - \log p(\theta_2)}. \tag{13.2}$$

Then we can solve approximate NNS with parameters θ_1 and θ_2 in time $\tilde{O}(n^\rho)$ and space $\tilde{O}(n^{1+\rho})$ with preprocessing time $\tilde{O}(n^{1+\rho})$, with parameters

$$k = \frac{\log n}{\log p_0 - \log p(\theta_2)}, \qquad t = O(n^{\rho_t}). \tag{13.3}$$

Proof. Using the parameters stated above, we can expand various quantities as follows:

$$n = \exp\left[\frac{\log n}{\log p_0 - \log p(\theta_2)} \cdot (\ln p_0 - \ln p(\theta_2))\right], \tag{13.4}$$

$$t = \exp\left[\frac{\log n}{\log p_0 - \log p(\theta_2)} \cdot (-\ln p(\theta_1))\right], \tag{13.5}$$

$$p^k = \exp\left[\frac{\log n}{\log p_0 - \log p(\theta_2)} \cdot (\ln p)\right]. \qquad (p \in \{p_0, p(\theta_1), p(\theta_2)\}) \tag{13.6}$$

Combining these expressions, we see that the number of relevant filters for one query is asymptotically given by $t \cdot p_0^k = n^\rho$; the nearby collision probability is of the order $t\cdot p(\theta_1)^k = O(1)$; and the number of distant vector collisions is approximately $t\cdot p(\theta_2)^k \cdot n = n^\rho$. This means the query is correctly answered with high probability, with query time $\tilde{O}(n^\rho)$. As each vector is assigned to $\tilde{O}(t \cdot p_0^k) = \tilde{O}(n^\rho)$ filters, in total the n vectors require a storage of $\tilde{O}(n^{1+\rho})$ entries in the filter buckets. Preprocessing takes time proportional to the storage, as we assume our decoding oracle is efficient. □

13.2.3 – Relation with locality-sensitive hashing. Notice the similarities and differences between Theorem 13.1 and Lemma 10.2 on the parameters for locality-sensitive

hashing, which we restate below (slightly differently) for comparison:

$$\rho' = \frac{\log 1 - \log p(\theta_1)}{\log 1 - \log p(\theta_2)}, \qquad k' = \frac{\log n}{\log 1 - \log p(\theta_2)}, \qquad t' = O(n^{\rho}). \tag{13.7}$$

Looking at the two exponents ρ and ρ_t for LSF and the exponent ρ' for LSH, we can establish the following simple but powerful inequalities from the fact that $p_0 \leqslant 1$:

$$\rho \leqslant \rho' \leqslant \rho_t. \tag{13.8}$$

These two inequalities say that the exponent ρ for solving NNS with filtering is potentially smaller (better) than for LSH with the same hash/filter family ($\rho \leqslant \rho'$). The reason we obtain a better exponent lies in the decoding oracle; without a decoding oracle, the naive cost for finding relevant vectors would be $\tilde{O}(t) = \tilde{O}(n^{\rho_t})$, and we would achieve a worse exponent than with LSH ($\rho_t \geqslant \rho'$).

Note that in LSH, the function p denotes collision probabilities in a hash function, and as each vector always has one output value (i.e. is assigned to one bucket) we have $\lim_{\theta \to 0} p(0) = 1$: a vector always collides with itself in LSH. For LSF we only get a collision of $\boldsymbol{v}$ with itself if $\boldsymbol{v}$ survives filter f. So generally $p_0 < 1$ for LSF, while informally $p_0 = 1$ in LSH, which leads to the different exponents above. Observe that substituting $p_0 = 1$ in the exponents for LSF unsurprisingly leads to $\rho = \rho' = \rho_t$.

Remark 13.1. The above theorem is given only as an illustration of our approach, as we do not know of any implementation of such an oracle when the filters are chosen independently and uniformly at random from $\mathcal{F}$. In Section 13.3.4 we do provide such an oracle for filters that are more structured, yet still ensuring the proper collision probabilities.

13.3 — Hypercone locality-sensitive filtering

For analyzing the hypercone filters which we will introduce in Section 13.3.2, we first introduce some terminology and preliminary lemmas regarding high-dimensional geometric objects on the unit sphere.

13.3.1 – Geometric objects on the sphere. Let μ be the canonical Lebesgue measure over $\mathbb{R}^d$, and let $S^{d-1} = \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| = 1\}$ again denote the unit sphere with respect to the Euclidean norm in $\mathbb{R}^d$. As usual, let $\langle \cdot, \cdot \rangle$ denote the standard Euclidean inner product. We denote half-spaces with normal vector $\boldsymbol{v}$ and parameter $\alpha \in [0, 1]$ by $H_{\boldsymbol{v},\alpha} = \{\boldsymbol{x} \in \mathbb{R}^d : \langle \boldsymbol{v}, \boldsymbol{x} \rangle \geqslant \alpha\}$. For $\boldsymbol{v}, \boldsymbol{w} \in S^{d-1}$ such that $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \cos\theta$ and $\alpha, \beta \in (0, 1)$ we further introduce spherical caps and wedges as follows:

$$C_{\boldsymbol{v},\alpha} = H_{\boldsymbol{v},\alpha} \cap S^{d-1}, \qquad W_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta} = H_{\boldsymbol{v},\alpha} \cap H_{\boldsymbol{w},\beta} \cap S^{d-1}. \tag{13.9}$$

The following two lemmas estimate the volumes of spherical caps and wedges for large d. Lemma 13.2 is elementary, while for Lemma 13.4 we make use of an additional lemma below (Lemma 13.3).

Lemma 13.2. *For any $\alpha \in (0, 1)$ and $\boldsymbol{v} \in S^{d-1}$, we have*

$$C_d(\alpha) := \frac{\mu(C_{\boldsymbol{v},\alpha})}{\mu(S^{d-1})} = \mathrm{poly}(d) \cdot \left(\sqrt{1 - \alpha^2}\right)^d. \tag{13.10}$$

Proof. Using [MV10b, Lemma 4.1] with $R = 1$ and $h = 1-\alpha$ we obtain the lower bound, while an upper bound can be found with similar techniques (see [NV08, Lemma 3.4]). □

Lemma 13.3. *Let $\gamma \in (0,1)$, and let g be a continuous function on $(\gamma,1)$ such that $g(r) = h(r)(r-\gamma)^\kappa$ for some positive real number $\kappa > 0$ where $h(r)$ is continuous over $(\gamma,1)$, has limit $h(r) \to 1$ as $r \to \gamma$, and is bounded by $H > 0$. Then*

$$I(\gamma) = \int_{r=\gamma}^{1} g(r)\left(\sqrt{1-r^2}\right)^d dr \overset{(d\to\infty)}{\sim} \Gamma(\kappa+1)\left(\frac{1-\gamma^2}{d\gamma}\right)^{\kappa+1} \cdot \left(\sqrt{1-\gamma^2}\right)^d. \tag{13.11}$$

In other words, assuming that γ, κ are constant, we have $I(\gamma) = \mathrm{poly}(d) \cdot \left(\sqrt{1-\gamma^2}\right)^d$.

Proof. Using the change of variable $q = d(r-\gamma)$ or equivalently $r = \gamma + q/d$, we get

$$I(\gamma) = \int_{q=0}^{d(1-\gamma)} h\left(\gamma + \frac{q}{d}\right)\left(\frac{q}{d}\right)^\kappa \left(\sqrt{1-\gamma^2-\frac{2\gamma q}{d}-\frac{q^2}{d^2}}\right)^d \frac{1}{d}\,dq. \tag{13.12}$$

Slightly rewriting the integral, and pulling out some constant factors, we obtain the following expression, where $\mathbb{1}\{P\} = 1$ if P holds and $\mathbb{1}\{P\} = 0$ otherwise.

$$I(\gamma) = \frac{\left(\sqrt{1-\gamma^2}\right)^d}{d^{\kappa+1}}\Bigg\{\int_{q=0}^{\infty} \mathbb{1}\{q \leqslant (1-\gamma)d\}\, h\left(\gamma+\frac{q}{d}\right) q^\kappa \tag{13.13}$$

$$\times \left(\sqrt{1-\frac{2\gamma q}{(1-\gamma^2)d}-\frac{q^2}{(1-\gamma^2)d^2}}\right)^d dq\Bigg\}. \tag{13.14}$$

For large d, the term inside the integral converges to $h(\gamma)q^\kappa \exp(-\gamma q/(1-\gamma^2))$ for all $q \geqslant 0$, and is bounded by $Hq^\kappa \exp(-\gamma q/(1-\gamma^2))$, which is integrable over $\mathbb{R}^+$. By the dominated convergence theorem, the integral converges to $\int_0^\infty q^\kappa \exp(-\gamma q/(1-\gamma^2))$, which is a standard Laplace integral [AS72, Equation (29.3.7)] equal to the given expression. This concludes the proof. □

Lemma 13.4. *Let $\alpha, \beta \in (0,1)$ and $\theta \in (0, \frac{\pi}{2})$ such that $\min\{\frac{\alpha}{\beta}, \frac{\beta}{\alpha}\} \geqslant \cos\theta$, and let $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{S}^{d-1}$ with $\langle \boldsymbol{v}, \boldsymbol{w}\rangle = \cos\theta$. Then we have*

$$\mathcal{W}_d(\alpha,\beta,\theta) := \frac{\mu(\mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta})}{\mu(\mathcal{S}^{d-1})} = \mathrm{poly}(d)\cdot\left(\sqrt{1-\gamma^2}\right)^d, \tag{13.15}$$

where γ satisfies $\gamma^2 = (\alpha^2+\beta^2-2\alpha\beta\cos\theta)/(\sin^2\theta)$. For $\alpha = \beta$, this simplifies to

$$\mathcal{W}_d(\alpha,\alpha,\theta) = \mathrm{poly}(d)\cdot\left(\sqrt{1-\frac{2\alpha^2}{1+\cos\theta}}\right)^d. \tag{13.16}$$

Proof. Let us compute the volume of a wedge with parameters α, β, θ, which is the volume of the intersection of the spherical caps of height $1-\alpha$ centered at $\boldsymbol{v} = \boldsymbol{e}_1$ (defined by $\langle \boldsymbol{v}, \boldsymbol{x} \rangle \geqslant \alpha$) and of height $1-\beta$ centered at $\boldsymbol{w} = \boldsymbol{e}_1 \cos\theta + \boldsymbol{e}_2 \sin\theta$ (i.e. $\langle \boldsymbol{w}, \boldsymbol{x} \rangle \geqslant \beta$).

Let us denote the orthogonal projection from the unit sphere onto the plane defined by the three points $\boldsymbol{0}, \boldsymbol{v}, \boldsymbol{w}$ by the function f. For any measurable subset U of the two-dimensional circle, the surface of $f^{-1}(U)$ in the unit sphere has the following volume:

$$\int_{(x,y)\in U} \frac{\mu(\mathcal{S}^{d-3})}{\mu(\mathcal{S}^{d-1})} \left(\sqrt{1-x^2-y^2}\right)^{d-4} dx\, dy. \tag{13.17}$$

Alternatively, if U is described in terms of radial coordinates r and ϕ:

$$\int_{(r,\phi)\in U} \frac{\mu(\mathcal{S}^{d-3})}{\mu(\mathcal{S}^{d-1})} \left(\sqrt{1-r^2}\right)^{d-4} r\, dr\, d\phi. \tag{13.18}$$

For all $r \in [0,1]$, let us write $g(r) = \int_{\phi:(r,\phi)\in U} d\phi \in [0, 2\pi]$. If U is a convex set, then $g(r)$ is a continuous function from $[0,1]$ to $[0,2\pi]$, and we can make use of Lemma 13.3.

In the case of the wedge, the set U is a rounded triangle, defined by the three inequalities $\langle \boldsymbol{v}, \boldsymbol{x} \rangle \geqslant \alpha$, $\langle \boldsymbol{w}, \boldsymbol{x} \rangle \geqslant \beta$, and $\|\boldsymbol{x}\| \leqslant 1$. The point of smallest norm in this rounded triangle is the vector $\boldsymbol{c}$ satisfying $\langle \boldsymbol{v}, \boldsymbol{c} \rangle = \alpha$ and $\langle \boldsymbol{w}, \boldsymbol{c} \rangle = \beta$, and its norm is equal to the given expression for γ. In this case, the function $g(r)$ is null over $[0,\gamma]$, continuous and increasing over $[\gamma, 1]$, and scales as $\Theta(r-\gamma)$ (i.e. $\kappa = 1$) when r is close to γ. Thus, applying Lemma 13.3 with $\kappa = 1$, the volume of the wedge is proportional to $d^{\Theta(1)}(\sqrt{1-\gamma^2})^d$ for large d. □

In the lemma above, the condition $\min\{\frac{\alpha}{\beta}, \frac{\beta}{\alpha}\} \geqslant \cos\theta$ is equivalent to the condition $\gamma^2 \geqslant \max\{\alpha^2, \beta^2\}$, which guarantees that the point $\boldsymbol{c}$ mentioned in the proof above is indeed the closest point to the origin. If e.g. $\beta > \alpha\cos\theta$ then the point of smallest norm in the resulting triangle has norm β and is not a corner of the triangle. The lemma can be extended to arbitrary α and β by carefully observing which point in the projection has the smallest norm, but for our purposes the above statement of the lemma suffices.

Note that the main difference between a wedge and a spherical cap is that for a cap of parameter γ, the function $g(r)$ would be proportional to $\Theta(\sqrt{r-\gamma})$ corresponding to $\kappa = \frac{1}{2}$, so the volume of the cap is proportional to $(\sqrt{1-\gamma^2})^d/\sqrt{d}$ for large d. Asymptotically, up to factors polynomial in d, a spherical cap with parameter γ and a wedge with parameters α, β, θ have the same volume.

13.3.2 – Spherical and hypercone filters. We will instantiate the concept of locality-sensitive filtering with the following spherical cap LSF distribution $\mathcal{F}$. A filter is constructed by drawing a random vector $\boldsymbol{s} \in \mathcal{S}^{d-1}$, and a vector $\boldsymbol{w} \in \mathcal{S}^{d-1}$ is inserted into the bucket corresponding to this filter if it satisfies $\langle \boldsymbol{w}, \boldsymbol{s} \rangle \geqslant \beta$. In other words, a vector $\boldsymbol{w}$ is inserted in the filter bucket if it lies in the spherical cap centered at $\boldsymbol{s}$ of height $1-\beta$. Similar to spherical and hypercone LSH, we can trivially extend this notion to all of $\mathbb{R}^d$ to form hypercone filters, by saying a vector $\boldsymbol{w} \in \mathbb{R}^d$ is inserted into this filter if $\langle \boldsymbol{w}, \boldsymbol{s} \rangle \geqslant \beta \cdot \|\boldsymbol{w}\|$, or alternatively, $\boldsymbol{w}$ lies in the hypercone defined by the spherical cap around $\boldsymbol{s}$ of height $1-\beta$.

Then, after inserting all vectors in the corresponding filter buckets, a query $\boldsymbol{v} \in \mathbb{R}^d$ is answered by recovering all those filters with filter vectors $\boldsymbol{s}$ satisfying $\langle \boldsymbol{v}, \boldsymbol{s} \rangle \geqslant \alpha \cdot \|\boldsymbol{v}\|$

(note the asymmetry between α and β), and checking these filters for nearby vectors. For searching, we thus use the parameter α for defining which filters are relevant, while for insertions we use the parameter β. We will see later that these parameters can be tuned to obtain a trade-off between the time and the memory: $\alpha = \beta$ is optimal for minimizing the query time complexity, while $\alpha < \beta$ can be used for more refined searching and less refined insertions in the database, so that less memory is used to store vectors in filters, and more time is spent on answering queries.

13.3.3 – Collision probabilities for $\alpha = \beta$. Next, we observe that when querying a vector $\boldsymbol{v}$ and encountering a relevant filter for $\boldsymbol{v}$, we find the vector $\boldsymbol{w}$ in the same bucket if and only if the corresponding filter vector $\mathbf{s}$ lies in the two spherical caps defined by $\boldsymbol{v}, \alpha$ and $\boldsymbol{w}, \beta$. Equivalently, $\boldsymbol{v}$ and $\boldsymbol{w}$ collide iff $\mathbf{s}$ lies in the wedge $\mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta}$. As these vectors $\mathbf{s}$ are drawn uniformly at random from $\mathcal{S}^{d-1}$, the probability of a collision in one filter is equal to $\mathcal{W}(\alpha, \beta, \theta)$ if $\boldsymbol{v}, \boldsymbol{w}$ have pairwise angle θ. By Lemma 13.4 we therefore obtain the following filter collision probabilities for the case of $\alpha = \beta$:

$$p(\theta) = \mathcal{W}_d(\alpha, \alpha, \theta)^{-1} = \exp\left[\frac{d}{2} \ln\left(1 - \frac{2\alpha^2}{1+\cos\theta}\right)(1 + o(1))\right]. \tag{13.19}$$

For an angle $\theta = 0$, note that the collision probability corresponds to the probability that $\boldsymbol{v}$ collides with itself in a given filter $\mathbf{s}$. The probability that $\boldsymbol{v}$ is both added to a filter and is found with a search over all filters is given by $\min\{\mathcal{C}_d(\alpha), \mathcal{C}_d(\beta)\}$, while if we are interested in $\boldsymbol{v}$'s relevant filters for searching, we find a probability of having to check this filter for nearby vectors of $\mathcal{C}_d(\alpha)$. The quantity p_0 mentioned earlier corresponds to which filters are queried for finding nearby vectors, so the right interpretation for p_0 is

$$p_0 = \mathcal{C}_d(\alpha) = \exp\left[\frac{d}{2} \ln\left(1 - \alpha^2\right)(1 + o(1))\right]. \tag{13.20}$$

For $\alpha = \beta$ this corresponds to $p_0 = \lim_{\theta\to 0} p(\theta)$ up to subexponential terms.

If we suppose we have an efficient oracle for determining a vector's relevant filters, then by Theorem 13.1 we obtain a performance parameter ρ in terms of $\alpha, \theta_1, \theta_2$ of

$$\rho = \frac{\log\left(1-\alpha^2\right) - \log\left(1 - \alpha^2\left[\frac{2}{1+\cos\theta_1}\right]\right)}{\log\left(1-\alpha^2\right) - \log\left(1 - \alpha^2\left[\frac{2}{1+\cos\theta_2}\right]\right)}(1 + o(1)). \tag{13.21}$$

Notice that a Taylor series expansion of ρ for $d \to \infty$ and $\alpha \approx 0$ gives us

$$\rho \overset{(d\to\infty)}{\longrightarrow} \frac{\tan^2(\theta_1/2)}{\tan^2(\theta_2/2)} - \alpha^2\left(\frac{(1-\cos\theta_1)(\cos\theta_1 - \cos\theta_2)}{(1+\cos\theta_1)^2(1-\cos\theta_2)}\right) + O(\alpha^4). \tag{13.22}$$

For $\alpha \to 0$, this is equivalent to the exponent ρ of spherical and hypercone LSH of Chapter 11 and of cross-polytope LSH considered in Chapter 12. In other words, for small α the performance of hypercone filtering (provided an oracle $\mathcal{O}$ exists) is equivalent to the performance of spherical, hypercone, and cross-polytope LSH. On the other hand, observe that if $\alpha > 0$, then ρ will become smaller than for each of these LSH methods, and hypercone LSF may be asymptotically superior to all these methods for large d.

Optimizing α and setting $k = 1$. An intrinsic lower bound on the number of filters k that we need to use sequentially for one combined filter is given by $k \geqslant 1$, which translates to a bound on α as follows:

$$k = \frac{\log n}{\log p_0 - \log p(\theta_2)} \geqslant 1 \quad \Longrightarrow \quad \alpha \leqslant \alpha_0 = \sqrt{1 + \frac{n^{2/d}(\cos\theta_2 - 1)}{2n^{2/d} - \cos\theta_2 - 1}}. \tag{13.23}$$

Depending on n, θ_2, as well as on the existence of efficient decoding oracles for given α, this limits which values α can be used. We further observe that ρ is decreasing in α, which implies that one should always choose α to be as large as possible. If we assume that our decoding oracle is not limited by certain values of α, this suggests taking $\alpha = \alpha_0$ is optimal, and we should always set $k = 1$. So in that case, we should always only use only one consecutive filter for each of the t combined filters.

Note that the upper bound α_0 is decreasing with $n^{2/d}$, and so high-density settings with $n = \exp(\kappa d)$ for large densities κ are easier to solve than low-density cases, which matches our intuition. For the extreme high-density setting of $\kappa \to \infty$ we further obtain $\alpha_0 \to \frac{1}{2}\sqrt{2} \approx 0.71$, while for $\kappa \to 0$ the upper bound on α converges to 0. This again illustrates that for low-density settings, we cannot use values α significantly larger than 0, and we already saw that as $\alpha \to 0$ we approach the performance of spherical LSH. We only obtain better results for high-density settings $\kappa > 0$, for which lower bounds on LSH do not apply and so it may be possible to do better even with LSH methods.

The exponent ρ for distances $\sqrt{2}$ and $\frac{1}{c}\sqrt{2}$.. For general θ_1, θ_2, n, we now have a recipe to choose our parameters α, t, ρ and $k = 1$. To study the performance of hypercone LSF and to compare it with other results, let us focus on the *random* case of [AR15a], where $\theta_1 = \arccos(1 - 1/c^2)$ and $\theta_2 = \frac{\pi}{2}$, so that nearby vectors are a factor c closer to the target vector than distant, orthogonal vectors. In that case we obtain $\alpha = \alpha_0 = \sqrt{(\exp(2\kappa) - 1)/(2\exp(2\kappa) - 1)}$ as our optimal value, which in turn can be substituted into the expression for ρ. Figure 13.2 illustrates the resulting values of ρ for different approximation factors c and densities $\kappa = (\ln n)/d$.

Performing a Taylor series expansion for ρ for small κ (and fixed $c > 1$), we obtain

$$\rho \overset{(d\to\infty)}{\longrightarrow} \frac{1}{2c^2 - 1} - \left(1 - \frac{1}{2c^2 - 1}\right)\kappa + O(\kappa^2). \qquad (\kappa \to 0) \tag{13.24}$$

Again, we see the standard LSH lower bound appear as the leading term, and the first order term is negative, leading to smaller (better) values of ρ as soon as κ is a bit larger than 0. For large approximation factors c this implies that we obtain $\rho \sim 1/(2c^2)$ for small κ. Alternatively, if we look at high-density settings, then for arbitrary c and large κ we get

$$\rho \overset{(d\to\infty)}{\longrightarrow} \frac{1}{2\kappa}\ln\left(1 + \frac{1}{2c^2 - 2}\right) + O\left(\frac{1}{\kappa^2}\right). \qquad (\kappa \to \infty) \tag{13.25}$$

In this case, for large approximation factors c and large densities κ we get $\rho \sim 1/(4\kappa c^2)$ with lower order terms disappearing as $c, \kappa, d \to \infty$.

13.3.4 – List-decodable random product codes. To build an oracle that is able to efficiently determine the set of relevant filters for a given vector in the context of hypercone LSF, we will modify the distribution of filters; rather than sampling all of the filters

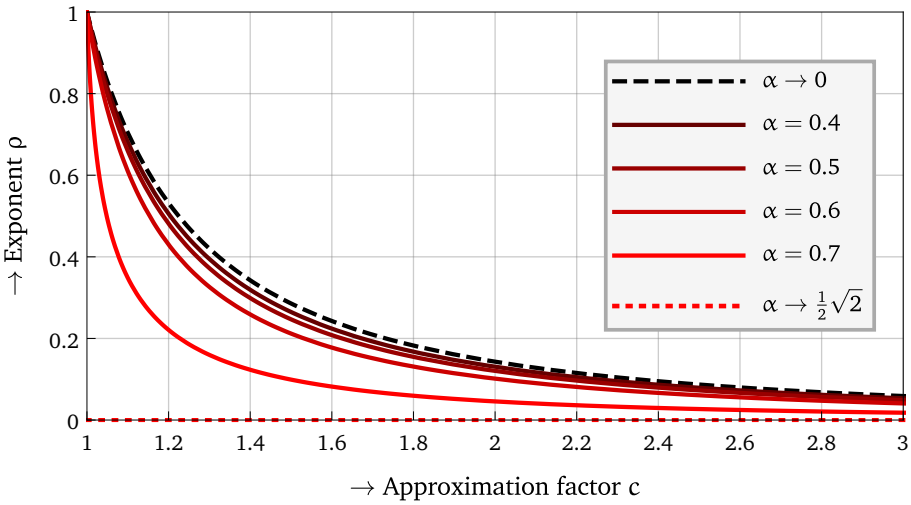


Figure 13.2: The performance parameter ρ for hypercone LSF plotted against the approximation factor c, in the *random* setting [AR15a] and for asymptotically large d. For $\kappa \to 0$ we have $\alpha \leqslant \alpha_0 \to 0$ and $\rho \to 1/(2c^2-1)$, while for $\kappa \to \infty$ the upper bound on α converges as $\alpha \leqslant \alpha_0 \to \frac{1}{2}\sqrt{2}$ so that $\rho \to 0$.

independently and uniformly at random from the sphere, we will use a structured code C which determines which filters we use, and which admits a fast decoding algorithm for finding the relevant vectors.

The idea behind choosing this code is that we split the d dimensions into m blocks of size $d' = d/m$, where we assume d', m are both integral. Then, instead of sampling t filter vectors $C = \{\mathbf{s}_1, \ldots, \mathbf{s}_t\} \subset \mathcal{S}^{d-1}$ at random[2], for each of the m blocks we sample $t' = t^{1/m}$ vectors $C_j = \{\mathbf{s}_{j,1}, \ldots, \mathbf{s}_{j,t'}\} \subset \mathcal{S}^{d'-1}$ at random. To form the complete set of filters, we then take the concatenation code of these m subcodes, by defining filter vectors $\mathbf{s} = (\mathbf{s}_{1,i_1}, \mathbf{s}_{2,i_2}, \ldots, \mathbf{s}_{m,i_m}) \in \mathcal{S}^{d-1}$ for $i_1, \ldots, i_m \in \{1, \ldots, t'\}$ and $\mathbf{s}_{j,i_j} \in C_j$. Formally, this leads to the following definition of random product codes.

Definition 13.5. *The distribution* $\mathrm{RPC}(d, m, t)$ *is defined as the distribution of* $C = C_1 \times \cdots \times C_m$*, where the codes* $C_i \subset \sqrt{1/m} \cdot \mathcal{S}^{b-1}$ *for* $i = 1, \ldots, m$ *are sets of* $t' = t^{1/m}$ *random, independent, uniform vectors over the sphere* $\sqrt{1/m} \cdot \mathcal{S}^{b-1}$*. A random product code* $C \leftarrow \mathrm{RPC}(d, m, t)$ *has* $(t^{1/m})^m = t$ *elements.*

Intuitively, choosing m large (using many blocks) means that the code becomes less similar to a uniformly random code, but large m also means that we can decode more efficiently, as we will see later. This means that we should not take m too small (decodability) and not too large (randomness). A suitable choice is to take m to be polylogarithmic in d (i.e. not even polynomial in the dimension), e.g. to take $m = O(\log d)$. To prove that this choice is suitable, we need to prove that the code is efficiently list-decodable, and it must behave almost as well as a random code when considering the probabilities of collision between two vectors from the sphere at some angle θ.

[2]For convenience we will present the construction below for $k = 1$ so that the $t \cdot k$ combined filters are just t simple filters, as taking $k = 1$ also seems to be the best choice in practice. The construction can be extended to arbitrary k with similar techniques.

Note that with these concatenated codes, the norms of vectors $\mathbf{s} \in C$ are guaranteed to be equal to 1 as we set the norm of each block to be equal to $\sqrt{1/m}$ and vectors in different blocks are orthogonal.

List-decodability of random product codes. We first describe an efficient list decoding algorithm for the above random product codes in the regime where the list L has exponential size in d. A short description is given in the following proof and pseudo-code as well as a more detailed description is provided below.

Lemma 13.6. *There exists an algorithm that, given the description $(C_1, \ldots, C_m)$ of a random product code $C = \mathrm{RPC}(d, m, t)$, a target vector $\boldsymbol{v} \in \mathbb{R}^d$, and a parameter $\alpha < 1$, returns the set of α-relevant filters $S = C \cap \mathcal{C}_{\boldsymbol{v}/\|\boldsymbol{v}\|,\alpha}$ of size $|S| = F_{\boldsymbol{v}} = \tilde{O}(t \cdot \mathcal{C}_d(\alpha))$ in average time $\tilde{O}(dt' + mt' \log t' + mt\mathcal{C}_d(\alpha))$ over the randomness of $C \leftarrow \mathrm{RPC}(d, m, t)$.*

Sketch of the proof. The algorithm receives as input a code $C = A \cdot (C_1 \times \cdots \times C_m)$ where $|C_i| = t' = t^{1/m}$; a target vector $\boldsymbol{v} \in \mathbb{R}^d$; and a parameter $\alpha < 1$.

We first parse $\boldsymbol{v}$ as $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m) \in (\mathbb{R}^{d/m})^m$. For all elements in each set C_j we then compute all dot products $\langle \boldsymbol{v}_i, \mathbf{s}_{i,j} \rangle$ and sort them by dot products into lists L_j, hence obtaining m lists of size t', where the first elements in the lists are those with the largest inner products with $\boldsymbol{v}_j$.

We then wish to identify all vectors $\mathbf{s} = (\mathbf{s}_{1,i_1}, \ldots, \mathbf{s}_{m,i_m}) \in C_1 \times \cdots \times C_m$ for which $\langle \boldsymbol{v}, \mathbf{s} \rangle \geqslant \alpha \|\boldsymbol{v}\|$, so that $\boldsymbol{v}/\|\boldsymbol{v}\|$ lies in the spherical cap $\mathcal{C}_{\mathbf{s},\alpha}$ and $\boldsymbol{v}$ lies in the hypercone corresponding to this cap. To do so, we visit the enumeration tree in a depth-first manner. Its nodes at level $k \leqslant m$ are labeled by vectors in $C_1 \times \cdots \times C_k$, and the parenthood is defined by the direct prefix relation. We use the sorted lists L_j to define in which order we will visit siblings. Because the lists are sorted, if a node has no solution in its descendants, then we know that all its next siblings will not lead to a solution either. This allows to prune the enumeration tree considerably, and guarantees that the number of visited nodes is no larger than $2mF_{\boldsymbol{v}} + 1$.

The overall running time is the sum of the three following terms: $m \cdot t'$ scalar products of dimension $d' = d/m$ for computing partial inner products; $m \cdot t' \log t'$ operations for sorting each of the lists by largest dot product; and finally the visit of $O(m \cdot F_{\boldsymbol{v}})$ nodes for the pruned enumeration, where $F_{\boldsymbol{v}} = \tilde{O}(t \cdot \mathcal{C}_d(\alpha))$. □

The detailed algorithm that can perform this search for relevant filters with random product codes is given as Algorithm 13.1. In this algorithm, we denote by $\mathbf{c}_{i,j}$ for $j \in [1, t']$ the elements of C_i after the sort, and $d_{i,j} = \langle \boldsymbol{v}_i, \mathbf{c}_{i,j} \rangle$ denotes their dot product with part of the target vector $\boldsymbol{v}$. The algorithm is inspired by the lattice enumeration algorithm of Fincke, Pohst, and Kannan [Kan83, FP85], with some additional precomputations exploiting the structure of the code, which significantly shrinks the enumeration tree.

For simplicity, the core of Algorithm 13.1 is described as m nested for-loops. If m is a variable and not a fixed parameter, we let the reader replace the m loops with its equivalent recursive or while-based construction. Note that if an index j_k is rejected at the k-th for-loop, then we know that the partial vector $(\mathbf{c}_{1,j_1}, \ldots, \mathbf{c}_{k,j_k})$ cannot be extended as a near neighbor, since even after adding all maximum partial dot products $d_{\ell,1}$ for $\ell \geqslant k+1$, the overall dot product remains smaller than α. This, combined with the fact that the condition in the m-th for-loop is exactly $\langle \boldsymbol{v}, (\mathbf{c}_{1,j_1}, \ldots, \mathbf{c}_{m,j_m}) \rangle \geqslant \alpha \cdot \|\boldsymbol{v}\|$, proves that the algorithm enumerates $C \cap \mathcal{C}_{\boldsymbol{v},\alpha}$, i.e., all code words which are neighbor to $\boldsymbol{v}$.

Algorithm 13.1 Efficient decoding for random product codes

Require: The description $C_1, \ldots, C_m$ of C, a target $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m) \in \mathbb{R}^d$, and $\alpha < 1$
Ensure: The output set S contains all α-close code words to $\boldsymbol{v}$ in C
1: Sort each C_i by decreasing dot-product with $\boldsymbol{v}_i$
2: Precompute partial bounds $R_i = \alpha \cdot \|\boldsymbol{v}\| - \sum_{k=i+1}^{m} d_{k,1}$ for $i \in \{1, \ldots, m\}$
3: Initialize $S \leftarrow \varnothing$
4: **for each** $j_1 \in \{1, \ldots, t'\}$ satisfying $d_{i,j_1} \geqslant R_1$ **do**
5: **for each** $j_2 \in \{1, \ldots, t'\}$ satisfying $d_{i,j_2} \geqslant R_2 - d_{1,j_1}$ **do**
6: **for each** $j_3 \in \{1, \ldots, t'\}$ satisfying $d_{i,j_2} \geqslant R_3 - d_{1,j_1} - d_{2,j_2}$ **do**
7: [...]
8: **for each** $j_m \in \{1, \ldots, t'\}$ satisfying $d_{i,j_m} \geqslant R_m - \sum_{i=1}^{m} d_{i,j_i}$ **do**
9: Add the solution $\boldsymbol{c} = (\boldsymbol{c}_{1,j_1}, \ldots, \boldsymbol{c}_{m,j_m})$ to the set S
10: [...]
11: **return** S

Furthermore in this algorithm, unlike in classical enumeration algorithms, this additional property proves that there is no dead branch during enumeration: each time we enter the k-th for-loop on index j_k, we are guaranteed that at least the neighbor $(\boldsymbol{c}_{1,j_1}, \ldots, \boldsymbol{c}_{k,j_k}, \boldsymbol{c}_{k+1,1}, \ldots, \boldsymbol{c}_{m,1})$ leads to a solution and will be added to the list S. Thus, the overall complexity of the for-loop parts is proportional to $2m$ times the number of solutions $F_{\boldsymbol{v}}$: m nodes for the solution, and potentially m direct siblings of these nodes for which we observe there are no other solutions.

Efficient list-decodability regime. If the parameters are such that the average output size $F_{\boldsymbol{v}}$ is larger than the main overhead of decoding of $t' \log t' = \tilde{O}(t^{1/m})$, then we are in the regime of efficient list-decoding: the running time is then essentially proportional to the output size. This is trivially the case when $t = 2^{\Omega(d)}$, $\alpha < 1$, $m = \log d$, and we are in the dense case of $n = 2^{\kappa d}$ for $\kappa > 0$.

Random behavior of random product codes. On average over the randomness of the code, for two vectors $\boldsymbol{v}, \boldsymbol{w}$ at angular distance θ we expect $t \cdot \mathcal{W}_d(\alpha, \alpha, \theta)$ code words $\boldsymbol{c} \in C$ to simultaneously fulfill $\langle \boldsymbol{v}, \boldsymbol{c} \rangle \geqslant \alpha \|\boldsymbol{v}\|$ and $\langle \boldsymbol{w}, \boldsymbol{c} \rangle \geqslant \beta \|\boldsymbol{w}\|$. But it could be the case that the set $I = C \cap \mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta}$ is empty most of the time, and very large in some cases; this is in particular the case if all the points of C are concentrated in a small region of the space. For large m we may lose this *smoothness* property of dividing the code words equally over the sphere, and we need to show that for our choice of m this is not an issue.

The following theorem states that the probability of collision for random product codes does not deviate much from the collision probabilities for uniformly random codes.

Theorem 13.7. *Suppose that* $t \cdot \mathcal{W}_d(\alpha, \beta, \theta) = 2^{\tilde{o}(\sqrt{d})}$ *or* $t \cdot \mathcal{W}_d(\alpha, \beta, \theta) = 2^{\tilde{\Omega}(\sqrt{d})}$ *as* $d \to \infty$. *Then, for all* $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^d$ *at angle* θ, *over the choice* $C \leftarrow \mathrm{RPC}(d, m, t)$, *the real probability* $p(\theta)$ *that at least one code word* $\boldsymbol{c} \in C$ *lies in the wedge* $\mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta}$ *satisfies:*

$$\min\left\{2^{-\tilde{o}(\sqrt{d})} \cdot t \cdot \mathcal{W}_d(\alpha, \beta, \theta),\ 1 - \mathrm{negl}(d)\right\} \leqslant p(\theta) \leqslant t \cdot \mathcal{W}_d(\alpha, \beta, \theta). \qquad (13.26)$$

Proof. The second inequality $p \leqslant t \cdot \mathcal{W}_d(\alpha, \beta, \theta)$ is straightforward: for any $\mathbf{c} \in C_1 \times \cdots \times C_m$, the probability that $\mathbf{c}$ falls in $\mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta}$ is exactly $\mathcal{W}_d(\alpha, \beta, \theta)$. Using the union bound over the t (dependent!) code words, the result follows.

The first inequality requires more care. We list the two geometric facts required to proceed, facts detailed and proved as Lemma 13.11 below. In the following, we have parsed $\boldsymbol{v}$ as $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)$ and $\boldsymbol{w}$ as $(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m)$.

1. The wedge $\mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta}$ contains the product of sub-wedges Π:

$$\Pi = \prod_{i=1}^{m} \frac{1}{\sqrt{m}} \mathcal{W}_{\sqrt{m}\boldsymbol{v}_i,\alpha,\sqrt{m}\boldsymbol{w}_i,\beta} \subset \mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta}. \tag{13.27}$$

2. The aforementioned sub-wedges have parameters close to the original wedges except with negligible probability. That is, for $\varepsilon = \tilde{O}(1/\sqrt{d})$,

$$\frac{\mu(\mathcal{W}_{\sqrt{m}\boldsymbol{v}_i,\alpha,\sqrt{m}\boldsymbol{w}_i,\beta})}{\mu(\mathcal{S}^{d'-1})} \geqslant \mathcal{W}_{d'}(\alpha - \varepsilon, \beta - \varepsilon, \theta - \varepsilon) \geqslant \frac{\mathcal{W}_d^{1/m}(\alpha, \beta, \theta)}{2^{\tilde{O}(\sqrt{d})}}. \tag{13.28}$$

Because of the inclusion (Equation (13.27)), the probability p that $C \cap \mathcal{W}_{\boldsymbol{v},\alpha,\boldsymbol{w},\beta}$ is not empty must be greater than the probability that each $C_i \cap \mathcal{W}_{\sqrt{m}\boldsymbol{v}_i,\alpha,\sqrt{m}\boldsymbol{w}_i,\beta}$ is non-empty. Since all codes C_i are perfectly random and uniformly independent, we have

$$p \geqslant p_1 p_2 \ldots p_m - \mathrm{negl}(d), \qquad p_i = 1 - \left(1 - \frac{\mu(\mathcal{W}_{\sqrt{m}\boldsymbol{v}_i,\alpha,\sqrt{m}\boldsymbol{w}_i,\beta})}{\mu(\mathcal{S}^{d'-1})}\right)^{t'}. \tag{13.29}$$

For conciseness, we set $W = \mathcal{W}_d(\alpha, \beta, \theta)$. Now, from (13.27), we deduce that $p_i = 1 - (1 - W^{1/m}/2^{\tilde{O}(\sqrt{d})})^{t'}$. We now discuss the two cases:

- If $t' \cdot W^{1/m} \to 0$, then $p_i = 1 - (1 - W^{1/m}/2^{\tilde{O}(\sqrt{d})})^{t'} = t' W^{1/m}/2^{\tilde{O}(\sqrt{d})}$, so

$$p \geqslant \left(\frac{t' \cdot W^{1/m}}{2^{\tilde{O}(\sqrt{d})}}\right)^m \geqslant \frac{t \cdot W}{2^{\tilde{O}(\sqrt{d})}}. \tag{13.30}$$

- If $t' \cdot W^{1/m} \geqslant 2^{\tilde{O}(\sqrt{d})}$, then $p_i = 1 - \mathrm{negl}(d)$, so that $p \geqslant 1 - \mathrm{negl}(d)$.

In both case, that provides the desired result. So assuming that (13.27) holds, this concludes the proof. □

To prove (13.27), we continue with a sequence of four lemmas below. The first (Lemma 13.8) is a standard bound on the tail of multivariate Gaussian distributions. The second (Lemma 13.9) describes that if we cut a vector into a small number of pieces m, then each of these pieces will roughly have the same norm. Lemma 13.10 then states that if two vectors are orthogonal and are cut into m pieces, then also the m pieces will be (almost) orthogonal. Finally, the main technical Lemma 13.11 proves (13.27).

Lemma 13.8. *[LM00] For $\boldsymbol{v}$ sampled as $\boldsymbol{v} \leftarrow \mathcal{N}(0,1)^d$ and $\alpha > 0$ we have*

$$\mathbb{P}(\|\boldsymbol{v}\|^2 - d \geqslant 2\sqrt{d\alpha} + 2\alpha) \leqslant e^{-\alpha}, \quad \textit{and } \mathbb{P}(\|\boldsymbol{v}\|^2 - d \leqslant 2\sqrt{d\alpha}) \leqslant e^{-\alpha}.$$

Taking $\alpha = \log^2 d$ we have $\frac{1}{d}\|\boldsymbol{v}\|^2 = 1 + \tilde{O}(d^{-1/2})$ except with negligible probability in d.

Lemma 13.9. *Let $\boldsymbol{v} \in \mathcal{S}^{d-1}$ be sampled uniformly at random from the sphere, and let us write $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)$ with $\boldsymbol{v}_i \in \mathbb{R}^{d/m}$ for $i = 1, \ldots, m$. Then, except with negligible probability, we have that for all $i = 1, \ldots, m$:*

$$\|\boldsymbol{v}_i\|^2 = \frac{1}{m}\left(1 + \tilde{O}\left(d^{-1/2}\right)\right). \tag{13.31}$$

Proof. The uniform distribution on the sphere can be sampled by drawing $\boldsymbol{v}' \leftarrow \mathcal{N}(0,1)^d$ and taking $\boldsymbol{v} = \boldsymbol{v}'/\|\boldsymbol{v}'\|$. Writing $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m)$ and $\boldsymbol{v}' = (\boldsymbol{v}'_1, \ldots, \boldsymbol{v}'_m)$, we then have $\boldsymbol{v}_i = \boldsymbol{v}'_i/\|\boldsymbol{v}'\|$. We conclude the proof by applying Lemma 13.8 to $\|\boldsymbol{v}'_i\|$ and $\|\boldsymbol{v}'\|$. □

Lemma 13.10. *Let $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{S}^{d-1}$ be sampled from the sphere, conditioned on $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = 0$. Then for all i we have $|\langle \boldsymbol{w}_i, \boldsymbol{v}_i \rangle| \leqslant O((\log d)/\sqrt{d})$ except with negligible probability.*

Proof. The distribution of the pair $(\boldsymbol{v}, \boldsymbol{w})$ may be sampled by applying Gram-Schmidt orthogonalization to independent normal vectors:

$$\boldsymbol{v} = \frac{\boldsymbol{v}'}{\|\boldsymbol{v}'\|}, \quad \boldsymbol{w} = \frac{\boldsymbol{w}' - \langle \boldsymbol{w}', \boldsymbol{v} \rangle \boldsymbol{v}}{\|\boldsymbol{w}' - \langle \boldsymbol{w}', \boldsymbol{v} \rangle \boldsymbol{v}\|}, \quad \text{where } \boldsymbol{v}', \boldsymbol{w}' \leftarrow \mathcal{N}(0, \tfrac{1}{d})^d. \tag{13.32}$$

By Lemma 13.8, we have that except with negligible probability, $\frac{3}{4} \leqslant \|\boldsymbol{v}'\|, \|\boldsymbol{w}'\| \leqslant \frac{5}{4}$. Additionally, $\langle \boldsymbol{w}', \boldsymbol{v} \rangle$ is distributed according to $\mathcal{N}(0, \frac{1}{d})$ over the randomness of $\boldsymbol{w}'$, so we have $|\langle \boldsymbol{w}', \boldsymbol{v} \rangle| \leqslant (\log d)/\sqrt{d}$ except with negligible probability. First, this implies that $\|\boldsymbol{w}' - \langle \boldsymbol{w}', \boldsymbol{v} \rangle \boldsymbol{v}\| \geqslant \frac{1}{2}$, and we derive:

$$|\langle \boldsymbol{w}_i, \boldsymbol{v}_i \rangle| \leqslant 2\left(\langle \boldsymbol{w}'_i, \boldsymbol{v}_i \rangle - \langle \boldsymbol{w}', \boldsymbol{v} \rangle \|\boldsymbol{v}_i\|\right) \tag{13.33}$$

Again, over the randomness of $\boldsymbol{w}'$, the inner products $\langle \boldsymbol{w}'_i, \boldsymbol{v}_i \rangle$ are distributed according to $\mathcal{N}(0, \frac{1}{d}\|\boldsymbol{v}_i\|^2)$, so with overwhelming probability we have:

$$|\langle \boldsymbol{w}_i, \boldsymbol{v}_i \rangle| \leqslant 4 \cdot \|\boldsymbol{v}_i\| \cdot (\log d)/\sqrt{d}. \tag{13.34}$$

Finally, we invoke Lemma 13.9 to conclude that for all i, we have $|\langle \boldsymbol{w}_i, \boldsymbol{v}_i \rangle| \leqslant O((\log d)/\sqrt{d})$ except with negligible probability. □

We are now ready to prove our main technical Lemma.

Lemma 13.11. *Let $\boldsymbol{v}, \boldsymbol{w}$ be independent, uniformly random samples from $\mathcal{S}^{d-1}$ conditioned on the fact that $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \cos\theta$. Then, except with negligible probability, for some $\varepsilon = \tilde{O}(d^{-1/2})$ the following holds for all i:*

$$\frac{\mu(\mathcal{W}_{\sqrt{m}\boldsymbol{v}_i, \alpha, \sqrt{m}\boldsymbol{w}_i, \beta})}{\mu(\mathcal{S}^{d'-1})} \geqslant \mathcal{W}_{d'}(\alpha - \varepsilon, \beta - \varepsilon, \theta - \varepsilon). \tag{13.35}$$

Additionally, the wedge product $\Pi = \prod_{i=1}^{m} \frac{1}{\sqrt{m}} \mathcal{W}_{\sqrt{m}\boldsymbol{v}_i, \alpha, \sqrt{m}\boldsymbol{w}_i, \beta}$ is included in $\mathcal{W}_{\boldsymbol{v}, \alpha, \boldsymbol{w}, \beta}$.

Proof. Let us start by proving the inclusion $\Pi \subset \mathcal{W}_{\boldsymbol{v}, \alpha, \boldsymbol{w}, \beta}$. Let $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) \in \Pi$, that is, each $\boldsymbol{x}_i$ belongs to $\frac{1}{\sqrt{m}}\mathcal{S}^{d'-1}$ and satisfies $\langle \boldsymbol{x}_i, \boldsymbol{v}_i \rangle \geqslant \frac{\alpha}{m}$ and $\langle \boldsymbol{x}_i, \boldsymbol{w}_i \rangle \geqslant \frac{\beta}{m}$. Summing over all i, we obtain $\|\boldsymbol{x}\|^2 = 1$, $\langle \boldsymbol{x}, \boldsymbol{v} \rangle \geqslant \alpha$ and $\langle \boldsymbol{x}, \boldsymbol{w} \rangle \geqslant \beta$, which concludes the proof of the inclusion.

We now move to the proof that $\mu(\mathcal{W}_{\sqrt{m}\boldsymbol{v}_i,\alpha,\sqrt{m}\boldsymbol{w}_i,\beta}) \geqslant \mathcal{W}_{d'}(\alpha-\varepsilon,\beta-\varepsilon,\theta-\varepsilon)$. First note that $\mathcal{W}_{\mathbf{a},\alpha,\mathbf{b},\beta}$ has volume $\mathcal{W}(\frac{\alpha}{\|\mathbf{a}\|},\frac{\beta}{\|\mathbf{b}\|},\frac{\langle\mathbf{a},\mathbf{b}\rangle}{\|\mathbf{a}\|\|\mathbf{b}\|})$, so it is enough to prove that $\|\boldsymbol{v}_i\|^2 = \frac{1}{m}(1+\tilde{O}(d^{-1/2}))$, $\|\boldsymbol{w}_i\|^2 = \frac{1}{m}(1+\tilde{O}(d^{-1/2}))$ and $\langle\boldsymbol{v}_i,\boldsymbol{w}_i\rangle = \frac{\cos\theta}{m}(1+\tilde{O}(d^{-1/2}))$. The first two statements follow from Lemma 13.9 with overwhelming probability. For the last one, write $\boldsymbol{w} = \boldsymbol{v}\cos\theta + \boldsymbol{w}'\sin\theta$ where $\boldsymbol{v},\boldsymbol{w}'$ are sampled uniformly on the sphere conditioned on being orthogonal. Then, one writes

$$\langle\boldsymbol{v}_i,\boldsymbol{w}_i\rangle = \|\boldsymbol{v}_i\|^2\cos\theta + \langle\boldsymbol{v}_i,\boldsymbol{w}_i'\rangle = \frac{\cos\theta}{m}\left(1+\tilde{O}\left(\frac{1}{\sqrt{d}}\right)\right) + O\left(\frac{\log d}{\sqrt{d}}\right), \quad (13.36)$$

where the second term follows from Lemma 13.10. This concludes the proof. □

Application to locality-sensitive filtering. Equipped with this code we may now replace, in the construction of Sections 13.2 and 13.3, the set of t independent filters, by a set of filters defined by a code $C \leftarrow \mathrm{RPC}(m,d,t)$. Algorithm 13.1 provides the efficient oracle $\mathcal{O}$ that computes the set of relevant filters for a vector for parameter α (query) or β (insertion/deletion). Theorem 13.7 ensures that the probabilities of collisions (and hence, the complexity analysis) presented in Sections 13.2 and 13.3 also hold when the filters are not chosen independently but according to a random product code.

13.3.5 – Relation with other techniques. Before continuing with the application to sieving, let us briefly highlight the relation between the hypercone filtering approach and other methods considered in previous chapters:

- Cross-polytope hashing: Taking $\alpha = \beta = 1/\sqrt{\log_2(2d)}$ in the filtering approach is very similar to using cross-polytope hashing, where instead of sampling $2^{\log_2(2d)} = 2d$ random filter vectors from the sphere, we take these points as the vertices of the cross-polytope.
- Hypercone hashing: Taking $\alpha = \beta = d^{-1/4}$ essentially leads to hypercone hashing, where the list-decoding now becomes unnecessary to guarantee that there is no exponential overhead in the decoding stages.
- Hyperplane hashing: Taking $\alpha = \beta = 0$ means we divide the space into half-spaces, which closely resembles the hyperplane hashing approach of Charikar [Cha02].

Note that we obtained the best results in this chapter with $\alpha,\beta \in (0,1)$ strictly larger than 0 (and not converging to 0 as $d \to \infty$). The previous best results were obtained using cross-polytope hashing and hypercone hashing, which both correspond to $\alpha = \beta = o(1)$. The worst asymptotic results were obtained with hyperplane hashing, corresponding to $\alpha = \beta = 0$. In the end all methods are very similar, and it seems that the larger α and β, the better.

13.4 — The Nguyễn–Vidick sieve with hypercone LSF

Let us now consider how we can apply the hypercone filtering framework to sieving, and as usual we first focus on the Nguyễn–Vidick sieve. Combining hypercone filtering with sieving is very similar to combining locality-sensitive hash methods with sieving, but there are some small changes. Taking the sieve described in Algorithm 10.1 as our starting point, we obtain the procedure described in Algorithm 13.2 as our new algorithm. Blue lines indicate modifications to the original algorithm, and computing the set of nearby filter vectors in Lines 5 and 11 is done using Algorithm 13.1.

Algorithm 13.2 Nguyễn and Vidick's sieve with hypercone LSF

Require: An input list L of $(4/3)^{d/2+o(d)}$ vectors of norm at most R
Ensure: The output list L' has $(4/3)^{d/2+o(d)}$ vectors of norm at most $\gamma \cdot R$

Initialize an empty list L_{m+1}
Sample $\hat{C} \subset L \cap \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x}\| \geqslant \gamma \cdot R\}$ of size $\mathrm{poly}(d) \cdot (4/3)^{d/2}$
Sample m random subcodes $C_1, \ldots, C_m$
for each $\boldsymbol{w} \in \hat{C}$ **do**
    Compute the set $\mathcal{F}_{\boldsymbol{w},\beta}$ of all β-close filters to $\boldsymbol{w}$
    Add $\boldsymbol{w}$ to all β-close filters $f \in \mathcal{F}_{\boldsymbol{w},\beta}$
for each $\boldsymbol{v} \in L \setminus \hat{C}$ **do**
    if $\|\boldsymbol{v}\| \leqslant \gamma R$ **then**
        Add $\boldsymbol{v}$ to the list L'
    else
        Compute the set $\mathcal{F}_{\boldsymbol{v},\alpha}$ of all α-close filters to $\boldsymbol{v}$
        Obtain the set of candidates $\mathcal{C} = \bigcup_{f \in \mathcal{F}_{\boldsymbol{v},\alpha}} B_f$
        for each $\boldsymbol{w} \in \mathcal{C}$ **do**
            if $\|\boldsymbol{v} - \boldsymbol{w}\| \leqslant \gamma \cdot R$ **then**
                Add $\boldsymbol{v} - \boldsymbol{w}$ to the list L'
                Continue the loop over "$\boldsymbol{v} \in L \setminus \hat{C}$"

13.4.1 – High-dimensional intuition ($\alpha = \beta$). We again provide a first basic estimate of the optimal asymptotic complexities and parameter choices, based on the assumption that everything in high dimensions is orthogonal (except for nearby pairs of vectors). This means that we can simply substitute $\theta_1 = \frac{\pi}{3}$ and $\theta_2 = \frac{\pi}{2}$ in earlier expressions. This leads to a greatly simplified analysis, with the following conclusion.

Proposition 13.12. *Assuming that non-reducing vectors are always pairwise orthogonal, the NV-sieve with hypercone filtering with parameters $\alpha = \beta = \frac{1}{5}\sqrt{5}$, $k = 1$ and $t = (15/11)^{d/2+o(d)} \approx 2^{0.2238d+o(d)}$ heuristically solves SVP in time and space $2^{0.2703d+o(d)}$. By varying the value of α and t, we further obtain the trade-off between the space and time complexities indicated by the straight dashed line in Figure 13.3.*

Proof. If 'random angles' are 90°, then we can substitute $\theta_1 = \frac{\pi}{3}$ and $\theta_2 = \frac{\pi}{2}$ into previous expressions for various parameters. Substituting $n = (4/3)^{d/2+o(d)}$ into α_0 in (13.23), we obtain $\alpha_0 = \frac{1}{5}\sqrt{5} + o(1)$, and the optimal choice for $\alpha = \beta$ then becomes $\alpha = \beta = \frac{1}{5}\sqrt{5} + o(1)$. Substituting this choice into the expressions for ρ and ρ_t of (13.2), we obtain $\rho = \log(12/11)/\log(4/3)$ and $\rho_t = \log(15/11)/\log(4/3)$. This leads to time and space complexities of $n^{1+\rho} = (12/11)^{d/2+o(d)}$ □

Note that the estimated exponent $\rho = \log(12/11)/\log(4/3) \approx 0.30$ and estimated time complexity $2^{0.2703d+o(d)}$ are better than the same estimates $\rho = \frac{1}{3} \approx 0.33$ and a time complexity of $2^{0.2767d+o(d)}$ for hypercone and cross-polytope LSH. So if this first estimate is better, then we might expect that the actual complexities are also better.

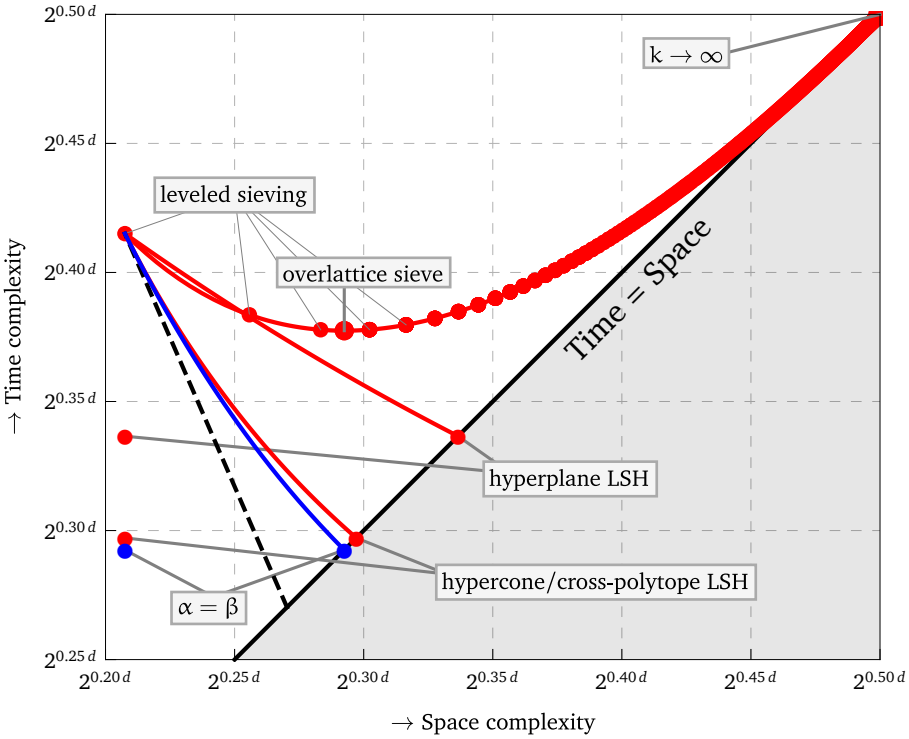


Figure 13.3: The asymptotic time-memory trade-off for hypercone filtering (blue) compared to the asymptotic complexities of other sieving algorithms considered in previous chapters (red), assuming that $\alpha = \beta$. The dashed line corresponds to the preliminary estimate under the assumption that distant vectors are always orthogonal to the target vector.

13.4.2 – Solving SVP in time and space $2^{0.2925d+o(d)}$ ($\alpha = \beta$). We now continue with the actual cost analysis of sieving with hypercone LSH, still assuming that $\alpha = \beta$. We will perform the analysis differently than in previous chapters, where we make use of one crucial property that did not hold in previous chapters: each filter bucket covers an equally big region of the space. Hypercones may intersect, but in that case we just add vectors to several buckets. This is different from hypercone hashing where regions become progressively smaller; hyperplane hashing where non-orthogonal hyperplanes may not equally divide the space in regions; and cross-polytope hashing, where the combination of k hash functions may again make some hash buckets significantly smaller or larger than others. This means that the simpler analysis presented below does not apply to the techniques considered in previous chapters.

Theorem 13.13. *The Nguyễn–Vidick sieve with hypercone LSF with parameters*

$$\alpha = \beta = \frac{1}{2}, \qquad k = 1, \qquad t = (3/2)^{d/2+o(d)}, \tag{13.37}$$

heuristically solves SVP in time and space $(3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}$. By varying $\alpha \in (0,1)$, we further obtain the trade-off indicated by the blue curve in Figure 13.3.

Proof. We fix $k = 1$, so that each bucket covers an equally-sized part of $\mathbb{R}^d$, and each vector is on average contained in $t \cdot \mathcal{C}_d(\alpha)$ buckets, we can conclude that there will be in total $t \cdot n \cdot \mathcal{C}_d(\alpha)$ entries in the hash filters, and each filter bucket will therefore have $\tilde{O}(n \cdot C_d(\alpha))$ occupied entries. This is using the assumption that the normalized list vectors follow a uniformly random distribution on the sphere.

This means that the cost of answering a query $\boldsymbol{v}$ is: (i) compute the $\tilde{O}(t \cdot \mathcal{C}_d(\alpha))$ relevant filters for $\boldsymbol{v}$; and (ii) compare $\boldsymbol{v}$ to all other $\tilde{O}(t \cdot \mathcal{C}_d(\alpha) \cdot n \cdot \mathcal{C}_d(\alpha))$ vectors in the same buckets. The cost of insertions in the relevant filters is $\tilde{O}(t \cdot \mathcal{C}_d(\beta))$ for each vector, matching the query costs as $\alpha = \beta$. Overall, this leads to a total time complexity of $\tilde{O}(n^2 \cdot t \cdot \mathcal{C}_d(\alpha)^2)$ and a space complexity of $\tilde{O}(n \cdot t \cdot \mathcal{C}_d(\alpha))$. The optimal choice minimizing the time complexity is $\alpha = \frac{1}{2}$ so that $\mathcal{C}_d(\alpha) \propto \frac{1}{n}$ and the complexities are as stated. Varying $\alpha \in (0, \frac{1}{2})$ further leads to the curve in Figure 13.3. □

Note that $\alpha = \alpha_0 = \frac{1}{2}$ and $n^{2/d} = \frac{4}{3}$ in (13.23) corresponds to $\theta_2 = \arccos(\frac{1}{7}) \approx 0.45\pi$, which may be compared with e.g. the dominant angle $\theta_2^* \approx 0.43\pi$ in Corollary 11.7 and $\theta_2^* \approx 0.46\pi$ in Corollary 10.9. The closer θ_2^* is to $\frac{\pi}{2}$, the closer the actual complexity is to the preliminary estimate under the assumption that $\theta_2 = \frac{\pi}{2}$.

13.4.3 – Reducing the space complexity with provable probing ($\alpha \neq \beta$). For $\alpha = \beta$ we saw in Figure 13.3 that we obtain slightly better time and space complexities than with hypercone LSH. To improve the practical space/time trade-off, for locality-sensitive hashing we saw we can use *probing*, but only in few cases [Kap15, Pan06] did this lead to provable improvements in the exponent; for cross-polytope hashing for instance, it is unclear how the proposed probing method from Section 12.4.3 affects the asymptotics.

By taking $\alpha \neq \beta$, we can trade time for memory as well, while still obtaining provable guarantees on the exponents. Taking the query parameter α smaller than the storage parameter β, we may even achieve a quasi-linear space complexity while still achieving better query complexities. One can achieve an optimal query complexity (without increasing the memory) by taking $\alpha = \frac{1}{4}$ and $\beta = \frac{1}{2}$. Indeed, if we work out the costs for this parameter choice, we obtain a space complexity of $(4/3)^{d/2+o(d)} \approx 2^{0.208d+o(d)}$, and a time complexity of only $(5/3)^{d/2+o(d)} \approx 2^{0.368d+o(d)}$. Note that this strict speed-up (rather than a trade-off) applies to the GaussSieve as well. Fixing $\beta = \frac{1}{2}$ and varying $\alpha \in [0, \frac{1}{2}]$ we obtain the trade-off curve given in Figure 13.1. This is indeed the best we can do; other choices $\alpha, \beta \in [0, 1]$ all lead to points above and to the right of this curve.

13.4.4 – Solving SVP in time $2^{0.2925d+o(d)}$ and space $2^{0.2075d+o(d)}$. Finally, similar to previous chapters, we see that for the Nguyễn–Vidick sieve we can obtain the improved time complexities on the various trade-off curves without increasing the memory, by processing the hash tables sequentially. In the case of filtering the parallelization is similar but not quite the same, as the number of filters $t = \tilde{O}(n^{\rho_t})$ is larger than before; processing all filters one by one would lead to a minimum time complexity of $\tilde{O}(t \cdot n) \geqslant \tilde{O}(n^2)$. The whole point of the filtering approach was that we used list-decoding to find relevant filters quickly, so that overall the cost of processing one filter is subexponential in d.

Fortunately, with a minor modification we can still apply the same ideas here. Instead of processing all t filters sequentially, we partition the t filters C into n^ρ groups of size t/n^ρ, and then process one subset of filters first. Repeating the procedure of processing a subset of filters n^ρ times, we ultimately obtain the desired quasi-linear space complexity,

Algorithm 13.3 Nguyễn and Vidick's sieve with hypercone LSF, space-efficient

Require: An input list L of $(4/3)^{d/2+o(d)}$ vectors of norm at most R
Ensure: The output list L' has $(4/3)^{d/2+o(d)}$ vectors of norm at most $\gamma \cdot R$
Initialize an empty list L'
Sample $\hat{C} \subset L \cap \{x \in \mathbb{R}^d : \|x\| \geqslant \gamma \cdot R\}$ of size $\mathrm{poly}(d) \cdot (4/3)^{d/2}$
Sample m random subcodes $C_1, \ldots, C_m$
for each $c' = (c_1, \ldots, c_u) \in C_1 \times \cdots \times C_u$ **do**
    for each $w \in \hat{C}$ **do**
        Compute the set $\mathcal{F}_{w,\beta}^{(c')}$ of all β-close filters to w with prefix c'
        Add w to all filters $f \in \mathcal{F}_{w,\beta}^{(c')}$
    for each $v \in L \setminus \hat{C}$ **do**
        if $\|v\| \leqslant \gamma R$ **then**
            Add v to the list L'
        else
            Compute the set $\mathcal{F}_{v,\alpha}^{(c')}$ of all α-close filters to v with prefix c'
            Obtain the set of candidates $\mathcal{C} = \bigcup_{f \in \mathcal{F}_{v,\alpha}^{(c')}} B_f$
            for each $w \in \mathcal{C}$ **do**
                if $\|v - w\| \leqslant \gamma R$ **then**
                    Add v to the list L'
                    Continue the loop over "$v \in L \setminus \hat{C}$"

while the time complexity hardly increases compared to processing all filters simultaneously; the overhead of list-decoding becomes slightly larger as for each vector we now go through the enumeration tree multiple times (searching for code words in the subset of the code that we are dealing with), but in the end this overhead remains subexponential.

Theorem 13.14. *The space-efficient Nguyễn–Vidick sieve with hypercone filtering with* $\alpha = \beta = \frac{1}{2}$, $k = 1$, $t = (3/2)^{d/2+o(d)}$, *and* $\gamma \to 1$ *heuristically solves SVP in time* $(3/2)^{d/2+o(d)} \approx 2^{0.2925d+o(d)}$ *and space* $(4/3)^{d/2+o(d)} \approx 2^{0.2075d+o(d)}$. *These complexities are indicated by the leftmost blue point in Figure 13.1.*

Although arbitrary partitions of the code C achieve these asymptotic complexities, perhaps the most efficient way to divide the code into subcodes in practice is to fix a prefix $c' = (c_1, \ldots, c_u) \in C_1 \times \cdots \times C_u$ and let one subcode correspond to one prefix. Then, processing these parts of the code sequentially corresponds to going through all possible prefixes, and for each prefix considering a partial enumeration tree of depth $m-u$ rather than m. To guarantee that the number of prefixes is $n^\rho = (9/8)^{d/2+o(d)}$, we take $u = \frac{\log(9/8)}{\log(3/2)} \cdot m \approx 0.2905 \cdot m$. Note that there are $t^{u/m}$ code words $C_1 \times \cdots \times C_m$, which combined with the expression for t shows there are $(9/8)^{d/2+o(d)}$ prefix code words in $C_1 \times \cdots \times C_u$, and $(4/3)^{d/2+o(d)}$ suffixes in $C_{u+1} \times \cdots \times C_m$ for each prefix. The result of this slightly more practical partitioning method is Algorithm 13.3.

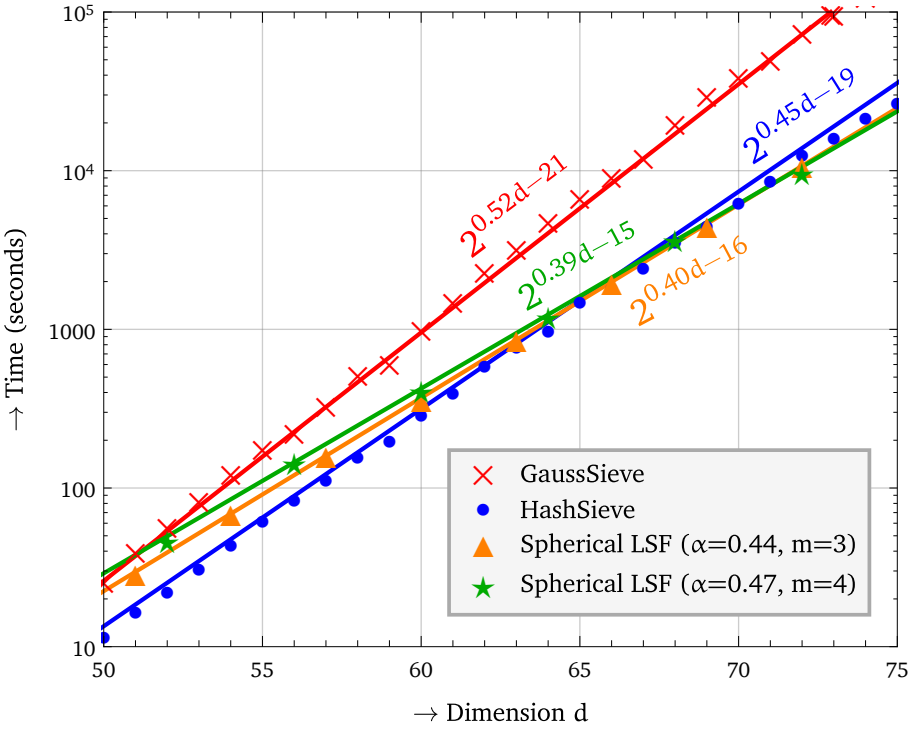


Figure 13.4: The running time of the GaussSieve (red), the GaussSieve with hyperplane hashing (HashSieve, blue), and the GaussSieve with hypercone filtering with $\alpha = 0.44$ and $m = 3$ (orange) and with $\alpha = 0.47$ and $m = 4$ (green). Points indicate experimental data, lines indicate least-squares fits of the form 2^{ad+b} for constants a, b. For simplicity we have only performed experiments for dimensions d which are multiples of the number of blocks m.

13.5 — The GaussSieve with hypercone LSF

To show the practicability of the proposed algorithm, let us finally present results of implementing the filtering approach in the GaussSieve algorithm [MV10b]. We ran experiments on an Intel Quad-Core(TM) Q9550 at 2.83GHz with 4GB RAM. Our implementation is not vectorized or parallelized. As input bases, we used LLL-reduced bases of the SVP challenge [SG15]. We chose $\alpha = \beta \in \{0.44, 0.47\}$ as $\alpha = \beta = \frac{1}{2}$ is only optimal asymptotically and appears to be slightly worse in low dimensions. Figure 13.4 compares the running time of this algorithm with the GaussSieve and the HashSieve (Chapter 10).

Observe that the acceleration matches predictions from the theoretical analysis. For example, for the asymptotically suboptimal choice of $\alpha = 0.44$, theory predicts an acceleration of $2^{0.10d+o(d)}$, while in practice we see a speed-up from $2^{0.52d-21}$ to $2^{0.40d-16}$. The polynomial overhead further appears to be small, as this method outperforms the GaussSieve around dimension 50, and the HashSieve around dimension 65.

CHAPTER 14

Effects of quantum search

14.1 — Overview

Context. In the previous chapters we studied sieving algorithms for finding shortest vectors in lattices, keeping in mind that the main application of these results is estimating the hardness of SVP in high dimensions, which is critical for accurately choosing parameters in lattice-based cryptography. Underestimating the computational hardness of SVP would lead to parameter choices which are insecure against cryptographic adversaries, while overestimating the hardness would lead to choosing the parameters too large, leading to unnecessarily inefficient cryptographic primitives. Understanding what a cryptographic adversary might do and how much effort it would cost him to break a scheme is ultimately the best way to defend against such adversaries.

One of the main reasons that lattice-based cryptography has been intensively studied in the past few decades, is due to the rise of quantum computers, making many currently deployed cryptographic schemes insecure. Over the last two decades, quantum algorithms have been developed to e.g. break public-key cryptography based on integer factorization or the discrete logarithm problem [Sho97], break the principal ideal problem in real quadratic number fields [Hal02], and provide subexponential attacks for some systems based on elliptic curve isogenies [CJS14]. Besides these applications in cryptography, quantum algorithms have also been invented to speed up exhaustive searching [BBHT98, Gro96], counting roots [BHMT02] and (with appropriate assumptions about the computing architecture) finding collisions and claws [Amb04, BHT98, BDH$^+$01], among many other quantum algorithmic speed-ups [CvD10, Mos09, SM12].

Quantum algorithms and lattice cryptography. Quantum algorithms have also been studied in the context of lattices (see e.g. [AR03, CGS14, Kup05, Kup13, Lud03, Reg04a, Reg04b]), but until now hard lattice problems have been able to withstand quantum attacks: there are no known quantum algorithms which would solve hard lattice problems in subexponential time. Lattice-based cryptography is still considered one of the main candidates for *post-quantum cryptography* [BBD09, BL15b]: cryptographic systems that are secure in a world where large-scale quantum computers are a reality.

For choosing parameters for lattice-based cryptographic primitives in a post-quantum world, it is crucial not only to study whether quantum algorithms can completely break these schemes, but also to study how quantum techniques affect the complexities of existing classical algorithms for solving these problems. Can quantum algorithms be used

*This chapter is based on results from [LMvdP13, LMvdP15] extended with results from [BL15a, BDGL16].

to speed up existing SVP algorithms? Should parameters be chosen differently to resist quantum attacks? How do quantum algorithms affect the computational complexity of current SVP algorithms? Even if the resulting quantum algorithms have a time complexity exponential in d, reducing the constant in the exponent would mean that security estimates would have to be adjusted for post-quantum settings.

Results. In this chapter we take another look at the sieving algorithms from Chapters 8–13, and we analyze how the celebrated algorithm of Grover for faster exhaustive searching [Gro96] affects the complexities of each of these algorithms. Grover's quantum search algorithm considers the following problem. Given a list L of length n and a function $f : L \to \{0, 1\}$ such that the number of elements $e \in L$ with $f(e) = 1$ is small, construct an algorithm which, given L and f as input, returns an $e \in L$ with $f(e) = 1$, or determines that with high probability no such e exists. We assume for simplicity that f can be evaluated on any particular element of L in unit time.

Classical algorithm. The natural way to find such an element is to go through the whole list, until a special element is found. This takes on average $O(n)$ time which is also optimal: no classical algorithm can find such an element in less than $\Omega(n)$ time.

Quantum algorithm. Using Grover's quantum search algorithm [BBHT98, BHMT02, Gro96], we can find a special list element in time $O(\sqrt{n})$. This is essentially optimal, as any quantum algorithm needs at least $\Omega(\sqrt{n})$ evaluations of f [BBBV97].

Performing the various optimizations again, taking into account that the cost of searching a list of size n may potentially be reduced from $O(n)$ to $O(\sqrt{n})$, we obtain the results given in Table 14.1 and Figure 14.1. Combining leveled sieving and the overlattices approach with quantum search does not lead to better complexities than combining the Nguyễn–Vidick sieve algorithm with quantum search, while for NNS techniques considered in the previous chapters we do get further improvements. The best asymptotic time complexity for solving SVP in high dimensions is obtained by combining hypercone filtering with quantum search, and based on these results we estimate the quantum hardness of SVP in dimension d to be $(13/9)^{d/2+o(d)} \approx 2^{0.265d+o(d)}$, improving upon the classical time complexity of $(3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}$ in dimension d. These results may serve as a guide for estimating the quantum hardness of SVP in high dimensions.

Note on the RAM model. For both the classical and the quantum versions of these search algorithms, we assume a RAM model of computation where the j-th entry of the list L can be looked up in constant or polylogarithmic time. In the case that L is a virtual list where the j-th element can be computed in time polynomial in the length of j (thus polylogarithmic in the length of the list L), then look-up time is not an issue. When L is indeed an unstructured list of values, for classical computation, the assumption of a RAM-like model has usually been valid in practice. However, there are fundamental reasons for questioning it [Ber09], and there are practical computing architectures where the assumption does not apply. In the case of quantum computation, a practical RAM-like quantum memory (e.g. [GLM08]) looks particularly challenging, especially for first generation quantum computers. Some authors have studied the limitations of quantum algorithms in this context [Ber09, GR04].

In this chapter, we consider conventional classical RAM memories for the classical algorithms, and RAM-like quantumly addressable classical memories for the quantum search algorithms. This is both a first step for future studies in assessing the impact of

Table 14.1: A comparison of the time and space complexities of heuristic sieving algorithms for solving SVP, both classically and quantumly. The given complexities for high-level sieving are conjectured (see Chapter 9), other results can be proven under certain heuristic assumptions.

	Algorithm	**Classical search**		**Quantum search**	
	Name [References]	$\log_2(\text{Time})$	$\log_2(\text{Space})$	$\log_2(\text{Time})$	$\log_2(\text{Space})$
Heuristic SVP	Nguyễn–Vidick sieve [NV08]	$0.415d$	$0.208d$	$0.311d$	$0.208d$
	GaussSieve [MV10b]	$0.415d$	$0.208d$	$0.311d$	$0.208d$
	2-level sieve [WLTB11]	$0.384d$	$0.256d$	$0.311d$	$0.208d$
	3-level sieve [ZPH13]	$0.3778d$	$0.283d$	$0.311d$	$0.208d$
	Overlattice sieve [BGJ14]	$0.3774d$	$0.293d$	$0.311d$	$0.208d$
	High-level sieving (Chapter 9)	$0.3774d$	$0.293d$	$0.311d$	$0.208d$
	Hyperplane LSH (Chapter 10)	$0.337d$	$0.208d$	$0.286d$	$0.208d$
	Hypercone LSH (Chapter 11)	$0.298d$	$0.208d$	$0.268d$	$0.208d$
	Cross-polytope LSH (Chapter 12)	$0.298d$	$0.208d$	$0.268d$	$0.208d$
	Hypercone filtering (Chapter 13)	$0.292d$	$0.208d$	$0.265d$	$0.208d$

more practical quantum architectures, and also represents a more conservative approach in determining parameter choices for lattice-based cryptographic primitives that are presumed to be resistant against the potential power of quantum algorithmic attacks.

Outline. The outline of this chapter is as follows. In Section 14.2 we revisit the sieving algorithms from the previous chapters, and describe the effects of quantum search on these algorithms, showing that nearest neighbor search techniques can potentially be combined with quantum search to obtain even better complexities for solving SVP. In Section 14.3 we then give a brief discussion of how quantum searching affects other SVP algorithms, such as sieving algorithms with provable guarantees, and enumeration.

14.2 — Quantum search speed-ups for sieving

14.2.1 – The Nguyễn–Vidick sieve. As described in previous chapters, the basic sieve of Nguyễn and Vidick [NV08] starts by generating a big list L of $n = (4/3)^{d/2+o(d)} \approx 2^{0.208d+o(d)}$ random lattice vectors, and then repeatedly applies a sieve to it to obtain shorter lists L' of shorter lattice vectors. The number of applications of this sieve is polynomial in d, and the bottleneck in the time (and space) complexity comes from the cost of one application of the sieve.

The sieving step basically consists of taking all n vectors in L, and looking at almost all pairwise difference vectors $\boldsymbol{v} - \boldsymbol{w}$ with $\boldsymbol{v}, \boldsymbol{w} \in L$. In other words, for each $\boldsymbol{v} \in L$ we search for vectors $\boldsymbol{w} \in L$ such that $\boldsymbol{v} - \boldsymbol{w}$ is short. For a fixed $\boldsymbol{v}$, this defines a function f which maps almost all vectors $\boldsymbol{w} \in L$ to 0 and a small subset of the vectors to 1. Note that Grover's algorithm can handle multiple solutions efficiently.

Classical complexities. Using a classical search routine to find nearby vectors, the Nguyễn–Vidick sieve has a time complexity quadratic in the initial list size, $\tilde{O}(n^2)$. Together with the polynomial number of iterations that need to be performed, this leads to time and space complexities of $(4/3)^{d+o(d)} \approx 2^{0.415d+o(d)}$ and $(4/3)^{d/2+o(d)} \approx 2^{0.208d+o(d)}$ respectively.

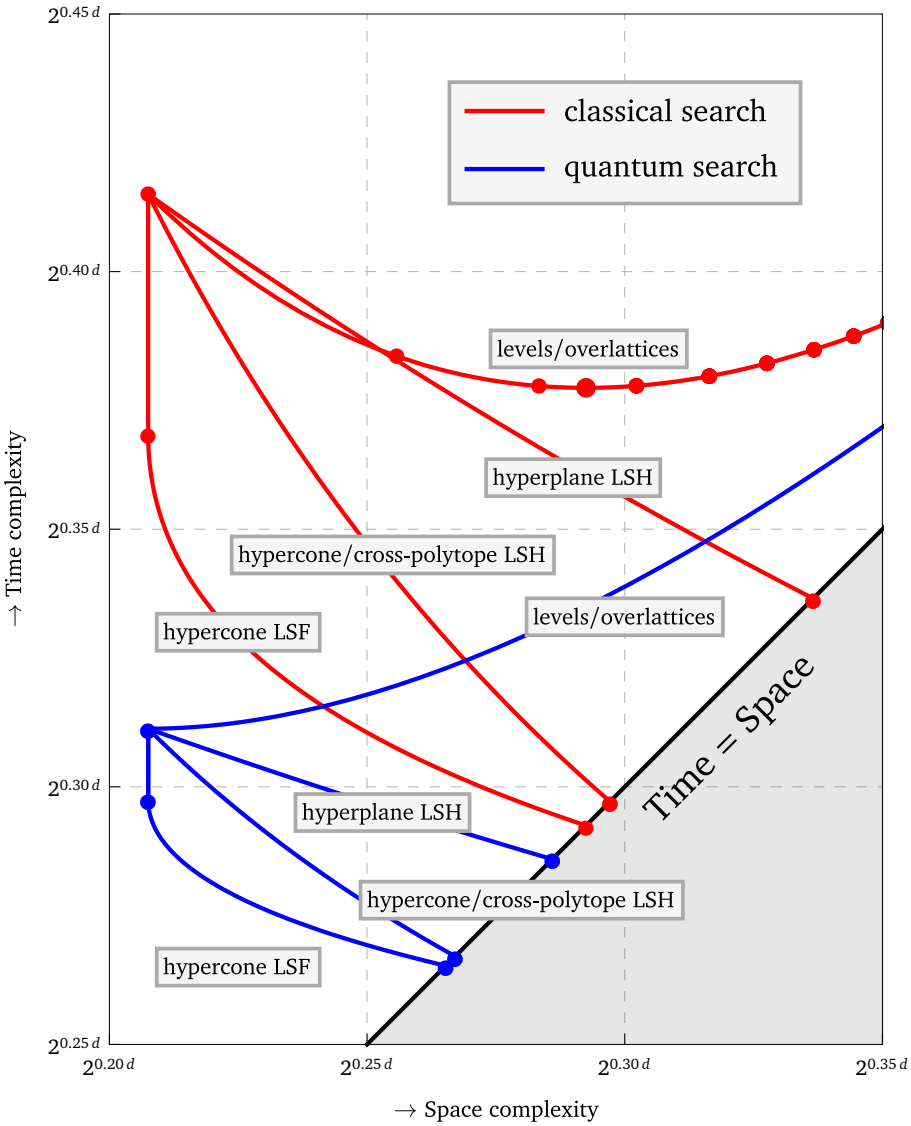


Figure 14.1: The classical space/time trade-offs of various heuristic sieving algorithms from previous chapters (red), and the trade-offs obtained with quantum search applied to these algorithms (blue). For nearest-neighbor techniques, points on the trade-off curves can be turned into a point with the same time complexity but with a space complexity of only $2^{0.208d+o(d)}$ using the ideas of e.g. Algorithm 10.3. Recall that turning a trade-off into a speed-up only works for the Nguyễn–Vidick sieve and not for the GaussSieve.

Quantum complexities. If we use quantum searching to find nearby vectors for a given vector $\boldsymbol{v} \in L$, we may potentially reduce the time complexity of a single search to $\tilde{O}(\sqrt{n})$. In total, this leads to time and space complexities of $(4/3)^{3d/4+o(d)} \approx 2^{0.311d+o(d)}$ and $(4/3)^{d/2+o(d)} \approx 2^{0.208d+o(d)}$ respectively.

In other words, applying quantum search to Nguyễn and Vidick's sieve algorithm leads

to a 25% decrease in the asymptotic exponent of the runtime.

14.2.2 – The GaussSieve. Micciancio and Voulgaris' GaussSieve algorithm [MV10b] does not have provable bounds on the time complexity, but this algorithm is commonly conjectured to have similar asymptotic complexities as the Nguyễn–Vidick sieve: a quadratic complexity in the list size, which is also estimated to be of size at most $n \approx 2^{0.208d+o(d)}$. Using a classical search we estimate the complexities to be the same as the Nguyễn–Vidick sieve, and with quantum searching that would still be the case, with conjectured quantum time and space complexities of $2^{0.311d+o(d)}$ and $2^{0.208d+o(d)}$ respectively.

14.2.3 – The 2-level sieve. To improve upon the time complexity of the algorithm of Nguyễn and Vidick, Wang–Liu–Tian–Bi [WLTB11] introduced a further trade-off between the time complexity and the space complexity. Their algorithm uses two layers of centers of sizes n_1 and n_2 as described in Chapter 9 (n_1 outer lists, each of size n_2) and for a new vector we first find a nearby outer center point, and then search the corresponding list of inner centers for a nearby inner center point.

Classical complexities. The classical space complexity of this algorithm is bounded by $\tilde{O}(n_1 n_2)$, while the time required to find a shortest vector is at most $\tilde{O}(n_1 n_2(n_1 + n_2))$: for each of the $n_1 n_2$ vectors, we perform a brute-force search over the outer list (n_1) and then the inner list (n_2). Optimizing the constants γ_1 and γ_2 leads to $(\gamma_1, \gamma_2) \approx (1.093, 1)$, with an asymptotic time complexity $2^{0.384d+o(d)}$ and a space complexity $2^{0.256d+o(d)}$.

Quantum complexities. By using the quantum search speed-up for searching the two lists of centers, the time complexity is reduced to $\tilde{O}(n_1 n_2(\sqrt{n_1} + \sqrt{n_2}))$, while the space complexity remains the same at $\tilde{O}(n_1 n_2)$. Re-optimizing the constants for a minimum quantum time complexity leads to $(\gamma_1, \gamma_2) \to (\sqrt{2}, 1)$, leading to the same time and space complexities as the quantum-version of the algorithm of Nguyễn and Vidick. The "trade-off" that can be obtained for this algorithm overlaps with the top blue curve in Figure 14.1, which also shows that the time complexity increases with the space complexity, and so there is nothing to be gained compared to Nguyễn and Vidick's basic sieve.

14.2.4 – The 3-level sieve. To further improve upon the classical time complexities of the 1- and 2-level sieves, Zhang–Pan–Hu [ZPH13] analyzed the 3-level sieve, with a further trade-off between the time and space complexity as described in Chapter 9. This algorithm uses three layers of centers, of respective sizes n_1, n_2, n_3.

Classical complexities. The classical time complexity of this algorithm is given by $\tilde{O}(n_1 n_2 n_3(n_1 + n_2 + n_3))$, with a space complexity of $\tilde{O}(n_1 n_2 n_3)$. Optimizing the constants $\gamma_1, \gamma_2, \gamma_3$ with a classical search leads to $(\gamma_1, \gamma_2, \gamma_3) \approx (1.140, 1.067, 1)$, with an asymptotic time complexity of $2^{0.378d+o(d)}$ and a space complexity of $2^{0.283d+o(d)}$.

Quantum complexities. Replacing the classical search with a quantum search subroutine, the time complexity is potentially reduced to $\tilde{O}(n_1 n_2 n_3(\sqrt{n_1} + \sqrt{n_2} + \sqrt{n_3}))$, with the same space complexity as in the classical case. Re-optimizing the constants for a minimum time complexity leads to $(\gamma_1, \gamma_2, \gamma_3) \approx (\sqrt{2}, \sqrt{2}, 1)$, again leading to the same time and space complexities as the quantum-version of the algorithm of Nguyễn and Vidick. As the hidden polynomial factors of the 3-level sieve are much larger than for 1-level sieving,

we again observe that with quantum searching there is no point in using multiple layers.

14.2.5 – The overlattice sieve. The overlattice sieve [BGJ14] works by decomposing the lattice into a sequence of overlattices such that the lattice at the bottom corresponds to the challenge lattice, whereas the lattice at the top corresponds to a lattice where enumerating short vectors is easy due to the almost orthogonal basis vectors. The algorithm begins by enumerating many short vectors in the top lattice and then iteratively moves down through the sequence of lattices by combining short vectors in the overlattice to form short vectors in the lattice directly below it in the sequence.

Considering the computational costs of this algorithm, at any time the algorithm deals with β^d vectors that are divided into α^d buckets with on average β^d/α^d vectors per bucket. These buckets are divided into pairs such that any vector from a bucket and any vector from its paired bucket combine into a lattice vector in the sublattice. Therefore, exactly β^{2d}/α^d combinations need to be made in each iteration. For large d there is the condition on α, β that $\beta\sqrt{1-\alpha^2/4} \geqslant 1 + o(1)$, which means that in high dimensions the best choice is to take $\beta \approx 1/\sqrt{1-\alpha^2/4}$.

Classical complexities. The classical running time of this algorithm is $\tilde{O}(\beta^{2d}/\alpha^d)$, as for each of $\tilde{O}(\beta^d)$ vectors we must go through a bucket containing $\tilde{O}(\beta^d/\alpha^d)$ vectors. The space complexity of this algorithm is $\tilde{O}(\beta^d)$ vectors. Optimizing α and β for the best time complexity gives $\alpha = \sqrt{4/3}$ and $\beta = \sqrt{3/2}$ for an asymptotic time complexity of $2^{0.3774d+o(d)}$ and a space complexity of $2^{0.2925d+o(d)}$.

Quantum complexities. Using quantum search to search for suitable pairs of vectors, the running time may be reduced to $\tilde{O}(\beta^{3d/2}/\alpha^{d/2})$, as for each of the β^d vectors we now do a search over the other vectors in time $\sqrt{\beta^d/\alpha^d}$. The quantum space complexity is still given by β^d. Optimizing the quantum time complexity, we obtain $\alpha \to 1$ and $\beta = \sqrt{4/3}$, which gives a quantum time complexity of $2^{0.311d+o(d)}$ and a space complexity of $2^{0.2075d+o(d)}$. For general α, Figure 14.1 sketches the "trade-off" and shows that indeed both the time and the space are increasing with α. So again, with quantum searching this algorithm is not better than the quantum Nguyễn–Vidick sieve.

14.2.6 – High-level sieving. Chapter 9 discusses leveled sieving with even more layers of centers, and how the classical time and space complexities behave as the number of levels increases. Although rigorously proving that high-level sieving does not lead to better results seems hard, we argued that based on the results for low-level sieving, we may predict what the complexities for high-level sieving are. In particular, all classical trade-offs for leveled sieving were conjectured to overlap with the overlattices approach of Becker–Gama–Joux [BGJ14].

With 2- and 3-level sieving, the quantum trade-offs also overlap with the quantum overlattice trade-off curve, and a natural conjecture would be to say that with quantum searching, also the asymptotic complexities of higher level-sieving will lie on the top blue curve given in Figure 14.1. As we further saw that quantum search applied to the overlattices approach does not lead to better results compared to the Nguyễn–Vidick sieve, we therefore also conjecture that high-level sieving with quantum search does not lead to any useful trade-offs compared to the basic Nguyễn–Vidick sieve.

14.2.7 – Hyperplane LSH. Using hyperplane LSH as described in Chapter 10, the search for nearby vectors can be sped up (classically), at the cost of extra memory for the GaussSieve, and without increasing the memory for the Nguyễn–Vidick sieve. With a number of hash tables $t = 2^{c_t d + o(d)}$ and a parameter α chosen as

$$\alpha = \frac{-1}{c_n}\left[c_t + \max_{\theta \in (\frac{\pi}{3}, \frac{\pi}{2})}\left\{\log_2(\sin\theta) + \frac{c_t}{\gamma_2}\log_2\left(1 - \frac{\theta}{\pi}\right)\right\}\right], \tag{14.1}$$

we are able to guarantee a collision probability for distant vectors of $p_2^* = n^{-\alpha + o(1)}$ while guaranteeing that nearby vectors are almost always found. For one target vector, there are roughly $n^{1-\alpha}$ lattice vectors which collide with this vector in one of the hash tables.

Classical complexities. With a classical search of the list of colliding vectors, the search for nearby vectors costs $n^{1-\alpha}$ time. With a proper balancing of the parameters, this leads to the trade-off curves illustrated in e.g. Figures 10.1 and 14.1. In particular, choosing $c_t \approx 0.1290$ leads to the optimal asymptotic time complexity of $2^{0.337d + o(d)}$.

Quantum complexities. Using quantum search on the set of colliding vectors, we can further reduce the time complexity. The search of the list of candidates can potentially be done in time $\sqrt{n^{1-\alpha}}$, which leads to a total number of comparisons of $\tilde{O}(n^{(3-\alpha)/2})$ with a number of hash computations which is still $\tilde{O}(t \cdot n)$. Performing the numerical optimization of the parameters again, this leads to an optimum at $c_t \approx 0.0784$ with $k \approx 0.1341d$ and a quantum time complexity for solving SVP of $2^{0.286d + o(d)}$. So in this case we do get an improvement over the Nguyễn–Vidick sieve.

Note that it is possible to obtain a trade-off between the quantum time and space complexities, by choosing $c_t \in [0, 0.0784]$. Figure 14.1 shows the resulting trade-off, and how it compares with other classical and quantum trade-offs for sieving.

14.2.8 – Hypercone LSH. As described in Chapter 11, using hypercone LSH rather than hyperplane LSH leads to even better asymptotic classical time and space complexities. Similar to sieving with hyperplane LSH, this algorithm stores n vectors in t hash tables, and the number of colliding vectors in one or more of the hash tables is given by $n^{-\alpha}$ with α now defined as

$$\alpha = \frac{-1}{c_n}\left[c_t + \max_{\theta \in (\frac{\pi}{3}, \frac{\pi}{2})}\left\{\log_2(\sin\theta) - 3c_t \tan^2\tfrac{\theta}{2}\right\}\right]. \tag{14.2}$$

Classical complexities. With a classical search of the list of colliding vectors, the search for nearby vectors takes time $n^{1-\alpha}$. Varying the parameter c_t, this leads to the trade-off curve in Figure 14.1. To minimize the classical time complexity, we choose $c_t \approx 0.0896$ to obtain a time complexity of $2^{0.2972d + o(d)}$.

Quantum complexities. Substituting the quantum search subroutine for the search over colliding vectors, we can potentially reduce the cost of comparing vectors to $\tilde{O}(n^{(3-\alpha)/2})$, while again the number of hash computations remains the same at $\tilde{O}(t \cdot n)$. For varying c_t, this leads to a different trade-off curve illustrated in Figure 14.1. Optimizing for the quantum time complexity, we set $c_t \approx 0.0595$ so that the costs of hashing and comparing vectors are balanced, and the quantum time complexity becomes

$2^{0.2671d+o(d)}$. So again, quantum search can potentially be combined with locality-sensitive hashing in a meaningful way to obtain even better time complexities.

14.2.9 – Cross-polytope LSH. For cross-polytope LSH, we saw in Chapter 12 that we essentially obtain the same asymptotic expressions as for hypercone LSH. The classical trade-off curve overlaps with the trade-off curve for hypercone LSH, and unsurprisingly if we apply quantum search to the cross-polytope sieve, we again obtain a trade-off which coincides with the quantum trade-off curve for hypercone LSH depicted in Figure 14.1. So optimizing for the time complexity, we should again take $c_t \approx 0.0595$ to obtain an asymptotic time complexity for solving SVP in dimension d of $2^{0.2671d+o(d)}$.

14.2.10 – Hypercone LSF. Finally, we obtained the best classical time complexity for solving SVP in high dimensions using hypercone filtering. In this algorithm there are two parameters α, β to choose, where a small α corresponds to *querying* many filters to find nearby vectors, and a small β corresponds to *storing* vectors in many filters. We have a number of filters $t = 2^{c_t d+o(d)}$, a number of vectors $n = 2^{c_n d+o(d)}$, and a filter collision probability $c(x) = 2^{c_x d+o(d)}$ for $x \in \{\alpha, \beta\}$, as described below:

$$c_t = -\tfrac{1}{2}\log_2\left(1 - \tfrac{4}{3}\left(\alpha^2 - \alpha\beta + \beta^2\right)\right), \qquad c_x = \tfrac{1}{2}\log_2\left(1 - x^2\right). \tag{14.3}$$

Classical complexities. To find nearby vectors, we take a vector $\boldsymbol{v}$, compute its relevant filters (cost $2^{(c_t+c_\alpha)d+o(d)}$), and search for other vectors in the same buckets ($2^{(c_n+c_\beta)d+o(d)}$ vectors in each of these buckets). Finally, an insertion/deletion of a vector in the right filters takes time $2^{(c_t+c_\beta)d+o(d)}$. In total, the classical time complexity is $2^{c_{\text{time}}d+o(d)}$ and the space complexity is $2^{c_{\text{space}}d+o(d)}$ with

$$c_{\text{time}} = \max\left\{c_n + c_t + c_\alpha, 2c_n + c_t + c_\alpha + c_\beta, c_n + c_t + c_\beta\right\}, \tag{14.4}$$

$$c_{\text{space}} = \max\left\{c_n + c_t + c_\beta, c_n\right\}. \tag{14.5}$$

Optimizing for the time, this leads to $\alpha = \beta = \frac{1}{2}$ with an asymptotic time complexity of $(3/2)^{d/2+o(d)} \approx 2^{0.292d+o(d)}$, while the optimal trade-off for varying α and β is achieved by varying $\alpha \in [\frac{1}{4}, \frac{1}{2}]$ and fixing $\beta = \frac{1}{2}$, leading to the curve in Figure 14.1. Without increasing the space complexity in the GaussSieve, we can still obtain an asymptotic improvement in the time complexity by taking $\alpha = \frac{1}{4}$ and $\beta = \frac{1}{2}$, with time complexity $(5/3)^{d/2+o(d)} \approx 2^{0.368d+o(d)}$.

Quantum complexities. With quantum search applied to the list of candidate nearby vectors, the cost of comparing a vector to other vectors colliding in one of the filters is potentially reduced from $2^{(c_t+c_n+c_\alpha+c_\beta)d+o(d)}$ to $2^{(c_t+c_n+c_\alpha+c_\beta)d/2+o(d)}$. So the quantum time and space exponents q_{time} and q_{space} are now given by

$$q_{\text{time}} = \max\left\{c_n + c_t + c_\alpha, \tfrac{1}{2}(3c_n + c_t + c_\alpha + c_\beta), c_n + c_t + c_\beta\right\}, \tag{14.6}$$

$$q_{\text{space}} = \max\left\{c_n + c_t + c_\beta, c_n\right\}. \tag{14.7}$$

Re-optimizing the complexities [Duc15], we see that the best quantum time complexity is obtained by taking $\alpha = \beta = \frac{1}{4}\sqrt{3}$, with time and space complexities of $(13/9)^{d/2+o(d)} \approx 2^{0.2653d+o(d)}$. Without increasing the memory, the best time complexity is obtained by setting $\beta = 2\alpha$ and $\alpha = \frac{1}{16}\sqrt{58 - 6\sqrt{65}} = \frac{1}{16}(3\sqrt{5} + \sqrt{13})$. This leads to a quantum

Table 14.2: A comparison of the time and space complexities of sieving algorithms with provable guarantees, and other SVP methods such as enumeration. The top rows all describe provable algorithms for SVP, while the last row describes provable complexities for solving approximate SVP (SVP_δ) with large approximation factors δ with sieving. Details on the quantum search exponents can be found in [LMvdP13, LMvdP15].

	Algorithm	**Classical search**		**Quantum search**	
	Name [References]	$\log_2(\text{Time})$	$\log_2(\text{Space})$	$\log_2(\text{Time})$	$\log_2(\text{Space})$
Provable SVP	Enumeration algorithms [FP85, Kan83]	$\Omega(d \log d)$	$O(\log d)$	$\Omega(d \log d)$	$O(\log d)$
	AKS-sieve [AKS01b, NV08, MV10b, HPS11]	$3.398d$	$1.985d$	$2.672d$	$1.877d$
	ListSieve [MV10b]	$3.199d$	$1.327d$	$2.527d$	$1.351d$
	Voronoi cell algorithm [AEVZ02, MV10a]	$2.000d$	$1.000d$	$2.000d$	$1.000d$
	AKS-sieve-birthday [HPS11, PS09]	$2.648d$	$1.324d$	$1.986d$	$1.324d$
	ListSieve-birthday [PS09]	$2.465d$	$1.233d$	$1.799d$	$1.286d$
	Discrete Gaussian sampling [ADRSD15]	$1.000d$	$0.500d$	$1.000d$	$0.500d$
	(SVP_δ) ListSieve-birthday [LWXZ11, WLW15]	$0.802d$	$0.401d$	$0.602d$	$0.401d$

time complexity of $2^{q_{\text{time}}d+o(d)}$ with $q_{\text{time}} = \frac{1}{2}\log_2(1+\sqrt{65}) - \frac{1}{2}\log_2 6 \approx 0.2975$, where the space complexity in this case is unchanged at $(4/3)^{d/2+o(d)} \approx 2^{0.208d+o(d)}$. These results, and the trade-off for general α and β, are again shown in Figure 14.1. So also quantumly, the best results are obtained with hypercone LSF.

14.3 — Other algorithms

Besides the (heuristic) sieving algorithms considered above, various other (provable) SVP algorithms have also been studied over the last decade, for which quantum searching might also lead to a significant speed-up. We will go through these different methods below, and sketch how quantum searching affects their complexities.

14.3.1 – Provable sieving. Sieving algorithms with provable guarantees of finding shortest vectors on arbitrary lattices (rather than relying on a heuristic assumption on the distribution of lattice points) have also been studied extensively [AKS01b, HPS11, MV10b, NV08, PS09, Vou11]. These algorithms use perturbed versions of lattice vectors rather than the actual lattice vectors in the sieves, which roughly means that a small amount of random noise is added to each lattice point before putting it in the sieve. Without assumptions on the distribution of points in space, this allows us to prove that the event of finding a vector which after reduction with the list results in the vector $\mathbf{0}$, is not much more likely to occur than the event of finding a shortest lattice vector after reducing this perturbed vector with the list. This allows us to prove that the size of the list will continually increase with non-negligible probability. As there are upper bounds on how many vectors can be in the list (similar to the kissing constant, but taking into account the size of the noise vectors) this proves that eventually we will find such collisions of $\mathbf{0}$, and so with high probability we will eventually find a shortest vector in our list.

Classical complexities. Different algorithms have been considered, all exploiting this proof strategy, leading to slightly different asymptotics on the time and space complexities of provable sieving. The best provable sieving method to date in high dimensions, the ListSieve-birthday algorithm [HPS11, PS09], further makes use of the birthday paradox to reduce the number of vectors that are needed in a list to guarantee that combining one pair of vectors leads to a shortest lattice vector. Classically, this algorithm achieves an asymptotic time complexity of $2^{2.465d+o(d)}$ with a space complexity of $2^{1.233d+o(d)}$.

Quantum complexities. Using quantum search, we can reduce the exponent for this algorithm, as well as other provable sieving algorithms, by roughly 25%. Working out the precise details[1], this leads to a quantum time complexity of $2^{1.799d+o(d)}$ and a space complexity of $2^{1.286d+o(d)}$, i.e., a reduction in the time exponent of slightly more than 27%. For other provable sieving algorithms, the quantum speed-up does not lead to better time complexities compared to the quantum ListSieve-birthday, as shown in Table 14.2.

14.3.2 – Enumeration algorithms. In enumeration algorithms (see e.g. [FP85, GNR10, HS07, HS10, Kan83, MW15, Poh81, PS08]), all lattice vectors are enumerated inside a giant ball around the origin that is known to contain at least one lattice vector. Let $\mathcal{L}$ be a lattice with basis $B = \{\mathbf{b}_1, \ldots, \mathbf{b}_d\}$. Consider each lattice vector $\boldsymbol{\nu} \in \mathcal{L}$ as a linear combination of the basis vectors, i.e., $\boldsymbol{\nu} = \sum_i \lambda_i \mathbf{b}_i$. Now, we can represent each lattice vector by its coefficient vector $(\lambda_1, \ldots, \lambda_d)$. We would like to have all combinations of values for $(\lambda_1, \ldots, \lambda_d)$ such that the corresponding vector $\boldsymbol{\nu}$ lies inside the ball. We could try any combination and see if it lies within the ball by computing the norm of the corresponding vector, but there is a smarter way that ensures we only consider vectors that lie within the ball and none that lie outside.

To this end, enumeration algorithms search from right to left, by identifying all values for λ_d such that there might exist $\lambda'_1, \ldots, \lambda'_{d-1}$ such that the vector corresponding to $(\lambda'_1, \ldots, \lambda'_{d-1}, \lambda_d)$ lies in the ball. To identify these values $\lambda'_1, \ldots, \lambda'_{d-1}$, enumeration algorithms use the Gram-Schmidt orthogonalization of the lattice basis as well as the projection of lattice vectors. Then, for each of these possible values for λ_d, the enumeration algorithm considers all possible values for λ_{d-1} and repeats the process until it reaches possible values for λ_1. This leads to a search which is serial in nature, as each value of λ_d will lead to different possible values for λ_{d-1} and so forth. Unfortunately, we can only really apply the quantum search algorithm to problems where the list of objects to be searched is known in advance.

One might suggest to forego the smart way to find short vectors and just search all combinations of $(\lambda_1, \ldots, \lambda_d)$ with appropriate upper and lower bounds on the different λ_i's. Then it becomes possible to apply quantum search, since we now have a predetermined list of vectors and just need to compute the norm of each vector. However, it is doubtful that this will result in a faster algorithm, because the heuristic changes by Gama–Nguyễn–Regev [GNR10] have reduced the running time of enumeration dramatically (roughly by a factor $2^{d/2}$) and these changes only complicate the search area further by changing the ball to an ellipsoid. There seems to be no simple way to apply quantum search to the enumeration algorithms that are currently used in practice, but perhaps the algorithms can be modified in some way.

[1] Technical details on this result, as well as the complexities obtained by applying quantum searching to other provable sieving algorithms, can be found in [LMvdP13, LMvdP15].

14.3.3 – The Voronoi cell algorithm. Consider a set of points in the Euclidean space. For any given point in this set, its Voronoi cell is defined as the region that contains all points that lie closer to this point than to any of the other points in the set. Now, given a Voronoi cell, we define a relevant vector to be any vector in the set whose removal from the set will change this particular Voronoi cell. If we pick our lattice as the set and we consider the Voronoi cell around the zero vector, then any shortest vector is also a relevant vector. Furthermore, given the relevant vectors of the Voronoi cell we can solve the closest vector problem in $2^{2d+o(d)}$ time.

So how can we compute the relevant vectors of the Voronoi cell of a lattice $\mathcal{L}$? Micciancio and Voulgaris [MV10a] show that this can be done by solving $2^d - 1$ instances of CVP in the lattice $2\mathcal{L} := \{2\boldsymbol{v} : \boldsymbol{v} \in \mathcal{L}\}$. However, in order to solve CVP we would need the relevant vectors which means we are back to our original problem. Micciancio and Voulgaris show that these instances of CVP can also be solved by solving several related CVP instances in a lattice of lower rank. They give a basic and an optimized version of the algorithm. The basic version only uses LLL as preprocessing and solves all these related CVP instances in the lower rank lattice separately. As a consequence, the basic algorithm runs in time $2^{3.5d+o(d)}$ and in space $2^{d+o(d)}$. The optimized algorithm uses a stronger preprocessing for the lattice basis, which takes exponential time. But since the most expensive part is the computation of the Voronoi relevant vectors, this extra preprocessing time does not increase the asymptotic running time as it is executed only once. In fact, having the reduced basis decreases the asymptotic running time to $\tilde{O}(2^{3d})$. Furthermore, the optimized algorithm employs a trick that allows it to reduce 2^k CVP instances in a lattice of rank k to a single instance of an enumeration problem related to the same lattice. The optimized algorithm solves CVP in time $\tilde{O}(2^{2d})$ using $\tilde{O}(2^d)$ space.

Now, in the basic algorithm, it would be possible to speed up the routine that solves CVP given the Voronoi relevant vectors using a quantum computer. It would also be possible to speed up the routine that removes non-relevant vectors from the list of relevant vectors using a quantum computer. Combining these two changes gives a quantum algorithm with an asymptotic running time $\tilde{O}(2^{2.5d})$, which is still slower than the optimized classical algorithm. It is not possible to apply these same speed-ups to the optimized algorithm due to the aforementioned trick with the enumeration problem. The algorithm to solve this enumeration problem makes use of a priority queue, which means the search is not trivially parallelized. Once again, there does not seem to be a simple way to apply quantum search to this special enumeration algorithm. However, it may be possible that the algorithm can be modified in such a way that quantum search can be applied.

14.3.4 – Discrete Gaussian sampling. A very recent method for finding shortest vectors in lattices is based on sampling and combining lattice vectors sampled from a discrete Gaussian on the lattice. Given a lattice, a discrete Gaussian distribution on the lattice is what you might expect it to be; the probability of sampling a non-lattice vector is 0, and the probability of sampling a lattice vector $\boldsymbol{x}$ is proportional to $\exp -O(\|\boldsymbol{x}\|^2)$, comparable to a regular multivariate Gaussian distribution. These discrete Gaussians are commonly used in lattice-based cryptographic primitives, such as lattice-based signatures [DDLL13, Lyu12], and it is well-known that sampling from a discrete Gaussian distribution with a very large standard deviation (above the *smoothing parameter* [GPV08, Mic04, MR07]) is easy, while sampling from a distribution with a small standard deviation (often sampling short vectors) is hard.

The idea of Aggarwal–Dadush–Regev–Stephens-Davidowitz [ADRSD15] to solve SVP with discrete Gaussian sampling is as follows. First, many vectors are sampled from a discrete Gaussian with a large standard deviation. Then, to find shorter and shorter lattice vectors, list vectors are combined and averaged to obtain samples from a Gaussian with a smaller standard deviation. More precisely, two samples $\boldsymbol{v}_1, \boldsymbol{v}_2$ from a discrete Gaussian with width σ can be combined and averaged to obtain a sample $\boldsymbol{v}' = (\boldsymbol{v}_1 + \boldsymbol{v}_2)/2$ which follows a discrete Gaussian with width $\sigma/\sqrt{2}$ using clever rejection sampling techniques. To be able to combine list vectors, they need to be in the same coset of $2\mathcal{L}$, which means that the algorithm needs to store 2^d buckets corresponding to the 2^d cosets of $2\mathcal{L}$, and within each bucket vectors are combined to obtain samples from a more narrow Gaussian distribution. Overall, this leads to time and space complexities of $2^{d+o(d)}$ for provably solving SVP.

This algorithm is actually quite similar to the overlattice sieve of Becker–Gama–Joux [BGJ14] which we discussed in Section 14.2.5. Vectors are already stored in buckets, and so searching for vectors in the same coset is not costly at all. In this algorithm, the number of vectors in each bucket is even sub-exponential in d, so a quantum search speed-up does not seem to bring down the asymptotic time or space complexities at all. Due to the number of cosets of $2\mathcal{L}$ (namely 2^d), this algorithm seems (classically and quantumly) bound by a time and space complexity of at least $2^{d+o(d)}$, which it achieves.

14.3.5 – Provably solving approximate SVP with sieving. While most sieving algorithms are concerned with finding exact solutions to the shortest vector problem, in many cryptographic applications finding a *short* (rather than a *shortest*) vector in the lattice also suffices to break the scheme. In 2011, Liu–Wang–Xu–Zheng [LWXZ11] analyzed the impact of this relaxation of SVP (solving SVP_δ with approximation factor $\delta > 1$) on lattice sieving algorithms. In particular, they analyzed the ListSieve-birthday algorithm considered in Section 14.3.1, taking into account the fact that an approximate solution may suffice. Their algorithm [LWXZ11, Algorithm 1] is effectively identical to the original ListSieve-birthday algorithm of Pujol and Stehlé [PS09], but parameters may be chosen differently to obtain better complexities for approximate SVP.

Classical complexities. Intuitively, the effect of large δ can be understood as that the impact of the use of perturbed lattice vectors in the sieves (instead of actual lattice vectors) becomes less and less. In the limit of large δ, the impact of perturbations disappears (although it still guarantees correctness of the algorithm), and we get the same upper bound on the list size of $2^{0.401d+o(d)}$ as for the GaussSieve [MV10b]: $2^{0.208d+o(d)}$ is a lower bound and estimate for the kissing constant in high dimensions, while $2^{0.401d+o(d)}$ is a proven asymptotic upper bound on the same quantity. Since the runtime of the sieve remains quadratic in the list size, this leads to a time complexity of $2^{0.802d+o(d)}$, where the order term $o(d)$ disappears as $d \to \infty$ and $\delta \to \infty$.

Quantum complexities. As expected, using quantum search in the ListSieve-birthday algorithm again leads to a gain in the exponent of 25%; a single search can be done in time $\tilde{O}(\sqrt{n})$, leading to a total time complexity of $\tilde{O}(n^{3/2})$ rather than $\tilde{O}(n^2)$. For solving approximate-SVP with large δ, this means that the quantum time and space complexities become $2^{0.602d+o(d)}$ and $2^{0.401d+o(d)}$ respectively.

CHAPTER 15

Conclusions and open problems

Conclusions

We finally conclude the second part by taking another look at the research questions presented in Chapter 8, and how these questions were answered in Chapters 9–14. After summarizing these contributions, we will end this chapter with some open problems that appeared throughout the second part, which may be interesting for future research.

Q1. Can leveled sieving be further improved?

In Chapter 9 we revisited the leveled sieving results of Nguyễn and Vidick [NV08], Wang–Liu–Tian–Bi [WLTB11], and Zhang–Pan–Hu [ZPH13]. We established an apparent pattern in the time and space complexities that led us to a natural conjecture on the complexities of sieving with even more layers of centers. If this conjectured pattern is indeed correct, then with 4-level sieving we may be able to slightly improve upon the time complexity of 3-level sieving, and higher-level sieving does not lead to further improvements. With 4-level sieving, the best time and space complexities match the asymptotics of the overlattices approach [BGJ14], and so sieving can be slightly improved compared to 3-level sieving, but ultimately does not lead to better asymptotics than previous work.

Q2. Can known techniques from NNS be used to speed up sieving?

In Chapters 10 and 11 we then focused on how existing techniques from nearest neighbor searching (locality-sensitive hashing) can be applied to sieving. This combination of sieving and LSH was previously considered and dismissed by Nguyễn and Vidick [NV08], but with an average-case analysis rather than a worst-case analysis, and using LSH techniques aimed at the angular distance rather than the Euclidean distance, we that it may be possible to improve upon previous results with LSH.

In Chapter 10 we first considered the method of Charikar [Cha02], which is very efficient in practice due to the low cost of computing hashes, and has a reasonable asymptotic performance. This led to an improvement in the asymptotic time complexity to $2^{0.3366d+o(d)}$. For the Nguyễn–Vidick sieve, we saw that this time complexity can be achieved without increasing the space complexity, leading to a significant theoretical improvement over previous trade-offs. For the more practical GaussSieve it seems that we cannot avoid increasing the space complexity to $2^{0.3366d+o(d)}$ as well. Experiments showed that with the GaussSieve-based HashSieve, we obtain an improvement over the GaussSieve already in moderate dimensions, and the increase in the space complexity is smaller than asymptotics suggest, and may be further reduced with probing.

In Chapter 11 we then considered the recent spherical cap LSH method of [AINR14, AR15a], which has a better asymptotic performance but appears to be less practical due to a high subexponential cost of computing hashes. We saw how this method can be extended to all of $\mathbb{R}^d$, dividing the space into hypercones, and we saw that combining this technique with sieving leads to a theoretical time complexity of $2^{0.2972d+o(d)}$ for solving SVP in dimension d. However, this method appears to be less suitable for practical applications, even though asymptotically it is superior to hyperplane LSH, and we speculated that the ultimate goal may be to design an LSH primitive achieving the asymptotics of hypercone LSH with the practicality of hyperplane LSH.

Q3. Can existing NNS techniques be improved (and applied to sieving)?

In Chapter 12 we set out to find a method which does achieve both an optimal asymptotic performance and a low overhead of computing hashes, and this resulted in cross-polytope LSH. This method was previously proposed by Terasawa and Tanaka [TT07], who already showed it appears to perform well in practice, but no asymptotic guarantees were provided then. We showed that indeed it achieves the same asymptotic performance as spherical and hypercone LSH, while being (almost) as practical as hyperplane LSH. Moreover, we saw that this method is very suitable for sieving on ideal lattices, due to the predictability of hash values of shifted vectors. In the context of sieving, this method already seems to outperform hyperplane LSH in moderate dimensions, and on ideal lattices it is much better than hyperplane LSH and the ideal GaussSieve. As cross-polytope LSH matches lower bounds on LSH and has a small polynomial overhead, this appears to be the best that we can do when combining sieving with LSH.

Q4. Can new NNS primitives be designed specifically for settings like sieving?

In Chapter 13 we showed that this is not quite the end of the story, as these lower bounds on LSH all make a small assumption: all these lower bounds only hold if the collision probabilities are assumed to be sufficiently large ($2^{o(d)}$). After all, if the collision probabilities were smaller, then we would need an exponential number of hash tables, which is impractical if $n = 2^{o(d)}$ (low-density). In sieving however we have a list of size $n = 2^{\Theta(d)}$ (high-density) and exponentially small collision probabilities may not be that bad after all. Indeed, slightly deviating from the LSH framework, we saw that a method similar to hypercone LSH is able to outperform hypercone LSH in high-density settings, which for sieving implies an asymptotic time complexity of only $2^{0.2925d+o(d)}$. Experiments showed that this method is also very practical with a low polynomial overhead, and may even be more practical in high dimensions than cross-polytope LSH.

Q5. How do quantum algorithms affect the asymptotic complexities of sieving?

In Chapter 14 we finally studied how quantum algorithms (and in particular Grover's quantum search algorithm [Gro96]) affect the asymptotic performance of sieving. Informally, quantum searching roughly reduces the exponent for sieving with naive searching by 25%, and the speed-up becomes smaller as the searching becomes more sophisticated with nearest-neighbor search techniques. For leveled sieving we saw that quantum searching does not improve the asymptotic complexities, while for sieving with LSH and other NNS methods we obtained various asymptotic speed-ups, with the best complexities obtained using the hypercone filtering approach of Chapter 13. For that algorithm, the asymptotic quantum complexity is potentially $2^{0.265d+o(d)}$ in high dimensions.

Open problems

The search for answers to the above research questions led to new questions as well, and below we state some open problems which may be a topic for future work.

Q6. What else can be said about the leveled sieving approach?

a. Can the various conjectures in Chapter 9 be proven?

b. Why do the complexities match those of the overlattices algorithm?

c. What does the algorithm in the limit of large k mean?

Especially in Chapter 9, one could say we raised more questions than we answered. The conjectured complexities for high-level sieving seem to answer the first research question, but as these are only conjectures, an open problem is to actually prove these results. The analysis in Chapter 9 may serve as a guideline for constructing a proof, as for instance we outlined which parameters seem optimal. One possible method to construct proofs for high-level sieving would be to use induction on the different layers, as we repeatedly apply the same procedure of partitioning a large ball into slightly smaller balls.

Besides these missing proofs, we saw that there seems to be a relation between leveled sieving and the overlattices approach; both the optimized complexities and the time/memory trade-offs of leveled sieving overlap with the trade-off curve for the algorithm of [BGJ14]. Why do these complexities overlap? Are the algorithms essentially the same, when viewed in the right way? And can leveled sieving with large k be somehow interpreted as using the overlattices approach with $\alpha = \beta = \sqrt{2}$?

More generally, the algorithm in the limit of large k shows how, using $2^{d/2+o(d)}$ lattice vectors as centers in all the different layers, we can essentially find nearby center vectors to a lattice vector in time $2^{o(d)}$. The use of many layers of centers, each time slightly decreasing the radius of the balls, also very much looks like the Nguyễn–Vidick sieve itself which also uses many iterations of applying a sieve to the data set, slightly shrinking the radius, until in the end a shortest vector is found. Can these ideas be combined to build a more efficient leveled sieve?

Q7. Can even better methods be designed for high-density NNS settings?

As argued above, bounds on locality-sensitive hashing all make an assumption which does not necessarily have to hold in the context of sieving, and so it is unclear what is the best performance that can possibly be achieved just by combining e.g. the Nguyễn–Vidick sieve or GaussSieve with nearest-neighbor search techniques. Is it possible to further improve upon the method described in Chapter 13? Or can we prove lower bounds on the complexity of any method for solving high-density NNS, which prove (or disprove) that the filtering method of Chapter 13 is optimal?

Q8. Can techniques from NNS be used for other lattice algorithms as well?

In this work we focused on sieving algorithms with provable guarantees under a certain heuristic assumption (variants of the Nguyễn–Vidick sieve), and on practical algorithms with no provable guarantees on the time complexity (variants of the GaussSieve). If one wants to be sure that the algorithm solves SVP on arbitrary lattices and lattice bases using a certain amount of time and memory, without making any heuristic assumptions, then

currently the best time complexity for SVP is based on discrete Gaussian sampling (time $2^{d+o(d)}$), with sieving running far behind ($2^{2.465d+o(d)}$). Can similar nearest-neighbor techniques be used to obtain provable speed-ups to sieving algorithms as well? Or can these techniques perhaps be used to speed up other algorithms, like the Voronoi cell algorithm [AEVZ02, MV10a]?

One particularly interesting direction to investigate with NNS seems to be the closest vector problem with preprocessing (CVPP): given a basis of a lattice, preprocess it in such a way that when given a non-lattice target vector later, one can quickly find the nearest lattice vector. This problem seems closely related to the general nearest-neighbor search problem, with the main differences being that (a) in CVPP, the data set has an infinite size (all lattice vectors), and (b) in CVPP, the data set is known to be very structured. For NNS, it is known that the problem can be solved in time $n^{o(1)}$ using more memory [Kap15, Pan06], and an interesting question would be to study whether these techniques from NNS can also be used to significantly speed up CVPP.[1]

Q9. What is the cross-over point between enumeration and sieving?

Already since the early 1980s, enumeration has had the status of the most practical algorithm for solving exact SVP in moderate to high dimensions. When sieving algorithms finally seemed to come close to enumeration around 2008-2010 [NV08, MV10b], the extreme pruning improvement to enumeration [GNR10] made sure that enumeration remained the best algorithm for solving SVP for several years to come. With the improvements to sieving presented in the second part of this work, the balance seems to shift more and more towards sieving. Micciancio and Walter [MW15] previously estimated the cross-over point of enumeration and sieving with hyperplane LSH to lie at dimensions $d > 700$, and with better preprocessing for enumeration this point would shift to $d > 1800$. As we can do better with cross-polytope LSH and hypercone filtering, this cross-over point may go down a lot. Practical experiments with sieving may further show if sieving can already be performed in dimensions as high as 130 or 140, and if in those dimensions it is faster than enumeration.

Q10. What can be done with less memory?

Finally, we focused on trade-offs between the time and the memory, where we used more memory (or the same amount of memory, for the Nguyễn–Vidick sieve) so that the asymptotic time complexity would be better. In each of these algorithms the memory complexity was at least $(4/3)^{d/2+o(d)} \approx 2^{0.208d+o(d)}$, and this complexity seems to be a lower bound inherent to the sieving method of combining pairs of vectors to find shorter vectors. However, in high dimensions even this space complexity may already be too much. Can trade-offs also be obtained in the other direction? Can we use less memory, perhaps using more time? Can we perhaps combine ideas from enumeration and sieving, where we consider combinations of multiple lattice vectors to find shorter vectors, and achieve a smaller asymptotic space complexity? More generally: given a certain asymptotic space complexity, what is the best asymptotic time complexity that we can achieve with this amount of memory?

[1] Note that one of the main applications of CVPP is as a subroutine within enumeration, as described in e.g. [GNR10, Open questions]. An efficient CVPP routine could be used to prune the branches in lower levels of the enumeration tree, which might lead to significant improvements in practice for enumeration.

Summary

To be completed.

Curriculum Vitae

To be completed.

Bibliography

[AB06] Nagaraj Prasanth Anthapadmanabhan and Alexander Barg. Random binary fingerprinting codes for arbitrarily sized coalitions. In *ISIT*, pages 351–355, 2006. doi:10.1109/ISIT.2006.261612.

[ABD08] Nagaraj Prasanth Anthapadmanabhan, Alexander Barg, and Ilya Dumer. On the fingerprinting capacity under the marking assumption. *IEEE Transactions on Information Theory*, 54(6):2678–2689, 2008. doi:10.1109/TIT.2008.921859.

[ABN08] Barry C. Arnold, Narayanaswamy Balakrishnan, and Haikady N. Nagaraja. *A First Course in Order Statistics*. SIAM, 2008. URL: http://epubs.siam.org/doi/book/10.1137/1.9780898719062.

[Ach01] Dimitris Achlioptas. Database-friendly random projections. In *PODS*, pages 274–281, 2001. doi:10.1145/375551.375608.

[Ach03] Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003. doi:http://dx.doi.org/10.1016/S0022-0000(03)00025-4.

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC*, pages 284–293, 1997. doi:10.1145/258533.258604.

[ADL11a] Rudolf Ahlswede, Christian Deppe, and Vladimir S. Lebedev. Bounds for threshold and majority group testing. In *ISIT*, pages 69–73, 2011. doi:10.1109/ISIT.2011.6034222.

[ADL11b] Rudolf Ahlswede, Christian Deppe, and Vladimir S. Lebedev. Majority group testing with density tests. In *ISIT*, pages 326–330, 2011. doi:10.1109/ISIT.2011.6034139.

[ADL13] Rudolf Ahlswede, Christian Deppe, and Vladimir S. Lebedev. Threshold and majority group testing. In *Information Theory, Combinatorics, and Search Theory*, pages 488–508, 2013. doi:10.1007/978-3-642-36899-8_24.

[ADRSD15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time via discrete Gaussian sampling. In *STOC*, pages 733–742, 2015. doi:10.1145/2746539.2746606.

[AEVZ02] Erik Agrell, Thomas Eriksson, Alexander Vardy, and Kenneth Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, Aug 2002. doi:10.1109/TIT.2002.800499.

[AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468, 2006. doi:10.1109/FOCS.2006.49.

[AI08] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008. doi:10.1145/1327452.1327494.

[AIL+15] Alexandr Andoni, Piotr Indyk, **Thijs Laarhoven**, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In *NIPS*, 2015. URL: http://arxiv.org/abs/1509.02897.

[AINR14] Alexandr Andoni, Piotr Indyk, Huy Lê Nguyễn, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *SODA*, pages 1018–1028, 2014. doi:10.1137/1.9781611973402.76.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996. doi:10.1145/237814.237838.

[Ajt98] Miklós Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions. In *STOC*, pages 10–19, 1998. doi:10.1145/276698.276705.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9, 1999. doi:10.1007/3-540-48523-6_1.

[AKS01a] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. An overview of the sieve algorithm for the shortest lattice vector problem. In *CALC*, pages 1–3, 2001. doi:10.1007/3-540-44670-2_1.

[AKS01b] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001. doi:10.1145/380752.380857.

[Amb04] Andris Ambainis. Quantum walk algorithm for element distinctness. In *FOCS*, pages 22–31, 2004. doi:10.1109/FOCS.2004.54.

[Ami10] Ehsan Amiri. *Fingerprinting Codes: Higher Rates, Quick Accusations*. PhD thesis, Simon Fraser University, 2010. URL: http://summit.sfu.ca/item/11454.

[And09] Alexandr Andoni. *Nearest Neighbor Search: the Old, the New, and the Impossible*. PhD thesis, Massachusetts Institute of Technology, 2009. URL: http://hdl.handle.net/1721.1/55090.

[AR03] Dorit Aharonov and Oded Regev. A lattice problem in quantum NP. In *FOCS*, pages 210–219, 2003. doi:10.1109/SFCS.2003.1238195.

[AR15a] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *STOC*, pages 793–801, 2015. doi:10.1145/2746539.2746553.

[AR15b] Alexandr Andoni and Ilya Razenshteyn. Tight lower bounds for data-dependent locality-sensitive hashing. *Manuscript*, pages 1–15, 2015. URL: http://www.ilyaraz.org/publications.html.

[AS72] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Formulas*. Dover Publications, 1972. URL: http://people.math.sfu.ca/~cbm/aands/toc.htm.

[AS12] George K. Atia and Venkatesh Saligrama. Boolean compressed sensing and noisy group testing. *IEEE Transactions on Information Theory*, 58(3):1880–1901, 2012. doi:10.1109/TIT.2011.2178156.

[AT09] Ehsan Amiri and Gábor Tardos. High rate fingerprinting codes and the fingerprinting capacity. In *SODA*, pages 336–345, 2009. URL: http://dl.acm.org/citation.cfm?id=1496808.

[AZ10] Emmanuel Abbe and Lizhong Zheng. Linear universal decoding for compound channels. *IEEE Transactions on Information Theory*, 56(12):5999–6013, 2010. doi:10.1109/TIT.2010.2080910.

[BBBV97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997. doi:10.1137/S0097539796300933.

[BBD09] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors. *Post-Quantum Cryptography*. Springer, 2009. URL: http://www.springerlink.com/content/978-3-540-88701-0.

[BBHT98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4–5):493–505, 1998. URL: http://onlinelibrary.wiley.com/doi/10.1002/(SICI)1521-3978(199806)46:4/5%3C493::AID-PROP493%3E3.0.CO;2-P/abstract.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and **Thijs Laarhoven**. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, 2016.

[BDH+01] Harry Buhrman, Christoph Dürr, Mark Heiligman, Peter Høyer, Frédéric Magniez, Miklos Santha, and Ronald de Wolf. Quantum algorithms for element distinctness. In *CCC*, pages 131–137, 2001. doi:10.1109/CCC.2001.933880.

[Ber09] Daniel J. Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete? In *SHARCS*, page 105, 2009. URL: http://repository.tue.nl/663426.

[BGJ14] Anja Becker, Nicolas Gama, and Antoine Joux. A sieve algorithm based on overlattices. In *ANTS*, pages 49–70, 2014. doi:10.1112/S1461157014000229.

[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*, pages 1–14, 2015. URL: http://eprint.iacr.org/2015/522.

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002. URL: http://arxiv.org/abs/quant-ph/0005055.

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In *LATIN*, pages 163–169, 1998. doi:10.1007/BFb0054319.

[BK04] George R. Blakley and Gregory Kabatiansky. Random coding technique for digital fingerprinting codes: Fighting two pirates revisited. In *ISIT*, page 202, 2004. doi:10.1109/ISIT.2004.1365239.

[BL15a] Anja Becker and **Thijs Laarhoven**. Efficient (ideal) lattice sieving using cross-polytope LSH. *Cryptology ePrint Archive, Report 2015/823*, pages 1–25, 2015. URL: http://eprint.iacr.org/2015/823.

[BL15b] Daniel J. Bernstein and Tanja Lange. Post-quantum cryptography, 2015. URL: http://pqcrypto.org/.

[BLS13] Jay Bartroff, Tze Leung Lai, and Mei-Chiung Shih. *Sequential Experimentation in Clinical Trials*. Springer, 2013. doi:10.1007/978-1-4614-6114-2.

[BNvdP14] Joppe W. Bos, Michael Naehrig, and Joop van de Pol. Sieving for shortest vectors in ideal lattices: a practical perspective. *Cryptology ePrint Archive, Report 2014/880*, pages 1–23, 2014. URL: http://eprint.iacr.org/2014/880.

[BPS01] Omer Berkman, Michal Parnas, and Jiří Sgall. Efficient dynamic traitor tracing. *SIAM Journal on Computing*, 30(6):1802–1828, 2001. doi:10.1137/S0097539700367984.

[BS98] Dan Boneh and James Shaw. Collusion-secure fingerprinting for digital data. *IEEE Transactions on Information Theory*, 44(5):1897–1905, 1998. doi:10.1109/18.705568.

[BS12] Waldemar Berchtold and Marcel Schäfer. Performance and code length optimization of joint decoding Tardos fingerprinting. In *MMSec*, pages 27–32, 2012. doi:10.1145/2361407.2361412.

[BŠ11] Dion Boesten and Boris Škorić. Asymptotic fingerprinting capacity for non-binary alphabets. In *IH*, pages 1–13, 2011. doi:10.1007/978-3-642-24178-9_1.

[BŠ12] Dion Boesten and Boris Škorić. Asymptotic fingerprinting capacity in the combined digit model. In *IH*, pages 255–268, 2012. doi:10.1007/978-3-642-36373-3_17.

[BT08] Oded Blayer and Tamir Tassa. Improved versions of Tardos' fingerprinting scheme. *Designs, Codes and Cryptography*, 48(1):79–103, 2008. doi:10.1007/s10623-008-9200-z.

[BUV14] Mark Bun, Jonathan Ullman, and Salil Vadhan. Fingerprinting codes and the price of approximate differential privacy. In *STOC*, pages 1–10, 2014. doi:10.1145/2591796.2591877.

[CCB+13] Chun Lam Chan, Sheng Cai, Mayank Bakshi, Sidharth Jaggi, and Venkatesh Saligrama. Stochastic threshold group testing. In *ITW*, pages 1–5, 2013. doi:10.1109/ITW.2013.6691242.

[CCJS11] Chun Lam Chan, Pak Hou Che, Sidharth Jaggi, and Venkatesh Saligrama. Non-adaptive probabilistic group testing with noisy measurements: Near-optimal bounds with efficient algorithms. In *ALLERTON*, pages 1832–1839, 2011. doi:10.1109/Allerton.2011.6120391.

[CGS14] Peter Campbell, Michael Groves, and Dan Shepherd. Soliloquy: A cautionary tale. In *ETSI 2nd Quantum-Safe Crypto Workshop*, pages 1–9, 2014. URL: http://docbox.etsi.org/Workshop/2014/201410_CRYPTO/S07_Systems_and_Attacks/S07_Groves_Annex.pdf.

[Cha02] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002. doi:10.1145/509907.509965.

[Che72] Herman Chernoff. *Sequential Analysis and Optimal Design*. SIAM, 1972. URL: http://books.google.nl/books?id=jXjRteblBLoC&dq=Sequential+Analysis+and+Optimal+Design&lr=&hl=nl&source=gbs_navlinks_s.

[Che13] Mahdi Cheraghchi. Improved constructions for non-adaptive threshold group testing. *Algorithmica*, 67(3):384–417, 2013. doi:10.1007/s00453-013-9754-7.

[CHKV09] Mahdi Cheraghchi, Ali Hormati, Amin Karbasi, and Martin Vetterli. Compressed sensing with probabilistic measurements: A group testing solution. In *ALLERTON*, pages 30–35, 2009. doi:10.1109/ALLERTON.2009.5394829.

[CHKV11] Mahdi Cheraghchi, Ali Hormati, Amin Karbasi, and Martin Vetterli. Group testing with probabilistic tests: Theory, design and application. *IEEE Transactions on Information Theory*, 57(10):7057–7067, 2011. doi:10.1109/TIT.2011.2148691.

[Cho94] Kwok Pui Choi. On the medians of gamma distributions and an equation of Ramanujan. *Proceedings of the American Mathematical Society*, 121(1):245–251, 1994. URL: http://www.jstor.org/stable/2160389.

[CHS10] Charles J. Colbourn, Daniel Horsley, and Violet R. Syrotiuk. Frameproof codes and compressive sensing. In *ALLERTON*, pages 985–990, 2010. doi:10.1109/ALLERTON.2010.5707016.

[CJS14] Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014. doi:10.1515/jmc-2012-0016.

[CJSA12] Chun Lam Chan, Sidharth Jaggi, Venkatesh Saligrama, and Samar Agnihotri. Non-adaptive group testing: Explicit bounds and novel algorithms. In *ISIT*, pages 1837–1841, 2012. doi:10.1109/ISIT.2012.6283597.

[CJSA14] Chun Lam Chan, Sidharth Jaggi, Venkatesh Saligrama, and Samar Agnihotri. Non-adaptive group testing: Explicit bounds and novel algorithms. *IEEE Transactions on Information Theory*, 60(5):3019–3035, 2014. doi:10.1109/TIT.2014.2310477.

[CN11] Yuanmi Chen and Phong Q. Nguyễn. BKZ 2.0: Better lattice security estimates. In *ASIACRYPT*, pages 1–20, 2011. doi:10.1007/978-3-642-25385-0_1.

[CNFS05] Josep Cotrina-Navau, Marcel Fernández, and Miguel Soriano. A family of collusion 2-secure codes. In *IH*, pages 387–397, 2005. doi:10.1007/11558859_28.

[CS99] John H. Conway and Neil J.A. Sloane. *Sphere packings, lattices and groups*. Springer, 1999. URL: http://books.google.nl/books/about/Sphere_Packings_Lattices_and_Groups.html?id=upYwZ6cQumoC&redir_esc=y.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (2nd Edition)*. Wiley, 2006. URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471241954.html.

[CvD10] Andrew M. Childs and Wim van Dam. Quantum algorithms for algebraic problems. *Reviews of Modern Physics*, 82(1):1–52, 2010. doi:10.1103/RevModPhys.82.1.

[CXFF09] Ana Charpentier, Fuchun Xie, Caroline Fontaine, and Teddy Furon. Expectation maximization decoding of Tardos probabilistic fingerprinting code. In *SPIE Media Forensics and Security*, pages 1–15, 2009. doi:10.1117/12.806034.

[Dam06] Peter Damaschke. Threshold group testing. In *General Theory of Information Transfer and Combinatorics*, pages 707–718, 2006. doi:10.1007/11889342_45.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In *CRYPTO*, pages 40–56, 2013. doi:10.1007/978-3-642-40041-4_3.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi:10.1109/TIT.1976.1055638.

[DHPG13] Mathieu Desoubeaux, Cédric Herzet, William Puech, and Gaëtan Le Guelvouit. Enhanced blind decoding of Tardos codes with new MAP-based functions. In *MMSP*, pages 283–288, 2013. doi:10.1109/MMSP.2013.6659302.

[Dor43] Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943. URL: http://www.jstor.org/stable/2235930.

[DTTZ14] Cynthia Dwork, Kunal Talwar, Abhradeep Thakurta, and Li Zhang. Analyze Gauss: Optimal bounds for privacy-preserving principal component analysis. In *STOC*, pages 11–20, 2014. doi:10.1145/2591796.2591883.

[Dub10] Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179, Aug 2010. doi:10.1109/TIT.2010.2050814.

[Duc15] Léo Ducas. Private communication, 2015.

[FBB+14] Robert Fitzpatrick, Christian Bischof, Johannes Buchmann, Özgür Dagdelen, Florian Göpfert, Artur Mariano, and Bo-Yin Yang. Tuning GaussSieve for speed. In *LATINCRYPT*, pages 288–305, 2014. URL: http://eprint.iacr.org/2014/788.

[FD14] Teddy Furon and Mathieu Desoubeaux. Tardos codes for real. In *WIFS*, pages 24–29, 2014. doi:10.1109/WIFS.2014.7084298.

[FGC08] Teddy Furon, Arnaud Guyader, and Frédéric Cérou. On the design and optimization of Tardos probabilistic fingerprinting codes. In *IH*, pages 341–356, 2008. doi:10.1007/978-3-540-88961-8_24.

[FGC12] Teddy Furon, Arnaud Guyader, and Frédéric Cérou. Decoding fingerprints using the Markov chain Monte Carlo method. In *WIFS*, pages 187–192, 2012. doi:10.1109/WIFS.2012.6412647.

[FP85] Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice. *Mathematics of Computation*, 44(170):463–471, 1985. URL: http://www.jstor.org/stable/2007966.

[FPF09a] Teddy Furon and Luis Pérez-Freire. EM decoding of Tardos traitor tracing codes. In *MMSec*, pages 99–106, 2009. doi:10.1145/1597817.1597835.

[FPF09b] Teddy Furon and Luis Pérez-Freire. Worst case attacks against binary probabilistic traitor tracing codes. In *WIFS*, pages 56–60, 2009. doi:10.1109/WIFS.2009.5386484.

[FPFGC09] Teddy Furon, Luis Pérez-Freire, Arnaud Guyader, and Frédéric Cérou. Estimating the minimal length of Tardos code. In *IH*, pages 176–190, 2009. doi:10.1007/978-3-642-04431-1_13.

[FT99] Amos Fiat and Tamir Tassa. Dynamic traitor tracing. In *CRYPTO*, pages 354–371, 1999. doi:10.1007/3-540-48405-1_23.

[FT01] Amos Fiat and Tamir Tassa. Dynamic traitor tracing. *Journal of Cryptology*, 14(3):211–223, 2001. doi:10.1007/s00145-001-0006-7.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009. doi:10.1145/1536414.1536440.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, pages 1–17, 2013. doi:10.1007/978-3-642-38348-9_1.

[GL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996. URL: https://jhupbooks.press.jhu.edu/content/matrix-computations.

[GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical Review Letters*, 100(16):160501, 2008. doi:10.1103/PhysRevLett.100.160501.

[GNR10] Nicolas Gama, Phong Q. Nguyễn, and Oded Regev. Lattice enumeration using extreme pruning. In *EUROCRYPT*, pages 257–278, 2010. doi:10.1007/978-3-642-13190-5_13.

[Gov04] Zakkula Govindarajulu. *Sequential Statistics*. World Scientific, 2004. URL: http://www.worldscientific.com/worldscibooks/10.1142/5575.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008. doi:10.1145/1374376.1374407.

[GR04] Lov K. Grover and Terry Rudolph. How significant are the known collision and element distinctness quantum algorithms. *Quantum Information and Computation*, 4(3):201–206, 2004. URL: http://dl.acm.org/citation.cfm?id=2011617.2011622.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC*, pages 212–219, 1996. doi:10.1145/237814.237866.

[Hal02] Sean Hallgren. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. In *STOC*, pages 653–658, 2002. doi:10.1145/509907.510001.

[HHGPW10] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, and William Whyte. *Practical Lattice-Based Cryptography: NTRUEncrypt and NTRUSign*, chapter 11, pages 349–390. In Nguyên and Vallée [NV10], 2010. doi:10.1007/978-3-642-02295-1_11.

[HM09a] Yen-Wei Huang and Pierre Moulin. Capacity-achieving fingerprint decoding. In *WIFS*, pages 51–55, 2009. doi:10.1109/WIFS.2009.5386483.

[HM09b] Yen-Wei Huang and Pierre Moulin. Saddle-point solution of the fingerprinting capacity game under the marking assumption. In *ISIT*, pages 2256–2260, 2009. doi:10.1109/ISIT.2009.5205882.

[HM10] Yen-Wei Huang and Pierre Moulin. Maximin optimality of the arcsine fingerprinting distribution and the interleaving attack for large coalitions. In *WIFS*, pages 1–6, 2010. doi:10.1109/WIFS.2010.5711451.

[HM12a] Yen-Wei Huang and Pierre Moulin. On fingerprinting capacity games for arbitrary alphabets and their asymptotics. In *ISIT*, pages 2571–2575, 2012. doi:10.1109/ISIT.2012.6283982.

[HM12b] Yen-Wei Huang and Pierre Moulin. On the saddle-point solution and the large-coalition asymptotics of fingerprinting games. *IEEE Transactions on Information Forensics and Security*, 7(1):160–175, 2012. doi:10.1109/TIFS.2011.2168212.

[HM14] Yen-Wei Huang and Pierre Moulin. On the fingerprinting capacity games for arbitrary alphabets and their asymptotics. *IEEE Transactions on Information Forensics and Security*, 9(9):1477–1490, 2014. doi:10.1109/TIFS.2014.2338739.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998. doi:10.1007/BFb0054868.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In *IWCC*, pages 159–190, 2011. doi:10.1007/978-3-642-20901-7_10.

[HS07] Guillaume Hanrot and Damien Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm. In *CRYPTO*, pages 170–186, 2007. doi:10.1007/978-3-540-74143-5_10.

[HS10] Guillaume Hanrot and Damien Stehlé. A complete worst-case analysis of Kannan's shortest lattice vector algorithm. *Manuscript*, pages 1–34, 2010. URL: http://perso.ens-lyon.fr/damien.stehle/KANNAN_EXTENDED.html.

[Hwa76] Frank K. Hwang. Group testing with a dilution effect. *Biometrika*, 63(3):671–680, 1976. doi:10.1093/biomet/63.3.671.

[IKMT14] Tsukasa Ishiguro, Shinsaku Kiyomoto, Yutaka Miyake, and Tsuyoshi Takagi. Parallel Gauss Sieve algorithm: Solving the SVP challenge over a 128-dimensional ideal lattice. In *PKC*, pages 411–428, 2014. doi:10.1007/978-3-642-54631-0_24.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998. doi:10.1145/276698.276876.

[IŠO14] Sarah Ibrahimi, Boris Škorić, and Jan-Jaap Oosterwijk. Riding the saddle point: Asymptotics of the capacity-achieving simple decoder for bias-based traitor tracing. *EURASIP Journal on Information Security*, 1(12):1–11, 2014. doi:10.1186/s13635-014-0012-6.

[JT00] Christoper Jennison and Bruce W. Turnbull. *Group Sequential Methods with Applications to Clinical Trials*. Chapman and Hall, 2000. URL: http://www.crcpress.com/product/isbn/9780849303166.

[Kan83] Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *STOC*, pages 193–206, 1983. doi:10.1145/800061.808749.

[Kap15] Michael Kapralov. Smooth tradeoffs between insert and query complexity in nearest neighbor search. In *PODS*, pages 329–342, 2015. doi:10.1145/2745754.2745761.

[KB80] Rob Kaas and J.M. Buhrman. Mean, median and mode in binomial distributions. *Statistica Neerlandica*, 34(1):13–18, 1980. doi:10.1111/j.1467-9574.1980.tb00681.x.

[KHN+08] Takashi Kitagawa, Manabu Hagiwara, Koji Nuida, Hajime Watanabe, and Hideki Imai. A group testing based deterministic tracing algorithm for a short random fingerprint code. In *ISITA*, pages 1–5, 2008. doi:10.1109/ISITA.2008.4895500.

[Kho04a] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. In *FOCS*, pages 126–135, 2004. doi:10.1109/FOCS.2004.31.

[Kho04b] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *Journal of the ACM*, 52(5):789–808, 2004. doi:10.1145/1089023.1089027.

[Kle00] Philip Klein. Finding the closest lattice vector when it's unusually close. In *SODA*, pages 937–941, 2000. URL: http://dl.acm.org/citation.cfm?id=338661.

[Kle14] Thorsten Kleinjung. Private communication, 2014.

[Kup05] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005. doi:10.1137/S0097539703436345.

[Kup13] Greg Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. In *TQC*, pages 1–10, 2013. doi:10.4230/LIPIcs.TQC.2013.20.

[Kur13] Minoru Kuribayashi. Bias equalizer for binary probabilistic fingerprinting codes. In *IH*, pages 269–283, 2013. doi:10.1007/978-3-642-36373-3.

[Laa13a] **Thijs Laarhoven**. Dynamic traitor tracing schemes, revisited. In *WIFS*, pages 191–196, 2013. doi:10.1109/WIFS.2013.6707817.

[Laa13b] **Thijs Laarhoven**. Efficient probabilistic group testing based on traitor tracing. In *ALLERTON*, pages 1358–1365, 2013. doi:10.1109/Allerton.2013.6736699.

[Laa14] **Thijs Laarhoven**. Capacities and capacity-achieving decoders for various fingerprinting games. In *IH&MMSec*, pages 123–134, 2014. doi:10.1145/2600918.2600925.

[Laa15a] **Thijs Laarhoven**. Asymptotics of fingerprinting and group testing: Capacity-achieving log-likelihood decoders. *EURASIP Journal on Information Security*, 2015. URL: http://arxiv.org/abs/1404.2825.

[Laa15b] **Thijs Laarhoven**. Asymptotics of fingerprinting and group testing: Tight bounds from channel capacities. *IEEE Transactions on Information Forensics and Security*, 10(9):1967–1980, 2015. doi:10.1109/TIFS.2015.2440190.

[Laa15c] **Thijs Laarhoven**. Optimal sequential fingerprinting: Wald vs. Tardos. In *IH&MMSec*, pages 97–107, 2015. doi:10.1145/2756601.2756603.

[Laa15d] **Thijs Laarhoven**. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO*, pages 3–22, 2015. doi:10.1007/978-3-662-47989-6_1.

[LDR+13] **Thijs Laarhoven**, Jeroen Doumen, Peter Roelse, Boris Škorić, and Benne de Weger. Dynamic Tardos traitor tracing schemes. *IEEE Transactions on Information Theory*, 59(7):4230–4242, 2013. doi:10.1109/TIT.2013.2251756.

[LdW13] **Thijs Laarhoven** and Benne de Weger. Discrete distributions in the Tardos scheme, revisited. In *IH&MMSec*, pages 13–18, 2013. doi:10.1145/2482513.2482533.

[LdW14] **Thijs Laarhoven** and Benne de Weger. Optimal symmetric Tardos traitor tracing schemes. *Designs, Codes and Cryptography*, 71(1):83–103, 2014. doi:10.1007/s10623-012-9718-y.

[LdW15] **Thijs Laarhoven** and Benne de Weger. Faster sieving for shortest lattice vectors using spherical locality-sensitive hashing. In *LATINCRYPT*, pages 101–118, 2015. doi:10.1007/978-3-319-22174-8_6.

[Leb10] Vladimir S. Lebedev. Separating codes and a new combinatorial search model. *Problems of Information Transmission*, 46(1):1–6, 2010. doi:10.1134/S0032946010010011.

[LHC06] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *KDD*, pages 287–296, 2006. doi:10.1145/1150402.1150436.

[LJW+07] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007. URL: http://dl.acm.org/citation.cfm?id=1325958.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. doi:10.1007/BF01457454.

[LM00] Beatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics*, 28(5):1302–1338, 2000. doi:10.1214/aos/1015957395.

[LMvdP13] **Thijs Laarhoven**, Michele Mosca, and Joop van de Pol. Solving the shortest vector problem in lattices faster using quantum search. In *PQCrypto*, pages 83–101, 2013. doi:10.1007/978-3-642-38616-9_6.

[LMvdP15] **Thijs Laarhoven**, Michele Mosca, and Joop van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2):375–400, 2015. doi:10.1007/s10623-015-0067-5.

[LOD12] **Thijs Laarhoven**, Jan-Jaap Oosterwijk, and Jeroen Doumen. Dynamic traitor tracing for arbitrary alphabets: Divide and conquer. In *WIFS*, pages 240–245, 2012. doi:10.1109/WIFS.2012.6412656.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *CT-RSA*, pages 319–339, 2011. doi:10.1007/978-3-642-19074-2_21.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, pages 1–23, 2010. doi:10.1007/978-3-642-13190-5_1.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In *EUROCRYPT*, pages 35–54, 2013. doi:10.1007/978-3-642-38348-9_3.

[Lud03] Christoph Ludwig. A faster lattice reduction method using quantum search. In *ISAAC*, pages 199–208, 2003. doi:10.1007/978-3-540-24587-2_22.

[LvdPdW12] **Thijs Laarhoven**, Joop van de Pol, and Benne de Weger. Solving hard lattice problems and the security of lattice-based cryptosystems. *Cryptology ePrint Archive, Report 2012/533*, pages 1–43, 2012. URL: http://eprint.iacr.org/2012/533.

[LWXZ11] Mingjie Liu, Xiaoyun Wang, Guangwu Xu, and Xuexin Zheng. Shortest lattice vectors in the presence of gaps. *Cryptology ePrint Archive, Report 2011/039*, pages 1–22, 2011. URL: http://eprint.iacr.org/2011/139.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, pages 738–755, 2012. doi:10.1007/978-3-642-29011-4_43.

[Mal78] Mikhail B. Malyutov. The separating property of random matrices. *Mathematical notes of the Academy of Sciences of the USSR*, 23(1):84–91, 1978. doi:10.1007/BF01104893.

[Mar90] Alexander L. Marshak. Asymptotic representation of the weights in the Gauss quadrature formula. *Mathematical notes of the Academy of Sciences of the USSR*, 47(4):354–358, 1990. doi:10.1007/BF01163817.

[McK15] Brendan D. McKay. Expected centered entropy of the binomial distribution, 2015. URL: http://mathoverflow.net/a/200287/11259.

[MF11a] Peter Meerwald and Teddy Furon. Group testing meets traitor tracing. In *ICASSP*, pages 4204–4207, 2011. doi:10.1109/ICASSP.2011.5947280.

[MF11b] Peter Meerwald and Teddy Furon. Towards joint Tardos decoding: The 'Don Quixote' algorithm. In *IH*, pages 28–42, 2011. doi:10.1007/978-3-642-24178-9_3.

[MF12] Peter Meerwald and Teddy Furon. Toward practical joint decoding of binary Tardos fingerprinting codes. *IEEE Transactions on Information Forensics and Security*, 7(4):1168–1180, 2012. doi:10.1109/TIFS.2012.2195655.

[Mic98] Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *FOCS*, pages 92–98, 1998. doi:10.1109/SFCS.1998.743432.

[Mic04] Daniele Micciancio. Almost perfect lattices, the covering radius problem, and applications to Ajtai's connection factor. *SIAM Journal of Computing*, 34(1):118–169, 2004. doi:10.1137/S0097539703433511.

[MLB15] Artur Mariano, **Thijs Laarhoven**, and Christian Bischof. Parallel (probable) lock-free HashSieve: a practical sieving algorithm for the SVP. In *ICPP*, 2015. URL: https://eprint.iacr.org/2015/041.

[MNP07] Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower bounds on locality sensitive hashing. *SIAM Journal of Discrete Mathematics*, 21(4):930–935, 2007. doi:10.1137/050646858.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015. doi:10.1007/978-3-662-46800-5_9.

[MODB14] Artur Mariano, Özgür Dagdelen, and Christian Bischof. A comprehensive empirical comparison of parallel ListSieve and GaussSieve. In *Euro-Par 2014*, pages 48–59, 2014. doi:10.1007/978-3-319-14325-5_5.

[Mos09] Michele Mosca. *Encyclopedia of Complexity and Systems Science*, chapter Quantum Algorithms, pages 7088–7118. Springer, 2009. doi:10.1007/978-0-387-30440-3_423.

[Mou08] Pierre Moulin. Universal fingerprinting: Capacity and random-coding exponents. *arXiv:0801.3837 [cs.IT]*, pages 1–69, 2008. URL: http://arxiv.org/abs/0801.3837.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007. doi:10.1137/S0097539705447360.

[MR09] Daniele Micciancio and Oded Regev. *Lattice-Based Cryptography*, chapter 5, pages 147–191. In Bernstein et al. [BBD09], 2009. doi:10.1007/978-3-540-88702-7_5.

[MS09] Nitis Mukhopadhyay and Basil M. De Silva. *Sequential Methods and Their Applications*. CRC Press, 2009. URL: http://books.google.nl/books?hl=nl&lr=&id=PYNKPyqiyfYC&oi=fnd&pg=PA1&ots=1n0-icxnBl&sig=Hp-Lgb-hrni714fzrzDpSDvse2Y#v=onepage&q&f=false.

[MS11] Benjamin Milde and Michael Schneider. A parallel implementation of GaussSieve for the shortest vector problem in lattices. In *PACT*, pages 452–458, 2011. doi:10.1007/978-3-642-23178-0_40.

[MTB14] Artur Mariano, Shahar Timnat, and Christian Bischof. Lock-free GaussSieve for linear speedups in parallel high performance SVP calculation. In *SBAC-PAD*, pages 278–285, 2014. doi:10.1109/SBAC-PAD.2014.18.

[MV10a] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *STOC*, pages 351–358, 2010. doi:10.1145/1806689.1806739.

[MV10b] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. In *SODA*, pages 1468–1480, 2010. URL: http://dl.acm.org/citation.cfm?id=1873720.

[MW15] Daniele Micciancio and Michael Walter. Fast lattice point enumeration with minimal overhead. In *SODA*, pages 276–294, 2015. doi:10.1137/1.9781611973730.21.

[NFH+09] Koji Nuida, Satoshi Fujitsu, Manabu Hagiwara, Takashi Kitagawa, Hajime Watanabe, Kazuto Ogawa, and Hideki Imai. An improvement of discrete Tardos fingerprinting codes. *Designs, Codes and Cryptography*, 52(3):339–362, 2009. doi:10.1007/s10623-009-9285-z.

[Ngu14] Huy Lê Nguyễn. *Algorithms for High Dimensional Data*. PhD thesis, Princeton University, 2014. URL: http://arks.princeton.edu/ark:/88435/dsp01b8515q61f.

[NHWI07] Koji Nuida, Manabu Hagiwara, Hajime Watanabe, and Hideki Imai. Optimization of Tardos's fingerprinting codes in a viewpoint of memory amount. In *IH*, pages 279–293, 2007. doi:10.1007/978-3-540-77370-2_19.

[NP33] Jerzy Neyman and Egon Sharpe Pearson. On the problem of the most efficient tests of statistical hypotheses. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 231(694-706):289–337, 1933. doi:10.1098/rsta.1933.0009.

[Nui09] Koji Nuida. An improvement of short 2-secure fingerprint codes strongly avoiding false-positive. In *IH*, pages 161–175, 2009. doi:10.1007/978-3-642-04431-1_12.

[Nui10] Koji Nuida. Short collusion-secure fingerprint codes against three pirates. In *IH*, pages 86–102, 2010. doi:10.1007/978-3-642-16435-4_8.

[Nui12] Koji Nuida. Short collusion-secure fingerprint codes against three pirates. *International Journal of Information Security*, 11(2):85–102, 2012. doi:10.1007/s10207-012-0155-8.

[NV08] Phong Q. Nguyễn and Thomas Vidick. Sieve algorithms for the shortest vector problem are practical. *Journal of Mathematical Cryptology*, 2(2):181–207, 2008. doi:10.1515/JMC.2008.009.

[NV10] Phong Q. Nguyễn and Brigitte Vallée, editors. *The LLL Algorithm: Survey and Applications*. Springer, 2010. doi:10.1007/978-3-642-02295-1.

[ODL14] Jan-Jaap Oosterwijk, Jeroen Doumen, and **Thijs Laarhoven**. Tuple decoders for traitor tracing schemes. In *SPIE Media Watermarking, Security, and Forensics*, pages 1–21, 2014. doi:10.1117/12.2037659.

[OŠD13] Jan-Jaap Oosterwijk, Boris Škorić, and Jeroen Doumen. Optimal suspicion functions for Tardos traitor tracing schemes. In *IH&MMSec*, pages 19–28, 2013. doi:10.1145/2482513.2482527.

[OŠD15] Jan-Jaap Oosterwijk, Boris Škorić, and Jeroen Doumen. A capacity-achieving simple decoder for bias-based traitor tracing schemes. *IEEE Transactions on Information Theory*, 61(7):3882–3900, 2015. doi:10.1109/TIT.2015.2428250.

[OWZ11] Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality sensitive hashing (except when q is tiny). In *ICS*, pages 276–283, 2011. URL: http://conference.itcs.tsinghua.edu.cn/ICS2011/content/papers/2.html.

[OWZ14] Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Transactions on Computation Theory*, 6(1):5:1–5:13, 2014. doi:10.1145/2578221.

[Pan06] Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *SODA*, pages 1186–1195, 2006. URL: http://dl.acm.org/citation.cfm?id=1109688.

[PFF09] Luis Pérez-Freire and Teddy Furon. Blind decoder for binary probabilistic traitor tracing codes. In *WIFS*, pages 46–50, 2009. doi:10.1109/WIFS.2009.5386486.

[Poh81] Michael E. Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *ACM SIGSAM Bulletin*, 15(1):37–44, 1981. doi:10.1145/1089242.1089247.

[PS08] Xavier Pujol and Damien Stehlé. Rigorous and efficient short lattice vectors enumeration. In *ASIACRYPT*, pages 390–405, 2008. doi:10.1007/978-3-540-89255-7_24.

[PS09] Xavier Pujol and Damien Stehlé. Solving the shortest lattice vector problem in time $2^{2.465n}$. *Cryptology ePrint Archive, Report 2009/605*, pages 1–7, 2009. URL: http://eprint.iacr.org/2009/605.

[PS15] Thomas Plantard and Michael Schneider. Ideal SVP challenge, 2015. URL: http://www.latticechallenge.org/ideallattice-challenge/.

[Raz14] Ilya Razenshteyn. Beyond locality-sensitive hashing. Master's thesis, MIT, 2014. URL: http://hdl.handle.net/1721.1/89862.

[Reg04a] Oded Regev. Quantum computation and lattice problems. *SIAM Journal on Computing*, 33(3):738–760, 2004. doi:10.1137/S0097539703440678.

[Reg04b] Oded Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. *arXiv:quant-ph/0406151*, pages 1–7, 2004. URL: http://arxiv.org/abs/quant-ph/0406151.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005. doi:10.1145/1060590.1060603.

[Reg06] Oded Regev. Lattice-based cryptography. In *CRYPTO*, pages 131–141, 2006. doi:10.1007/11818175_8.

[Reg10] Oded Regev. The learning with errors problem (invited survey). In *CCC*, pages 191–204, 2010. doi:10.1109/CCC.2010.26.

[Roe11] Peter Roelse. Dynamic subtree tracing and its application in pay-TV systems. *International Journal of Information Security*, 10(3):173–187, 2011. doi:10.1007/s10207-011-0126-5.

[RS10] Markus Rückert and Michael Schneider. Estimating the security of lattice-based cryptosystems. *Cryptology ePrint Archive, Report 2010/137*, pages 1–33, 2010. URL: http://eprint.iacr.org/2010/137.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. doi:10.1145/359340.359342.

[SBM05] Anelia Somekh-Baruch and Neri Merhav. On the capacity game of private fingerprinting systems under collusion attacks. *IEEE Transactions on Information Theory*, 51(3):884–899, 2005. doi:10.1109/TIT.2004.842702.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2–3):201–224, 1987. doi:10.1016/0304-3975(87)90064-8.

[Sch03] Hans-Georg Schaathun. Fighting two pirates. In *AAECC*, pages 71–78, 2003. doi:10.1007/3-540-44828-4_9.

[Sch04] Hans-Georg Schaathun. Fighting three pirates with scattering codes. In *ISIT*, page 203, 2004. doi:10.1109/ISIT.2004.1365240.

[Sch08] Hans-Georg Schaathun. On the assumption of equal contributions in fingerprinting. *IEEE Transactions on Information Forensics and Security*, 3(3):569–572, 2008. doi:10.1109/TIFS.2008.926991.

[Sch11] Michael Schneider. Analysis of Gauss-Sieve for solving the shortest vector problem in lattices. In *WALCOM*, pages 89–97, 2011. doi:10.1007/978-3-642-19094-0_11.

[Sch13] Michael Schneider. Sieving for short vectors in ideal lattices. In *AFRICACRYPT*, pages 375–391, 2013. doi:10.1007/978-3-642-38553-7_22.

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(2–3):181–199, 1994. doi:10.1007/BF01581144.

[Seb85] András Sebő. On two random search problems. *Journal of Statistical Planning and Inference*, 11(1):23–31, 1985. doi:10.1016/0378-3758(85)90022-9.

[SG15] Michael Schneider and Nicolas Gama. SVP challenge, 2015. URL: http://latticechallenge.org/svp-challenge/.

[Sho94] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *FOCS*, pages 124–134, 1994. doi:10.1109/SFCS.1994.365700.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. doi:10.1137/S0097539795293172.

[Sho15] Victor Shoup. NTL: A library for doing number theory, 2015. URL: http://www.shoup.net/ntl/.

[Sie85] David Siegmund. *Sequential Analysis, Tests and Confidence Intervals*. Springer, 1985. doi:10.1007/978-1-4757-1862-1.

[Sim14] Antonino Simone. *Error probabilities in Tardos codes*. PhD thesis, Eindhoven University of Technology, 2014. doi:10.6100/IR774667.

[SJ10] Dino Sejdinovic and Oliver Johnson. Note on noisy group testing: Asymptotic bounds and belief propagation reconstruction. In *ALLERTON*, pages 998–1003, 2010. doi:10.1109/ALLERTON.2010.5707018.

[SLH12] Malcolm Slaney, Yury Lifshits, and Junfeng He. Optimal parameters for locality-sensitive hashing. *Proceedings of the IEEE*, 100(9):2604–2623, 2012. doi:10.1109/JPROC.2012.2193849.

[SM12] Jamie Smith and Michele Mosca. *Handbook of Natural Computing*, chapter Algorithms for Quantum Computers, pages 1451–1492. Springer, 2012. doi:10.1007/978-3-540-92910-9_43.

[SNW03] Reihaneh Safavi-Naini and Yejing Wang. Sequential traitor tracing. *IEEE Transactions on Information Theory*, 49(5):1319–1326, 2003. doi:10.1109/TIT.2003.810629.

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In *EUROCRYPT*, pages 27–47, 2011. doi:10.1007/978-3-642-20465-4_4.

[ŠKC08] Boris Škorić, Stefan Katzenbeisser, and Mehmet U. Celik. Symmetric Tardos fingerprinting codes for arbitrary alphabet sizes. *Designs, Codes and Cryptography*, 46(2):137–166, 2008. doi:10.1007/s10623-007-9142-x.

[Ško14] Boris Škorić. Private communication, 2014.

[Ško15] Boris Škorić. Tally-based simple decoders for traitor tracing and group testing. *IEEE Transactions on Information Forensics and Security*, 10(6):1221–1233, 2015. doi:10.1109/TIFS.2015.2403575.

[ŠKSC09] Boris Škorić, Stefan Katzenbeisser, Hans-Georg Schaathun, and Mehmet U. Celik. Tardos fingerprinting codes in the combined digit model. In *WIFS*, pages 41–45, 2009. doi:10.1109/WIFS.2009.5386485.

[ŠKSC11] Boris Škorić, Stefan Katzenbeisser, Hans-Georg Schaathun, and Mehmet U. Celik. Tardos fingerprinting codes in the combined digit model. *IEEE Transactions on Information Forensics and Security*, 6(3):906–919, 2011. doi:10.1109/TIFS.2011.2116783.

[ŠO15] Boris Škorić and Jan-Jaap Oosterwijk. Binary and q-ary Tardos codes, revisited. *Designs, Codes and Cryptography*, 74(1):75–111, 2015. doi:10.1007/s10623-013-9842-3.

[SŠ11] Antonino Simone and Boris Škorić. Asymptotically false-positive-maximizing attack on non-binary Tardos codes. In *IH*, pages 14–27, 2011. doi:10.1007/978-3-642-24178-9_2.

[SŠ12] Antonino Simone and Boris Škorić. Accusation probabilities in Tardos codes: Beyond the Gaussian approximation. *Designs, Codes and Cryptography*, 63(3):379–412, 2012. doi:10.1007/s10623-011-9563-4.

[SŠ14] Antonino Simone and Boris Škorić. False positive probabilities in q-ary Tardos codes: comparison of attacks. *Designs, Codes and Cryptography*, 75(3):519–542, 2014. doi:10.1007/s10623-014-9937-5.

[ŠVCT08] Boris Škorić, Tatiana U. Vladimirova, Mehmet U. Celik, and Joop C. Talstra. Tardos fingerprinting is better than we thought. *IEEE Transactions on Information Theory*, 54(8):3663–3676, 2008. doi:10.1109/TIT.2008.926307.

[SU15] Thomas Steinke and Jonathan Ullman. Interactive fingerprinting codes and the hardness of preventing false discovery. In *COLT*, pages 1588–1628, 2015. URL: http://jmlr.org/proceedings/papers/v40/Steinke15.html.

[SvTW00] Douglas R. Stinson, Tran van Trung, and Ruizhong Wei. Secure frameproof codes, key distribution patterns, group testing algorithms and related structures. *Journal of Statistical Planning and Inference*, 86(2):595–617, 2000. doi:10.1016/S0378-3758(99)00131-7.

[SW49] Milton Sobel and Abraham Wald. A sequential decision procedure for choosing one of three hypotheses concerning the unknown mean of a normal distribution. *The Annals of Mathematical Statistics*, 20(4):502–522, 1949. URL: http://www.jstor.org/stable/2236307.

[Sze75] Gábor Szegő. *Orthogonal Polynomials*. American Mathematical Society, 1975. URL: http://books.google.nl/books?id=3hcW8HBh7gsC.

[TABB02] Luca G. Tallini, Sulaiman Al-Bassam, and Bella Bose. On the capacity and codes for the Z-channel. In *ISIT*, page 422, 2002. doi:10.1109/ISIT.2002.1023694.

[Tar03] Gábor Tardos. Optimal probabilistic fingerprint codes. In *STOC*, pages 116–125, 2003. doi:10.1145/780542.780561.

[Tar08] Gábor Tardos. Optimal probabilistic fingerprint codes. *Journal of the ACM*, 55(2):1–24, 2008. doi:10.1145/1346330.1346335.

[Tar10] Gábor Tardos. Capacity of collusion secure fingerprinting – a tradeoff between rate and efficiency. In *IH*, pages 81–85, 2010. doi:10.1007/978-3-642-16435-4_7.

[Tas05] Tamir Tassa. Low bandwidth dynamic traitor tracing schemes. *Journal of Cryptology*, 18(2):167–183, 2005. doi:10.1007/s00145-004-0214-z.

[TT07] Kengo Terasawa and Yuzuru Tanaka. Spherical LSH for approximate nearest neighbor search on unit hypersphere. In *WADS*, pages 27–38, 2007. doi:10.1007/978-3-540-73951-7_4.

[TWWL03] Wade Trappe, Min Wu, Z. Jane Wang, and K.J. Ray Liu. Anti-collusion fingerprinting for multimedia. *IEEE Transactions on Signal Processing*, 51(4):1069–1087, 2003. doi:10.1109/TSP.2003.809378.

[Ull13] Jonathan Ullman. Answering $n^{2+o(1)}$ counting queries with differential privacy is hard. In *STOC*, pages 361–370, 2013. doi:10.1145/2488608.2488653.

[vdP11] Joop van de Pol. Lattice-based cryptography. Master's thesis, Eindhoven University of Technology, 2011. URL: http://www.cs.bris.ac.uk/home/csjhvdp/.

[vdPS13] Joop van de Pol and Nigel Smart. Estimating key sizes for high dimensional lattice-based systems. In *IMACC*, pages 290–303, 2013. doi:10.1007/978-3-642-45239-0_17.

[Vou11] Panagiotis Voulgaris. *Algorithms For The Closest And Shortest Vector Problems On General Lattices*. PhD thesis, University of California at San Diego, 2011. URL: http://escholarship.org/uc/item/4zt7x45z.

[Wal45] Abraham Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945. URL: http://www.jstor.org/stable/2235829.

[Wal47] Abraham Wald. *Sequential Analysis*. Wiley and Sons, 1947. URL: http://www.getcited.org/pub/101175034.

[WG86] George Barrie Wetherill and Kevin D. Glazebrook. *Sequential Methods in Statistics (3rd Edition)*. Chapman and Hall, 1986. URL: http://eprints.lancs.ac.uk/47982/.

[WLTB11] Xiaoyun Wang, Mingjie Liu, Chengliang Tian, and Jingguo Bi. Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem. In *ASIACCS*, pages 1–9, 2011. doi:10.1145/1966913.1966915.

[WLW15] Wei Wei, Mingjie Liu, and Xiaoyun Wang. Finding shortest lattice vectors in the presence of gaps. In *CT-RSA*, pages 239–257, 2015. doi:10.1007/978-3-319-16715-2_13.

[WW48] Abraham Wald and Jacob Wolfowitz. Optimum character of the sequential probability ratio test. *The Annals of Mathematical Statistics*, 19(3):326–339, 1948. doi:10.1214/aoms/1177730197.

[ZPH13] Feng Zhang, Yanbin Pan, and Gengran Hu. A three-level sieve algorithm for the shortest vector problem. In *SAC*, pages 29–47, 2013. doi:10.1007/978-3-662-43414-7_2.